ARTICLE Communicated by Terrence Sejnowski

On the Compressive Power of Autoencoders With Linear and ReLU Activation Functions

Liangjie Sun
ljsun_seu@126.com
Bioinformatics Center, Institute for Chemical Research, Kyoto University, Kyoto 611-0011, Japan, and Department of Mathematics, University of Hong Kong, Hong Kong

Chenyao Wu
wcy3442@gmail.com
Bioinformatics Center, Institute for Chemical Research, Kyoto University, Kyoto 611-0011, Japan

Wai-Ki Ching
wching@hku.hk
Department of Mathematics, University of Hong Kong, Hong Kong

Tatsuya Akutsu
takutsu@kuicr.kyoto-u.ac.jp
Bioinformatics Center, Institute for Chemical Research, Kyoto University, Kyoto 611-0011, Japan

In this article, we mainly study the depth and width of autoencoders consisting of rectified linear unit (ReLU) activation functions. An autoencoder is a layered neural network consisting of an encoder, which compresses an input vector to a lower-dimensional vector, and a decoder, which transforms the low-dimensional vector back to the original input vector exactly (or approximately). In a previous study, Melkman et al. (2023) studied the depth and width of autoencoders using linear threshold activation functions with binary input and output vectors. We show that similar theoretical results hold if autoencoders using ReLU activation functions with real input and output vectors are used. Furthermore, we show that it is possible to compress input vectors to one-dimensional vectors using ReLU activation functions, although the size of compressed vectors is trivially $\Omega(\log n)$ for autoencoders with linear threshold activation functions, where n is the number of input vectors. We also study the cases of linear activation functions. The results suggest that the compressive power of autoencoders using linear activation functions is considerably limited compared with those using ReLU activation functions.

Neural Computation **37, 235–259** (2025)
https://doi.org/10.1162/neco_a_01729

1 Introduction

Over the past decade, we have seen a rapid progress in both developments and applications of artificial neural network (ANN) technologies. Among various models of ANNs, much attention has recently been paid on autoencoders because of their generative power of new data. Indeed, autoencoders have been applied to various areas including image processing (Doersch, 2016), natural language processing (Tschannen et al., 2018), and drug discovery (Gómez-Bombarelli et al., 2018). An autoencoder is a layered neural network consisting of an encoder and a decoder, where the former transforms an input vector $\mathbf{x}$ to a low-dimensional vector $\mathbf{z} = \mathbf{f}(\mathbf{x})$ and the latter transforms $\mathbf{z}$ to an output vector $\mathbf{y} = \mathbf{g}(\mathbf{z})$, which should be the same as or similar to the input vector. Therefore, an autoencoder performs a kind of dimensionality reduction. Encoder and decoder functions, $\mathbf{f}$ and $\mathbf{g}$, are usually obtained via unsupervised learning that minimizes the difference between input and output data by adjusting weights (and some additional parameters).

Although autoencoders have a long history (Ackley et al., 1985; Baldi & Hornik, 1989; Hinton & Salakhutdinov, 2006), how data are compressed via autoencoders is not yet very clear. Baldi and Hornik (1989) studied relations between principal component analysis (PCA) and autoencoders with one hidden layer. Hinton and Salakhutdinov (2006) conducted empirical studies on relations between the depth of autoencoders and the dimensionality reduction. The results suggest that deeper networks can produce lower reconstruction errors. Kärkkäinen and Hänninen (2023) also empirically studied relations between the depth and the dimensionality reduction using a variant model of autoencoders. The results suggest that deeper networks obtain lower autoencoding errors during the identification of the intrinsic dimension, but the detected dimension does not change compared to a shallow network. Recently, several analyses have been done on mutual information between layers in order to understand information flow in autoencoders (Lee & Jo, 2021; Tapia & Estévez, 2020; Yu & Príncipe, 2019). Baldi (2012) presented and studied a general framework on autoencoders with both linear and nonlinear activation functions. In particular, he showed that learning in the autoencoder with Boolean activation functions is NP-hard in general by a reduction from a clustering problem.

However, as far as we know, no theoretical studies had been done on relations between the compressive power and the size of autoencoders, whereas extensive theoretical studies have been done on the representational power of deep neural networks (Delalleau & Bengio, 2011; Montufar et al., 2014; Vershynin, 2020; Yun et al., 2019). Recently, some theoretical studies have been done on the compressive power of autoencoders with linear threshold functions (Akutsu & Melkman, 2023; Melkman et al., 2023) with respect to the depth (number of layers) and width (number of nodes in a layer). However, linear threshold networks are not popular in recent studies on

neural networks. Furthermore, linear threshold networks can only handle binary input and output, which is far from the practical settings.

Motivated by this situation, we study in this article the compressive power of autoencoders with real input and output vectors using rectified linear unit (ReLU) functions as the activation functions, with focusing on theoretical aspects. In order to clarify the superiority of ReLU functions over linear functions, we also study the compressive power of autoencoders with linear activation functions (not linear threshold activation functions).

The results are summarized in Table 1, where D and d denote the number of dimensions of input and compressed vectors, respectively; n denotes the number of input vectors; and A is a matrix explained in theorem 2. Note that the number and dimensions of input vectors must be the same as those of output vectors. Here, we first obtain some new results about autoencoders with linear activation functions. Then we modify theorems 12, 19, and 22 in Melkman et al. (2023) for ReLU functions and real input vectors. In addition, we modify theorem 1 in Zhang et al. (2017) so that the number of nodes in the middle layer decreases from n to $2\lceil\sqrt{n}\rceil$. Based on the proof of theorem 3.1 in Yun et al. (2019), we design a four-layer ReLU neural network to reduce the number of nodes in the middle layer. The difference between this paper and theorem 3.1 is that hard-tanh activation functions are used in the proof of theorem 3.1, while we use ReLU activation functions. Finally, we modify the decoders in the proofs of theorems 19 and 22 in Melkman et al. (2023).

Specifically, theorem 1 reveals that when a set of $n = d + 1$ vectors in D-dimensional Euclidean space is given, there is a three-layer perfect autoencoder with linear activation functions that has the middle layer with d nodes. When the number of vectors in the given set is greater than $d + 1$, a three-layer perfect autoencoder with linear activation functions that has the middle layer with d nodes may not exist, and in theorem 2, some conditions for the existence of a three-layer perfect autoencoder with linear activation functions that has the middle layer with d nodes are given.

Theorem 3 is obtained by modifying theorem 12 in Melkman et al. (2023). Compared with theorem 12, we apply ReLU activation functions in theorem 3 to replace the binary input vectors with real input vectors, so that the number of layers of the ReLU neural network designed is increased by 2 and the number of hidden nodes is increased by $D + \frac{5d}{2}$.

By adding a decoder to the encoder constructed by theorem 3, we obtain a seven-layer perfect autoencoder in theorem 4. It is worth noting that in the decoder part, we do not simply represent the threshold function of theorem 19 in Melkman et al. (2023) as three ReLU functions with two layers, but design some new ReLU activation functions. Therefore, compared with theorem 19 in Melkman et al. (2023), the numbers of nodes and layers in the decoder part do not increase.

In theorem 4, the size of the middle layer is $2\lceil\sqrt{n}\rceil$. In theorem 5, we reduce the size of the middle layer from $2\lceil\sqrt{n}\rceil$ to $2\lceil\log\sqrt{n}\rceil$. Moreover,

Table 1: Summary of Results.

	Vector	Middle Layer (d)	Architecture	Type	Activation
Theorem 1	real	$n-1$	$D/d/D$	Encoder/Decoder	Linear
Theorem 2	real	d	$D/d/D$ if rank$(A) \leqslant d+1$	Encoder/Decoder	Linear
Theorem 3	real	$2\lceil\sqrt{n}\rceil$	$D/(D+d)/(D+\frac{d}{2})/\frac{3d}{2}/d$	Encoder	ReLU
(Theorem 12 (Melkman et al., 2023))	binary	$2\lceil\sqrt{n}\rceil$	$D/(D+\frac{d}{2})/d$	Encoder	Threshold
Theorem 4	real	$2\lceil\sqrt{n}\rceil$	$D/(D+d)/(D+\frac{d}{2})/\frac{3d}{2}/d/\frac{dD}{2}/D$	Encoder/Decoder	ReLU
(Theorem 19 (Melkman et al., 2023))	binary	$2\lceil\sqrt{n}\rceil$	$D/(D+\frac{d}{2})/d/\frac{dD}{2}/D$	Encoder/Decoder	Threshold
Theorem 5	real	$2\lceil\log\sqrt{n}\rceil$	Theorem 2 $+2d/d/4\lceil\sqrt{n}\rceil/2\lceil\sqrt{n}\rceil/D\lceil\sqrt{n}\rceil/D$	Encoder/Decoder	ReLU
(Theorem 22 (Melkman et al., 2023))	binary	$2\lceil\log\sqrt{n}\rceil$	Theorem 12 (Melkman et al., 2023) $+d/2\lceil\sqrt{n}\rceil/D\lceil\sqrt{n}\rceil/D$	Encoder/Decoder	Threshold
(Corollary 8 (Akutsu & Melkman, 2023))	binary	$\lceil\log n\rceil$	Theorem 12 (Melkman et al., 2023) $+d/2\lceil\sqrt{nD}\rceil/\lceil\sqrt{nD}\rceil/D$	Encoder/Decoder	Threshold
Proposition 2	real	1	$D/1/(2\lceil\sqrt{n}\rceil+1)/(\lceil\sqrt{n}\rceil+1)/$ $3\lceil\sqrt{n}\rceil/2\lceil\sqrt{n}\rceil/D\lceil\sqrt{n}\rceil/D$	Encoder/Decoder	ReLU
Theorem 7	real	$2\lceil\sqrt{n}\rceil$	$D/d/d/1$	Memorizer	ReLU
(Theorem 1 (Zhang et al., 2017))	real	n	$D/d/1$	Memorizer	ReLU

Note: Theorems in parentheses are existing results.

theorem 5 is obtained by modifying theorem 22 in Melkman et al. (2023). In proposition 2, we further reduce the size of the middle layer to 1 by using a real number as a compressed vector. The proof is obtained by simple modifications of those for theorems 3 and 4.

We also consider the number of nodes and layers to represent a memorizer, which is a set of pairs of input vectors and their output values. Compared with theorem 1 in Zhang et al. (2017), in theorem 7, the number of nodes in the middle layer decreases to $2\lceil\sqrt{n}\rceil$ and the number of layers increases by 1. It is worth noting that the four-layer ReLU neural network we designed in theorem 7 is based on the four-layer fully connected neural network of theorem 3.1 in Yun et al. (2019).[1] However, in theorem 3.1, it mainly describes the design process of a fully connected neural network using hard-tanh activation functions, while we need to design a ReLU neural network, so there are some differences in the design process, and we explain these differences.

It is seen from Table 1 that there is a large difference in the number of nodes of the middle layer between the linear and ReLU autoencoders. Linear autoencoders need $n - 1$ nodes in the middle layer (especially when $\text{rank}(A) = n$), whereas ReLU autoencoders need a much smaller number of nodes in the middle layer. This is a crucial limitation of linear autoencoders. However, ReLU autoencoders need more than three layers and a large number of nodes (e.g., $D\lceil\sqrt{n}\rceil$ nodes) in some layers. This is a limitation of ReLU autoencoders. In addition, errors are not taken into account in both types of autoencoders, a limitation from a practical viewpoint.

In summary, the contribution of this article is to theoretically analyze the number of nodes and layers of autoencoders for real input and output vectors for the first time. Although we use the framework and some techniques introduced in Melkman et al. (2023), we introduce additional techniques in this article, and thus there exist substantial differences between the results of Melkman et al. (2023) and those of this article, as seen from Table 1. We also use some techniques introduced in Yun et al. (2019). However, we use to show theorem 7, which can also be used as a decoder part of the autoencoder.

2 Problem Definitions

R stands for real numbers. R^n denotes the set of n-dimensional column vectors. For integers a and b, $a < b$, we denote $[a : b] := \{a, a + 1, \ldots, b\}$.

A function f: $\{0, 1\}^h \to \{0, 1\}$ is called a *Boolean threshold function* if it is represented as

[1] In Yun et al. (2019), their network is stated as a three-layer network excluding the input layer.

$$f(\mathbf{x}) = \begin{cases} 1, & \mathbf{a} \cdot \mathbf{x} \geq \theta, \\ 0, & \text{otherwise}, \end{cases}$$

for some $(\mathbf{a}, \theta)$, where $\mathbf{a} \in R^h$, $\theta \in R$, and $\mathbf{a} \cdot \mathbf{x}$ denotes the inner product between two vectors $\mathbf{a}$ and $\mathbf{x}$. We also denote the same function f as $[\mathbf{a} \cdot \mathbf{x} \geqslant \theta]$.

A function f: $R^h \to R$ is called a *ReLU* function if it is represented as

$$f(\mathbf{x}) = \max(\mathbf{a} \cdot \mathbf{x} + b, 0).$$

In this article, we only consider layered neural networks in which a linear, linear threshold, or ReLU function is assigned to each node except input nodes. The nodes in a network are divided into L-layers, and each node in the ith layer has inputs only from nodes in the $(i-1)$th layer ($i = 2, \ldots, L-1$). Then the states of nodes in the ith layer can be represented as a W_i-dimensional binary vector where W_i is the number of nodes in the ith layer and is called the *width* of the layer. A layered neural network is represented as $\mathbf{y} = \mathbf{f}^{(L-1)}(\mathbf{f}^{(L-2)}(\cdots \mathbf{f}^{(1)}(\mathbf{x}) \cdots))$, where $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors, respectively, and $\mathbf{f}^{(i)}$ is a list of activation functions for the $(i+1)$th layer. The 1st and (L)th layers are called the *input* and *output layers*, respectively, and the corresponding nodes are called *input* and *output nodes*, respectively. When we consider autoencoders, one layer (kth layer where $k \in \{2, \ldots, L-1\}$) is specified as the *middle layer*, and the nodes in this layer are called the *middle nodes*. Then the *middle vector* $\mathbf{z}$, *encoder* $\mathbf{f}$, and *decoder* $\mathbf{g}$ are defined by

$$\mathbf{z} = \mathbf{f}^{(k-1)}(\mathbf{f}^{(k-2)}(\cdots \mathbf{f}^{(1)}(\mathbf{x}) \cdots)) = \mathbf{f}(\mathbf{x}),$$
$$\mathbf{y} = \mathbf{f}^{(L-1)}(\mathbf{f}^{(L-2)}(\cdots \mathbf{f}^{(k)}(\mathbf{z}) \cdots)) = \mathbf{g}(\mathbf{z}).$$

Since we consider autoencoders, the input layer and the output layer have the same number of nodes, denoted by D. We use d to denote the number of nodes in the middle layer.

Let $X_n = \{\mathbf{x}^0, \ldots, \mathbf{x}^{n-1}\}$ be a set of n D-dimensional binary or real input vectors that are all different. We define perfect encoder, decoder, autoencoder as follows (Melkman et al., 2023).

Definition 1. *A mapping* $\mathbf{f}$ *is called a* perfect encoder *for* X_n *if* $\mathbf{f}(\mathbf{x}^i) \neq \mathbf{f}(\mathbf{x}^j)$ *holds for all* $i \neq j$.

Definition 2. *A pair of mappings* $(\mathbf{f}, \mathbf{g})$ *with* $\mathbf{f}$ *and* $\mathbf{g}$ *is called a perfect autoencoder if* $\mathbf{g}(\mathbf{f}(\mathbf{x}^i)) = \mathbf{x}^i$ *holds for all* $\mathbf{x}^i \in X_n$. *Furthermore,* $\mathbf{g}$ *is called a* perfect decoder.

In a word, a perfect encoder maps distinct input vectors into distinct middle vectors, a perfect decoder maps each middle vector into the original input vector, and a perfect autodecoder maps each input vector into the original input vector via a distinct middle vector.

Note that a perfect decoder exists only if there exists a perfect autoencoder. Furthermore, it is easily seen from the definitions that if (**f**, **g**) is a perfect autoencoder, **f** is a perfect encoder.

Many of the results in Table 1 rely on the following proposition. Since it is mentioned in Zhang et al. (2017) without a proof, we give our own proof here:

Proposition 1. *For any set of D-dimensional distinct real vectors* $X = \{\mathbf{x}^0, \ldots, \mathbf{x}^{n-1}\}$, *there exists a real vector* $\mathbf{a}$ *satisfying* $\mathbf{a} \cdot \mathbf{x}^i \neq \mathbf{a} \cdot \mathbf{x}^j$ *for all* $i \neq j$.

Proof. We prove the proposition by mathematical induction on D.

In the case of $D = 1$, the claim trivially holds by letting $\mathbf{a} = [1]$. Assume that the claim holds for all X in the case of $D = d - 1$. Let $X = \{\mathbf{x}^0, \ldots, \mathbf{x}^{n-1}\}$ be a set of d-dimensional distinct vectors. For each vector $\mathbf{x} = [x_0, \ldots, x_{d-1}]$, let $\hat{\mathbf{x}} = [x_0, \ldots, x_{d-2}]$ and $\hat{X} = \{\hat{\mathbf{x}}^0, \ldots, \hat{\mathbf{x}}^{n-1}\}$. It should be noted that $\hat{\mathbf{x}}^i = \hat{\mathbf{x}}^j$ may hold for some $i \neq j$. From the induction hypothesis, we can assume that there exists a vector $\hat{\mathbf{a}} = [a_0, \ldots, a_{d-2}]$ satisfying $\hat{\mathbf{a}} \cdot \hat{\mathbf{x}}^i \neq \mathbf{a} \cdot \hat{\mathbf{x}}^j$ for all $\hat{\mathbf{x}}^i \neq \hat{\mathbf{x}}^j$. Here, we define $\mathbf{a} = [a_0, \ldots, a_{d-2}, a_{d-1}]$ by using a sufficiently large real number a_{d-1} such that $|a_{d-1} \cdot (x^i_{d-1} - x^j_{d-1})| \gg |\hat{\mathbf{a}} \cdot (\hat{\mathbf{x}}^h - \hat{\mathbf{x}}^k)|$ holds for any i, j, k, h such that $x^i_{d-1} \neq x^j_{d-1}$. Then $\mathbf{a} \cdot \mathbf{x}^i \neq \mathbf{a} \cdot \mathbf{x}^j$ clearly holds for all $i \neq j$. □

Furthermore, we can assume without loss of generality (w.l.o.g.) that $\mathbf{x}^i$s are reindexed so that

$$0 < c_0 < \cdots < c_{n-1}, \; c_i = \mathbf{a} \cdot \mathbf{x}^i, \tag{2.1}$$

holds and additionally, $c_{-1} = c_0 - \delta > 0$ and $c_n = c_{n-1} + \delta$, hold for any $\delta > 0$.

For the input vector, we can assume w.l.o.g. that all elements of the real input vector $\mathbf{x}^i$ are nonnegative, because this assumption can be satisfied by adding a sufficiently large constant to each element. For example, let $z = \min_{i \in [0:n-1],\, j \in [0:D-1]}(x^i_j)$. If $z < 0$, all elements of the real input vector $\mathbf{x}^i$ can be made nonnegative by adding any value not less than $-z$ to each element, and this value does not affect network performance.

3 Autoencoders with Linear Activation Functions

In this section, we consider autoencoders using linear functions as the activation functions.

We begin with a simple example. Consider the case of $D = 3$, $d = 1$, and $n = 2$. Then $X = \{[x_0, y_0, z_0], [x_1, y_1, z_1]\}$. Let $\mathbf{f}([x_i, y_i, z_i]) = [x_i]$. Let $\mathbf{g}([x_i]) = [x_i, ax_i + b, cx_i + d]$, where $ax_i + b$ satisfies $ax_0 + b = y_0$ and $ax_1 + b = y_1$, and $cx_i + d$ satisfies $cx_0 + d = z_0$ and $cx_1 + d = z_1$. Then (**f**, **g**) is a perfect autoencoder for most X (precisely, if $\det\left(\begin{bmatrix} x_0 & 1 \\ x_1 & 1 \end{bmatrix}\right) \neq 0$), where $\det(A)$ denotes the determinant of a matrix A.)

For another example, consider the case of $D = 4$, $d = 2$, and $n = 3$. Then

$$X = \{[x_0, y_0, z_0, w_0], [x_1, y_1, z_1, w_1], [x_2, y_2, z_2, w_2]\}.$$

Let $\mathbf{f}([x_i, y_i, z_i, w_i]) = [x_i, y_i]$. Let $\mathbf{g}([x_i, y_i]) = [x_i, y_i, ax_i + by_i + c, dx_i + ey_i + f]$, where $ax_i + by_i + c$ satisfies $ax_0 + by_0 + c = z_0$, $ax_1 + by_1 + c = z_1$, and $ax_2 + by_2 + c = z_2$, and $dx_i + ey_i + f$ satisfies similar equations. Then, $(\mathbf{f}, \mathbf{g})$ is a perfect autoencoder for most X (precisely, if $\det\left(\begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix}\right) \neq 0$).

By generalizing these simple examples, we have the following theorem.

Theorem 1. *Let X be a set of $d + 1$ vectors in D-dimensional Euclidean space, where D is any integer such that $D > d$. Then, for X, there exists a perfect autoencoder $(\mathbf{f}, \mathbf{g})$ with linear activation functions that has the middle layer (i.e., compressed layer) with d nodes (i.e., perform dimensionality reduction to d).*

Proof. Suppose that a set of $d + 1$ vectors in D-dimensional Euclidean space is described as $X = \{\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^d\}$, where $\mathbf{x}^i = [x_0^i, x_1^i, \ldots, x_{D-1}^i]$, $i \in [0 : d]$ and $D > d$.

Then a matrix A can be constructed as follows:

$$A = \begin{bmatrix} x_0^0 & x_1^0 & \cdots & x_{D-1}^0 & 1 \\ x_0^1 & x_1^1 & \cdots & x_{D-1}^1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_0^d & x_1^d & \cdots & x_{D-1}^d & 1 \end{bmatrix}.$$

Here, A is a $(d + 1) \times (D + 1)$ matrix and $D > d$, so $\operatorname{rank}(A) \leqslant d + 1$ and assuming w.l.o.g. that $\operatorname{rank}(A) = d_1 + 1$ ($d_1 \leqslant d$).

Let

$$\mathbf{x}_0 = \begin{bmatrix} x_0^0 \\ x_0^1 \\ \vdots \\ x_0^d \end{bmatrix}, \ldots, \mathbf{x}_{D-1} = \begin{bmatrix} x_{D-1}^0 \\ x_{D-1}^1 \\ \vdots \\ x_{D-1}^d \end{bmatrix}, \mathbf{1}_{d+1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

and $S = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{D-1}, \mathbf{1}_{d+1}\}$. According to the fact that the rank of the matrix A is the maximum number of linearly independent column vectors in A, we assume w.l.o.g. that a maximal linearly independent subset of S is $\{\mathbf{x}_{\alpha_1}, \mathbf{x}_{\alpha_2}, \ldots, \mathbf{x}_{\alpha_{d_1}}, \mathbf{1}_{d+1}\}$, where $0 \leq \alpha_1 < \cdots < \alpha_{d_1} \leq D - 1$.

A maximal linearly independent subset of S can be obtained as follows:

Step 1: Initialize $\bar{S} = \{\mathbf{1}_{d+1}\}$.
Step 2: Choose a column vector $\mathbf{x}_i \in S$. If $\mathbf{1}_{d+1}$ and $\mathbf{x}_i$ are linearly dependent, remove $\mathbf{x}_i$ from S and choose another column vector in S and repeat this step. If $\mathbf{1}_{d+1}$ and $\mathbf{x}_i$ are linearly independent, add $\mathbf{x}_i$ to the set $\bar{S}$, that is, $\bar{S} = \{\mathbf{1}_{d+1}, \mathbf{x}_i\}$, and go to the next step.
Step 3: Let $\bar{S} = \{\mathbf{1}_{d+1}, \mathbf{x}_i\}$, where $\mathbf{1}_{d+1}$ and $\mathbf{x}_i$ are linearly independent. Choose a column vector $\mathbf{x}_j \in S$. If $\mathbf{1}_{d+1}$, $\mathbf{x}_i$ and $\mathbf{x}_j$ are linearly dependent, remove $\mathbf{x}_j$ from S and choose another column vector in S and repeat this step. If $\mathbf{1}_{d+1}$, $\mathbf{x}_i$ and $\mathbf{x}_j$ are linearly independent, add $\mathbf{x}_j$ to the set $\bar{S}$, that is, $\bar{S} = \{\mathbf{1}_{d+1}, \mathbf{x}_i, \mathbf{x}_j\}$ and go to the next step.
Step 4: . . .

Going on in turn, we finally find that $\bar{S}$ is a maximal linearly independent subset of S. Here, $\mathbf{1}_{d+1} \in \bar{S}$.

Then any other vector in S can be expressed as a linear combination of elements of the maximal linearly independent subset, so for any $\mathbf{x}_j \in S$, we can always have

$$\mathbf{x}_j = a_1^j \mathbf{x}_{\alpha_1} + a_2^j \mathbf{x}_{\alpha_2} + \cdots + a_{d_1}^j \mathbf{x}_{\alpha_{d_1}} + a^j \mathbf{1}_{d+1},$$

where $a_1^j, a_2^j, \ldots, a_{d_1}^j, a^j \in R$.

And especially if $d_1 < d$, we add other $d - d_1$ vectors $\mathbf{x}_{\alpha_{d_1+1}}, \ldots, \mathbf{x}_{\alpha_d}$ from $S \setminus \{\mathbf{x}_{\alpha_1}, \ldots, \mathbf{x}_{\alpha_{d_1}}, \mathbf{1}_{d+1}\}$ to the maximal linearly independent subset. Then for any $\mathbf{x}_j \in S$, we still have

$$\mathbf{x}_j = a_1^j \mathbf{x}_{\alpha_1} + a_2^j \mathbf{x}_{\alpha_2} + \cdots + a_d^j \mathbf{x}_{\alpha_d} + a^j \mathbf{1}_{d+1}.$$

Let

$$\mathbf{f}(\mathbf{x}^i) = \mathbf{f}([x_0^i, x_1^i, \ldots, x_{D-1}^i]) = [x_{\alpha_1}^i, x_{\alpha_2}^i, \ldots, x_{\alpha_d}^i]$$

and

$$\mathbf{g}([x_{\alpha_1}^i, x_{\alpha_2}^i, \ldots, x_{\alpha_d}^i]) = [y_0^i, y_1^i, \ldots, y_{D-1}^i],$$

where $y_j^i = a_1^j x_{\alpha_1}^i + a_2^j x_{\alpha_2}^i + \cdots + a_d^j x_{\alpha_d}^i + a^j$, $j \in [0 : D-1]$, and $i \in [0 : d]$. Clearly, we have $a_1^j x_{\alpha_1}^i + a_2^j x_{\alpha_2}^i + \cdots + a_d^j x_{\alpha_d}^i + a^j = x_j^i$, $j \in [0 : D-1]$, and $i \in [0 : d]$, that is, $\mathbf{g}([x_{\alpha_1}^i, x_{\alpha_2}^i, \ldots, x_{\alpha_d}^i]) = [x_0^i, x_1^i, \ldots, x_{D-1}^i]$. Hence, $(\mathbf{f}, \mathbf{g})$ is a perfect autoencoder for X.

For X, there exists a perfect autoencoder $(\mathbf{f}, \mathbf{g})$ with linear activation functions that has the middle layer (i.e., compressed layer) with d nodes (i.e., perform dimensionality reduction to d). □

Furthermore, for X with $|X| > d+1$, we consider whether there is a perfect autoencoder with linear activation functions that has the compressed layer with d nodes and we have the following result:

Theorem 2. *Consider a set of $c+1$ vectors $X = \{\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^c\}$ in D-dimensional Euclidean space, where $\mathbf{x}^i = [x_0^i, x_1^i, \ldots, x_{D-1}^i]$, $i \in [0 : c]$. Let*

$$A = \begin{bmatrix} x_0^0 & x_1^0 & \cdots & x_{D-1}^0 & 1 \\ x_0^1 & x_1^1 & \cdots & x_{D-1}^1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_0^c & x_1^c & \cdots & x_{D-1}^c & 1 \end{bmatrix}.$$

There are two cases:

- *If* $\text{rank}(A) \leq d+1$, *then there exists a perfect autoencoder with linear activation functions that has the compressed layer with d, $d < c$ nodes.*
- *If* $\text{rank}(A) > d+1$, *then there does not exist a perfect autoencoder with linear activation functions that has the compressed layer with d, $d < c$ nodes.*

Proof. When $\text{rank}(A) \leqslant d+1$, a perfect autoencoder can be constructed similar to theorem 1.

When $\text{rank}(A) > d+1$, let

$$\mathbf{x}_0 = \begin{bmatrix} x_0^0 \\ x_0^1 \\ \vdots \\ x_0^c \end{bmatrix}, \ldots, \mathbf{x}_{D-1} = \begin{bmatrix} x_{D-1}^0 \\ x_{D-1}^1 \\ \vdots \\ x_{D-1}^c \end{bmatrix}, \mathbf{1}_{c+1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

and $S = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{D-1}, \mathbf{1}_{c+1}\}$. If we want to find a basis for S, then the number of vectors in the basis must be larger than $d+1$. Hence, there does not exist a perfect autoencoder with linear activation functions that has the compressed layer with d nodes. □

Remark 1. According to theorems 1 and 2, we know that the number of nodes in the compression layer is at least $\text{rank}(A) - 1$ for the existence of a perfect autoencoder.

Remark 2. Theorem 1 still holds for multilayer networks because composition of linear functions is a linear function.

Remark 3. It is worth noting that for the above example, if $x_0 = x_1$, the method for the case of $D = 3$, $d = 1$, and $n = 2$ does not work. The reason is that in this case,

$$\left\{\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\}$$

is not the maximal linearly independent subset of

$$\left\{\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}, \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}, \begin{bmatrix} z_0 \\ z_1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\}.$$

More generally, if $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{d-1}, \mathbf{1}_{d+1}\}$ does not contain the maximal linearly independent subset of S, then in order to get a perfect autoencoder, we cannot design the encoder f as $f(\mathbf{x}^i) = [\mathbf{x}_0^i, \mathbf{x}_1^i, \ldots, \mathbf{x}_{d-1}^i]$.

Remark 4. According to theorem 2, we need to calculate the rank of the matrix A. For the large matrix A, it is not very easy to directly obtain its rank, but we can use rank estimation methods to estimate its rank, such as in Ubaru and Saad (2016).

4 Autoencoders with ReLU functions for Real Vectors

According to Kumano and Akutsu (2022), we can simulate a threshold function by using three ReLU functions in two layers. Hence, theorems 12, 19, and 22 in Melkman et al. (2023) can be modified for ReLU functions such that binary input vectors are replaced by real input vectors, with increasing the number of layers by some constant and increasing the number of nodes in some layers by twice (because one threshold function can be simulated using three ReLU functions with two layers). Note that the compressed layer still corresponds to binary vectors.

Specifically, given n different binary input vectors of dimension D, in theorem 12 (Melkman et al., 2023), it is mentioned that there is a three-layer network whose activation functions are Boolean threshold functions that maps these vectors to n different binary vectors of dimension $2\lceil\sqrt{n}\rceil$ using $\lceil\sqrt{n}\rceil + D$ hidden nodes, and then in theorem 19 (Melkman et al., 2023), it is mentioned that there is a five-layer perfect Boolean threshold network autoencoder, whose encoding is constructed based on theorem 12, and the number of nodes in its middle hidden layer is $2\lceil\sqrt{n}\rceil$. Further, in theorem 22 (Melkman et al., 2023), it is mentioned that there is a seven-layer perfect Boolean threshold network autoencoder, and compared with the autoencoder constructed in theorem 19, the number of nodes in the middle hidden layer is reduced to $2\lceil\log\sqrt{n}\rceil$, although the number of layers is increased by 2.

As in Melkman et al. (2023), we define an r-dimensional binary vector $\mathbf{h}^i[r] = [h_0^i, \ldots, h_{r-1}^i]$ by

$$h_j^i = \begin{cases} 1 & \text{if } j \le i, \\ 0 & \text{otherwise.} \end{cases}$$

We first modify theorem 12 in Melkman et al. (2023) for ReLU functions and real input vectors as follows:

Theorem 3. *Let $r = \lceil \sqrt{n} \rceil$. For any set of D-dimensional real vectors,*

$$X = \{\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{n-1}\},$$

where $\mathbf{x}^i = [x_0^i, x_1^i, \ldots, x_{D-1}^i]$ and $i \in [0 : n-1]$, there exists a five-layer ReLU neural network that maps $\mathbf{x}^i$ to $(\mathbf{h}^k[r], \mathbf{h}^l[r])$, where $i = kr + l$ with $i \in [0 : n-1]$, $k \in [0 : r-1]$, and $l \in [0 : r-1]$.

The neural network has D input nodes x_j, $j \in [0 : D-1]$; $D + 2r$ nodes in the second layer $\alpha_i^1, \alpha_{r+i}^1$, $i \in [0 : r-1]$ and α_j^2, $j \in [0 : D-1]$, $D + r$ nodes in the third layer β_i^1, $i \in [0 : r-1]$ and β_j^2, $j \in [0 : D-1]$, $3r$ nodes in the fourth layer γ_i^1, $i \in [0 : r-1]$ and $\gamma_i^2, \gamma_{r+i}^2$, $i \in [0 : r-1]$, and $2r$ output nodes η_i^1, η_i^2, $i \in [0 : r-1]$ (see also Figure 1).

Proof. First, the nodes α_j^2 simply copy the input, $\alpha_j^2 = x_j$, $j \in [0 : D-1]$. For any $i \in [0 : r-1]$, we choose s_i with $c_{ir-1} < s_i < c_{ir}$, and there exists an $\varepsilon_i^1 > 0$, such that $s_i - \varepsilon_i^1 > c_{ir-1}$ and $s_i + \varepsilon_i^1 < c_{ir}$. Let the ReLU activation function of α_i^1 be

$$\max\left(\frac{1}{2\varepsilon_i^1}(\mathbf{a} \cdot \mathbf{x} - (s_i - \varepsilon_i^1)), 0\right)$$

and the ReLU activation function of α_{r+i}^1 be

$$\max\left(\frac{1}{2\varepsilon_i^1}(\mathbf{a} \cdot \mathbf{x} - (s_i + \varepsilon_i^1)), 0\right).$$

In the third layer, let $\beta_j^2 = \alpha_j^2$, $j \in [0 : D-1]$ and

$$\beta_i^1 = \max\left(\alpha_i^1 - \alpha_{r+i}^1, 0\right), i \in [0 : r-1].$$

Here, it is easy to see that $\beta^1 = \mathbf{h}^k[r]$.

In the fourth layer, the node γ_i^1 copies β_i^1, where $i \in [0 : r-1]$. For any $i \in [0 : r-1]$, there exists an $\varepsilon_i^2 > 0$ such that $c_{kr+i} - 3\varepsilon_i^2 > c_{kr+(i-1)}$. Let the ReLU activation function of γ_i^2 be

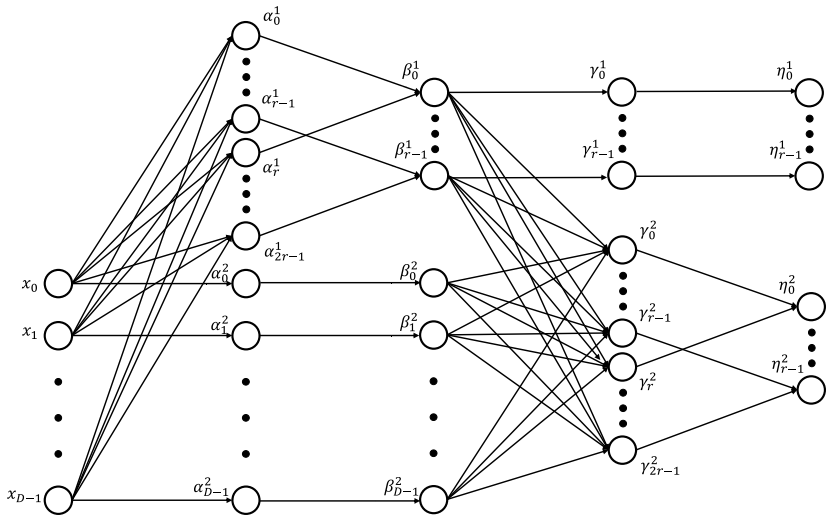

Figure 1: Five-layer ReLU neural network. The circle nodes represent those carrying input and output information. Specifically, there are D input nodes x_j, $j \in [0 : D-1]$ and $2r$ output nodes η_i^1, η_i^2, $i \in [0 : r-1]$. The arrows connecting the circle nodes show that the states of nodes in the ith ($i = 2, 3, 4, 5$) layer are determined based on the ReLU activation functions related to the states of some nodes in the $(i-1)$th layer.

$$\max\left(\frac{1}{2\varepsilon_i^2}(\mathbf{a}\cdot\beta^2 - t_i - (0-\varepsilon_i^2)), 0\right)$$

and the ReLU activation function of γ_{r+i}^2 be

$$\max\left(\frac{1}{2\varepsilon_i^2}(\mathbf{a}\cdot\beta^2 - t_i - (0+\varepsilon_i^2)), 0\right),$$

where $t_i = c_i\beta_0^1 + (c_{r+i} - c_i)\beta_1^1 + \cdots + (c_{(r-1)r+i} - c_{(r-2)r+i})\beta_{r-1}^1 - 2\varepsilon_i^2$ and $i \in [0 : r-1]$.

Finally, the output node η_i^1 copies γ_i^1, where $i \in [0 : r-1]$. The ReLU activation functions of the remaining output nodes η_i^2, $i \in [0 : r-1]$ are $\max\left(\gamma_i^2 - \gamma_{r+i}^2, 0\right)$. Here, we can observe that $\eta^2 = \mathbf{h}^l[r]$. □

Example 1. Suppose that $n = 16$. Then, we have $r = \sqrt{16} = 4$, $\mathbf{h}^0[r] = [1, 0, 0, 0]$, $\mathbf{h}^1[r] = [1, 1, 0, 0]$, $\mathbf{h}^2[r] = [1, 1, 1, 0]$, and $\mathbf{h}^3[r] = [1, 1, 1, 1]$. Furthermore, $\mathbf{x}^9$ is mapped to $(\eta^1, \eta^2) = (\mathbf{h}^2[r], \mathbf{h}^1[r])$ because $9 = 2 \cdot 4 + 1$.

By modifying theorem 19 in Melkman et al. (2023) for ReLU functions and real input vectors, we get the following theorem.

Theorem 4. *Let $r = \lceil \sqrt{n} \rceil$. For any set of D-dimensional real vectors,*

$$X = \{\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{n-1}\},$$

where $\mathbf{x}^i = [x_0^i, x_1^i, \ldots, x_{D-1}^i]$ and $i \in [0 : n-1]$, there exists a seven-layer perfect ReLU neural network autoencoder. There are $2r$ nodes in the middle layer $\beta_i^1, \beta_i^2, i \in [0 : r-1]$; rD nodes in the sixth layer $\eta_{i,j}$, $i \in [0 : r-1]$, $j \in [0 : D-1]$; and D output nodes y_j, $j \in [0 : D-1]$.

Proof. On top of the encoder constructed in the proof of theorem 3, we add a decoder that is a two-layer ReLU neural network.

Here, we need to design a decoder that outputs $\mathbf{y} = \mathbf{x}^{kr+l}$ for an input $(\mathbf{h}^k[r], \mathbf{h}^l[r])$. Let $(\beta^1, \beta^2) = (\mathbf{h}^k[r], \mathbf{h}^l[r])$. Note that (β^1, β^2) corresponds to (η^1, η^2) in theorem 3.

First, we choose an ϵ, where

$$\epsilon > \max\left(\{x_j^i | i \in [0 : n-1],\ j \in [0 : D-1]\}\right).$$

Equip the node $\eta_{i,j}$, $i \in [0 : r-1]$, $j \in [0 : D-1]$ with the ReLU activation function

$$\max(-x_j^i\beta_0^1 + (-x_j^{r+i} + x_j^i)\beta_1^1 + \cdots + (-x_j^{r(r-1)+i} + x_j^{r(r-2)+i})\beta_{r-1}^1 + \epsilon(\beta_i^2 - \beta_{i+1}^2), 0),$$

where $\beta_r^2 = 0$.

Note that when $\beta^1 = \mathbf{h}^k[r]$, we have

$$-x_j^i\beta_0^1 + (-x_j^{r+i} + x_j^i)\beta_1^1 + \cdots + (-x_j^{r(r-1)+i} + x_j^{r(r-2)+i})\beta_{r-1}^1 = -x_j^{kr+i} \le 0.$$

Furthermore, when $\beta^2 = \mathbf{h}^l[r]$, we have

$$\epsilon(\beta_i^2 - \beta_{i+1}^2) = \begin{cases} \epsilon & \text{if } i = l, \\ 0 & \text{otherwise.} \end{cases}$$

Hence, the value of $\eta_{i,j}$ is $-x_j^{kr+l} + \epsilon$ if $i = l$ and 0 otherwise.

Finally, the ReLU activation function of the output node y_j is

$$\max\left(\sum_{i=0}^{r-1}(-1 \cdot \eta_{i,j}) + \epsilon, 0\right),$$

where $j \in [0 : D-1]$. □

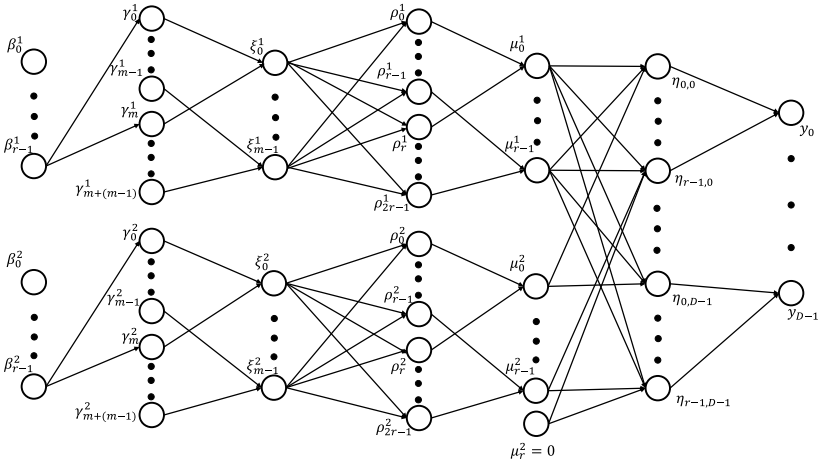

Figure 2: Eleven-layer perfect ReLU neural network autoencoder (omit the first 4 layers). The circle nodes represent those carrying input and output information. Specifically, there are $2m$ nodes ξ_j^1, ξ_j^2, $j \in [0 : m-1]$ in the middle layer and D nodes y_j, $j \in [0 : D-1]$ in the output layer. The arrows connecting the circle nodes show that the states of nodes in the ith ($i = 6, 7, \ldots, 11$) layer are determined based on the ReLU activation functions related to the states of some nodes in the $(i-1)$th layer.

In theorem 4, the size of the middle layer is $2\lceil\sqrt{n}\rceil$. Next, we reduce the size of the middle layer from $2\lceil\sqrt{n}\rceil$ to $2\lceil\log\sqrt{n}\rceil$ by increasing the number of layers.

Theorem 5. *Let $r = \lceil\sqrt{n}\rceil$. For any set of D-dimensional real vectors*

$$X = \{\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{n-1}\},$$

where $\mathbf{x}^i = [x_0^i, x_1^i, \ldots, x_{D-1}^i]$ and $i \in [0 : n-1]$, there exists an 11-layer perfect ReLU neural network autoencoder. There are $2\lceil\log\sqrt{n}\rceil$ nodes in the middle layer (see also Figure 2).

Proof. The first five layers are constructed in the proof of theorem 3, and then the input $\mathbf{x}^i$ maps to $(\mathbf{h}^k[r], \mathbf{h}^l[r])$. Let $(\beta^1, \beta^2) = (\mathbf{h}^k[r], \mathbf{h}^l[r])$. For ease of exposition, we assume that $r = 2^m$.

On (β^1, β^2), the ReLU activation functions for γ_j^i, $i = 1, 2$, $j \in [0 : m-1]$ can be computed as follows:

$$\gamma_{m-1}^i : \max\left(5 \cdot \left(\beta_{\frac{r}{2}}^i - 0.5\right), 0\right),$$

$$\gamma_{m-2}^i : \max\left(5 \cdot \left(\beta_{\frac{r}{4}}^i - \beta_{\frac{2r}{4}}^i + \beta_{\frac{3r}{4}}^i - 0.5\right), 0\right),$$

$$\gamma^i_{m-3} : \max\left(5 \cdot \left(\beta^i_{\frac{r}{8}} - \beta^i_{\frac{2r}{8}} + \beta^i_{\frac{3r}{8}} - \beta^i_{\frac{4r}{8}} + \cdots + \beta^i_{\frac{7r}{8}} - 0.5\right), 0\right),$$

$$\vdots$$

$$\gamma^i_0 : \max\left(5 \cdot \left(\beta^i_1 - \beta^i_2 + \beta^i_3 - \beta^i_4 + \cdots + \beta^i_{r-1} - 0.5\right), 0\right),$$

and the ReLU activation functions for γ^i_{m+j}, $i = 1, 2$, $j \in [0 : m-1]$ can be computed as follows:

$$\gamma^i_{m+(m-1)} : \max\left(5 \cdot \left(\beta^i_{\frac{r}{2}} - 0.7\right), 0\right),$$

$$\gamma^i_{m+(m-2)} : \max\left(5 \cdot \left(\beta^i_{\frac{r}{4}} - \beta^i_{\frac{2r}{4}} + \beta^i_{\frac{3r}{4}} - 0.7\right), 0\right),$$

$$\gamma^i_{m+(m-3)} : \max\left(5 \cdot \left(\beta^i_{\frac{r}{8}} - \beta^i_{\frac{2r}{8}} + \beta^i_{\frac{3r}{8}} - \beta^i_{\frac{4r}{8}} + \cdots + \beta^i_{\frac{7r}{8}} - 0.7\right), 0\right),$$

$$\vdots$$

$$\gamma^i_{m+0} : \max\left(5 \cdot \left(\beta^i_1 - \beta^i_2 + \beta^i_3 - \beta^i_4 + \cdots + \beta^i_{r-1} - 0.7\right), 0\right).$$

Let $\xi^i_j = \max\left(\gamma^i_j - \gamma^i_{m+j}, 0\right)$, where $i = 1, 2$ and $j \in [0 : m-1]$. Then ξ^i gives a binary representation (in the reverse order) of β^i. For example, $\xi^i = [1, 1, 0]$ for $\beta^i = [1, 1, 1, 1, 0, 0, 0, 0]$, where $n = 64$ and $r = 8$.

In the next layer, let the ReLU activation functions for ρ^i_j, $i = 1, 2$, $j \in [0 : r-1]$ be

$$\max\left(\frac{1}{\varepsilon}\left(\sum_{h=0}^{m-1} \xi^i_h 2^h - (j - 2\varepsilon)\right), 0\right)$$

and the ReLU activation functions for ρ^i_{r+j}, $i = 1, 2$, $j \in [0 : r-1]$ be

$$\max\left(\frac{1}{\varepsilon}\left(\sum_{h=0}^{m-1} \xi^i_h 2^h - (j - \varepsilon)\right), 0\right),$$

where $0 < \varepsilon < 0.5$.

Let $\mu^i_j = \max\left(\rho^i_j - \rho^i_{r+j}, 0\right)$, where $i = 1, 2$ and $j \in [0 : r-1]$. Here, we can find that $\mu^1 = \beta^1$ and $\mu^2 = \beta^2$.

Finally, the last two layers are constructed as shown in the proof of theorem 4. □

In the above, the compressed layer corresponds to binary vectors. However, we can simply use $\mathbf{a} \cdot \mathbf{x}^i$ to compress a D-dimensional vector to a 1-dimensional vector (i.e., a scalar). As shown below, decoders can be

developed for this compressed scalar value by modifying the decoders in the proofs of theorems 3 and 4.

Proposition 2. *Let $r = \lceil \sqrt{n} \rceil$. For any set of D-dimensional real vectors $X = \{\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{n-1}\}$, where $\mathbf{x}^i = [x_0^i, x_1^i, \ldots, x_{D-1}^i]$ and $i \in [0 : n-1]$, there exists an eight-layer perfect ReLU neural network autoencoder.*

The neural network has D input nodes x_j, $j \in [0 : D-1]$ one node in the second layer $\mathbf{a} \cdot \mathbf{x}$, $2r+1$ nodes in the third layer $\alpha_i^1, \alpha_{r+i}^1$, $i \in [0 : r-1]$ and α^2, $r+1$ nodes in the fourth layer β_i^1, $i \in [0 : r-1]$ and β^2, $3r$ nodes in the fifth layer γ_i^1, $i \in [0 : r-1]$ and $\gamma_i^2, \gamma_{r+i}^2$, $i \in [0 : r-1]$, $2r$ nodes in the sixth layer η_i^1, η_i^2, $i \in [0 : r-1]$, Dr nodes in the seventh layer $\xi_{i,j}$, $i \in [0 : r-1]$, $j \in [0 : D-1]$, and D output nodes y_j, $j \in [0 : D-1]$.

Proof. We have one node $\mathbf{a} \cdot \mathbf{x}$ in the second layer, which corresponds to an encoder.

The decoder is obtained by simple modifications of the proofs of theorems 3 and 4. We use the output of the second layer in place of $\mathbf{a} \cdot \mathbf{x}$ in the activation function for each of α_i^1 in the proof of theorem 3. Furthermore, we replace β^2 by a single node copying $\mathbf{a} \cdot \mathbf{x}$, whose value is also used in place of $\mathbf{a} \cdot \beta^2$ in the proof of theorem 3. Then, as in the proof of theorem 4, we identify the nodes in (η^1, η^2) of the proof of theorem 3 with the nodes in (β^1, β^2) in the proof of theorem 4. It is straightforward to see that this modification gives a perfect autoencoder. □

Next, we consider memorizers, that is, functions for finite input samples. The following theorem is shown in Zhang et al. (2017).

Theorem 6 *(Zhang et al., 2017). There exists a three-layer neural network with ReLU activations and $2n + D$ weights that can represent any function on a sample of size n in D dimensions.*

For ease of understanding, we explain this theorem briefly. Here, $X = \{\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{n-1}\}$ denotes a set of n D-dimensional real input vectors that are all different, and $Y = \{y_0, y_1, \ldots, y_{n-1}\}$ denotes a set of n 1-dimensional real outputs that all $y_i \geqslant 0$.

Then there exist weights

$$w_0, w_1, \ldots, w_{n-1}, b_0, b_1, \ldots, b_{n-1}, \mathbf{a},$$

such that $b_0 < \mathbf{a} \cdot \mathbf{x}^0 < b_1 < \mathbf{a} \cdot \mathbf{x}^1 < \cdots < b_{n-1} < \mathbf{a} \cdot \mathbf{x}^{n-1}$, and a three-layer neural network with ReLU activation functions can be designed as follows:

$$z_j = \max\left(\mathbf{a} \cdot \mathbf{x} - b_j, 0\right), \quad \text{where } j \in [0 : n-1],$$

$$y = \max\left(\sum_{j=0}^{n-1} w_j z_j, 0\right).$$

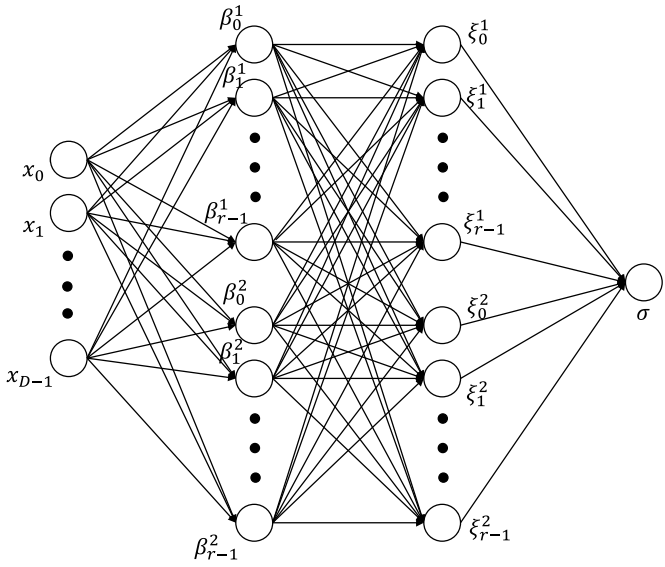

Figure 3: Four-layer ReLU neural network. The circle nodes represent those carrying input and output information. The arrows connecting the circle nodes show that the states of nodes in the ith ($i = 2, 3, 4$) layer are determined based on the ReLU activation functions related to the states of nodes in the $(i-1)$th layer.

Now, we consider reducing the number of nodes in the middle layer from $O(n)$ to $O(\sqrt{n})$ (although the number of layers increases by one).

Theorem 7. *Let $r = \lceil \sqrt{n} \rceil$. For any set of D-dimensional real vectors,*

$$X = \{\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{n-1}\},$$

and any set $Y = \{y_0, y_1, \ldots, y_{n-1}\}$ that all $y_i \in [-1, 1]$, $i \in [0 : n-1]$, there exists a four-layer ReLU neural network with $2r$ nodes in the second layer and $2r$ nodes in the third layer (see also Figure 3) whose output value satisfies $\sigma = y_i + 1$ for each input vector $\mathbf{x}^i$.

Proof. Here, the four-layer ReLU neural network we designed is based on the four-layer fully connected neural network of theorem 3.1 in Yun et al. (2019). Notably, in the proof of theorem 3.1 (Yun et al., 2019), it mainly indicates that there is a four-layer fully connected neural network using hard-tanh activation functions, which can fit any arbitrary data set $\{(x_i, y_i)\}_{i=1}^N$, where all inputs $x_i \in R^{d_x}$ are distinct and all $y_i \in [-1, 1]$. Although our construction is very similar to their construction, there are some differences because they use a neural network with hard-tanh activation functions.

Therefore, in this proof, we mainly describe our construction of the ReLU neural network with briefly explaining the differences from that in Yun et al. (2019). The full proof is given in the online supplemental material.

For ease of exposition, we assume that $r = \sqrt{n}$ and r is a multiple of 2. Divide total n input vectors into r groups with r vectors each.

First, let the ReLU activation functions for β_j^1, $j \in [0 : r-1]$ be

$$\max\left(f_1(\mathbf{x}) + 1, 0\right),$$

and the ReLU activation functions for β_j^2, $j \in [0 : r-1]$ be

$$\max\left(f_1(\mathbf{x}) - 1, 0\right),$$

where

$$f_1(\mathbf{x}) = (-1)^{j-1} \frac{4\mathbf{a} \cdot \mathbf{x}}{c_{jr+r-1} + c_{(j+1)r} - c_{(j-1)r+r-1} - c_{jr}} + (-1)^j \frac{c_{jr+r-1} + c_{(j+1)r} + c_{(j-1)r+r-1} + c_{jr}}{c_{jr+r-1} + c_{(j+1)r} - c_{(j-1)r+r-1} - c_{jr}},$$

where each c_i is a constant given in equation 2.1. Here, we should note that $\gamma_j = \beta_j^1(\mathbf{x}) - \beta_j^2(\mathbf{x}) - 1$ is similar to the output of the jth node of the first hidden layer $\alpha_j^1(\mathbf{x})$ in theorem 3.1 (Yun et al., 2019). More specifically, when j is even, we have

$$-1 < \beta_j^1(\mathbf{x}^{jr+r-1}) - \beta_j^2(\mathbf{x}^{jr+r-1}) - 1 < \cdots < \beta_j^1(\mathbf{x}^{jr}) - \beta_j^2(\mathbf{x}^{jr}) - 1 < 1,$$
$$\beta_j^1(\mathbf{x}^i) - \beta_j^2(\mathbf{x}^i) - 1 = -1, \quad \text{if } i > jr + r - 1,$$
$$\beta_j^1(\mathbf{x}^i) - \beta_j^2(\mathbf{x}^i) - 1 = 1, \quad \text{if } i \le (j-1)r + r - 1.$$

When j is odd, we have

$$-1 < \beta_j^1(\mathbf{x}^{jr}) - \beta_j^2(\mathbf{x}^{jr}) - 1 < \cdots < \beta_j^1(\mathbf{x}^{jr+r-1}) - \beta_j^2(\mathbf{x}^{jr+r-1}) - 1 < 1,$$
$$\beta_j^1(\mathbf{x}^i) - \beta_j^2(\mathbf{x}^i) - 1 = -1, \quad \text{if } i < jr,$$
$$\beta_j^1(\mathbf{x}^i) - \beta_j^2(\mathbf{x}^i) - 1 = 1, \quad \text{if } i \ge (j+1)r.$$

In the next layer, let the ReLU activation functions for $\xi_k^1, k \in [0 : r-1]$ be

$$\max\left(\sum_{l=0}^{r-1} w_{k,l}(\beta_l^1(\mathbf{x}) - \beta_l^2(\mathbf{x}) - 1) + b_k + 1, 0\right)$$

and the ReLU activation functions for $\xi_k^2, k \in [0 : r-1]$ be

$$\max\left(\sum_{l=0}^{r-1} w_{k,l}(\beta_l^1(\mathbf{x}) - \beta_l^2(\mathbf{x}) - 1) + b_k - 1, 0\right),$$

where $\mathbf{w}_k = [w_{k,0}, w_{k,1}, \ldots, w_{k,r-1}]$ and b_k are solutions of the following equations:

$$\sum_{l=0}^{r-1} w_{k,l}\gamma_l(\mathbf{x}^{i_{k,j}}) + b_k = y_{i_{k,j}},\ j \in [0 : r-1],$$

under the condition that each element of $\mathbf{w}_k$ is sufficiently large negative (resp., positive) when k is even (resp., odd). The main difference from Yun et al. (2019) is that we define set $\mathcal{I}_k, k \in [0 : r-1]$ as

$$\mathcal{I}_k := \{k, 2r-1-k, 2r+k, 4r-1-k, \ldots, r^2-1-k\},$$

where $i_{k,j}, j \in [0 : r-1]$ are used to represent the elements in the set $\mathcal{I}_k$, and $i_{k,0} = k, i_{k,1} = 2r-1-k, \ldots, i_{k,r-1} = r^2-1-k$.

Finally, in the last layer, let $\sigma = \max\left(\sum_{l=0}^{r-1}(\xi_l^1 - \xi_l^2 - 1), 0\right)$. Here, $\xi_l^1(\mathbf{x}) - \xi_l^2(\mathbf{x}) - 1, l \in [0 : r-1]$ is similar to the output of the lth node of the second hidden layer $\alpha_l^2(\mathbf{x})$ in theorem 3.1, and for $j \in [0 : r-1]$ and $k \in [0 : r-1]$, we also have

$$\sigma = \sum_{l=0}^{r-1}(\xi_l^1(\mathbf{x}^{i_{k,j}}) - \xi_l^2(\mathbf{x}^{i_{k,j}}) - 1) = y_{i_{k,j}} + 1.$$

□

Remark 5. When $y_i, i \in [0 : n-1]$ is any real number, we can also construct a four-layer ReLU neural network. First, let

$$y_{\max} = \max(y_0, y_1, \ldots, y_{n-1}), y_{\min} = \min(y_0, y_1, \ldots, y_{n-1})$$

and

$$y_{\text{mean}} = \frac{y_0 + y_1 + \cdots + y_{n-1}}{n}.$$

By means of $z_i = \frac{y_i - y_{\text{mean}}}{y_{\max} - y_{\min}}$, we can scale the elements $y_i \in Y$ to $-1 \leqslant z_i \leqslant 1$, and then put $\{z_0, z_1, \ldots, z_{n-1}\}$ as outputs; then we can construct a four-layer ReLU neural network as shown in theorem 7. Only for the last layer do we change it to

$$\sigma = (y_{\max} - y_{\min}) \cdot \max\left(\sum_{l=0}^{r-1}(\xi_l^1 - \xi_l^2 - 1), 0\right) + y_{\text{mean}} - y_{\max} + y_{\min}.$$

Remark 6. It is worth noting that theorem 1 in Zhang et al. (2017) (i.e., theorem 6 in this article) shows that there exists a two-layer neural network. Similarly, theorem 3.1 in Yun et al. (2019) shows that there is a three-layer hard-tanh, fully connected neural network. However, since here we regard the input layer as the first layer, to avoid confusion, the number of layers is increased by 1 when we describe the neural networks of theorems 1 and 3.1 in this article.

5 Computational Experiments

In order to test whether some of the autoencoder (memorizer) architectures obtained through theoretical analyses can be used in the design of practical neural networks, we conducted some computational experiments using neural networks. Here, the activation functions are learned from input and output data. All numerical experiments were conducted on a PC with Xeon Gold 5222 CPU and A100 GPU under the Ubuntu 18.04.

First, we performed computational experiments on theorem 4, considering the use of 256 36-dimensional real vectors as input vectors and 324 40-dimensional real vectors as input vectors, respectively. For the training of neural networks with gaussian error linear unit (GELU) functions (Dubey et al., 2022), we employed the Adam optimizer in PyTorch with a learning rate 0.01 and 800 epochs repetition. The specific process is as follows.

Step 1: Generate a neural network $\mathcal{N}$ with the architecture given in theorem 4.
Step 2: Randomly generate D-dimensional real vectors $\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{n-1}$, where each $x_j^i \in [0, 2]$, $i = 0, 1, \ldots, n-1$, $j = 0, 1, \ldots, D-1$.
Step 3: Train $\mathcal{N}$ using $\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{n-1}$ for both input and output data.
Step 4: Compute the training accuracy of the trained $\mathcal{N}$ using $\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{n-1}$ as the input data. Assuming $\mathbf{y}^0, \mathbf{y}^1, \ldots, \mathbf{y}^{n-1}$ are the corresponding output data, the training accuracy is defined as

$$\frac{\sum_{i=0}^{n-1}\sum_{j=0}^{D-1}\mathcal{I}_{[-0.2,0.2]}\left(y_j^i - x_j^i\right)}{n \cdot D}, \tag{5.1}$$

where $\mathcal{I}_{[-0.2,0.2]}\left(y_j^i - x_j^i\right)$ is an indicator function, that is,

$$\mathcal{I}_{[-0.2,0.2]}\left(y_j^i - x_j^i\right) = \begin{cases} 1 & \text{if } y_j^i - x_j^i \in [-0.2, 0.2], \\ 0 & \text{otherwise.} \end{cases}$$

Table 2: Results of Computational Experiments on Theorem 4.

Architecture	D	n	d	Training Accuracy
Theorem 4	36	256	32	0.9899
Theorem 4	40	324	36	0.9837

Table 3: Results of Computational Experiments on Theorem 7.

Architecture	D	n	d	Training Accuracy
Theorem 7	36	256	32	1.0000
Theorem 7	40	324	36	1.0000

We repeated this procedure 100 times to obtain the average training accuracy; the results are shown in Table 2.

In the above experiment, we used the GELU function instead of the ReLU function, mainly because we found that the training accuracy obtained by training a neural network with ReLU functions was not ideal. We believe that the main reason for this phenomenon is that the neural network constructed by theorem 4 has a slightly large number of layers and relatively complex connection methods for each layer. Using the GELU function, described as a smoother version of the ReLU function, for neural network training can provide smoother gradients, which can help maintain gradient flow in complex models and avoid problems such as vanishing or exploding gradients. In contrast, when the data range is small and the network hierarchy is complex, using the ReLU function may lead to numerical instability, thereby affecting the convergence and final performance of the model.

Subsequently, we also conducted experiments using neural networks with ReLU functions on theorem 7, and the process is similar to that mentioned above. Here, generate a neural network $\mathcal{N}$ with the architecture given in theorem 7 and randomly generate data set $\{\mathbf{x}^i, y_i\}_{i=0}^{n-1}$, where $\mathbf{x}^i \in R^D$ (each $x_j^i \in [0,1]$, $i = 0, 1, \ldots, n-1$, $j = 0, 1, \ldots, D-1$) is the input and $y_i \in [-1, 1]$ is the output. Consider two cases: $D = 36$, $n = 256$ and $D = 40$, $n = 324$. We also repeated this procedure 100 times and recorded the average training accuracy in Table 3. Here, the training accuracy is defined as

$$\frac{\sum_{i=0}^{n-1} \mathcal{I}_{[-0.2,0.2]}\left(\hat{y}_i - y_i\right)}{n}, \tag{5.2}$$

where $\hat{y}_i$ is the output obtained by the trained neural network $\mathcal{N}$ using $\mathbf{x}^i$ as the input data and

$$\mathcal{I}_{[-0.2,0.2]}\left(\hat{y}_i - y_i\right) = \begin{cases} 1 & \text{if } \hat{y}_i - y_i \in [-0.2, 0.2], \\ 0 & \text{otherwise.} \end{cases}$$

The above experimental results indicate that our proposed architectures may provide useful insights for designing practical autoencoders (memorizer).

6 Discussion

In this article, we studied relations between the compressed vectors and the depth and width (the number of node in a layer) of autoencoders with real input and output vectors using linear and ReLU activation functions, under the condition that the input and output vectors must be the same. The results on ReLU activation functions suggest that we can achieve the same compression ratio as in the case of binary input and output vectors (Melkman et al., 2023) using similar archtectures. The results are interesting because real input and output vectors can be handled by replacing linear threshold activation functions with ReLU activation functions, where some modifications are required. Furthermore, the results on linear activation functions suggest that ReLU activation functions are much more powerful than linear activation functions with respect to autoencoders with real input and output vectors.

Although we have mainly given upper bounds on the depth and width, we have not shown any lower bounds for autoencoders using ReLU activation functions. Note that a lower bound of $\Omega(\sqrt{Dn/d})$ is known for the width of autoencoders using linear threshold activation functions (Akutsu & Melkman, 2023). However, the techniques used there heavily depend on properties of Boolean functions and thus cannot be applied to the case of ReLU activation functions. Thus, showing lower bounds on autoencoders with ReLU activation functions is left as future work. For the case of $d = \lceil \log n \rceil$, an $O(\sqrt{Dn})$ upper bound is shown for the width of autoencoders using linear threshold activation functions (see corollary 8 of Akutsu & Melkman, 2023), which is better than an $O(D\sqrt{n})$ upper bound shown in Melkman et al. (2023) and this article. Hence, development of an autoencoder with width $O(\sqrt{Dn})$ using ReLU activation functions is also left as future work. Furthermore, it is interesting to study whether it is possible to develop autoencoders with a smaller width using ReLU activation functions than those using linear threshold activation functions.

In our definition of the perfect autoencoder, we assumed that the input vectors must be the same as the output vectors. But in practical situations, the input and output vectors need not be the same but should be similar. For the case of autoencoders with linear threshold activation functions, it is shown that the width of the decoder part can be reduced by a constant factor if some Hamming distance error is allowed (Akutsu & Melkman, 2023). However, the techniques used in that study cannot be applied to real input and output vectors because the Hamming distance cannot be directly generalized to real vectors and the construction of autoencoders heavily depends on binary values. Therefore, conducting theoretical studies on autoencoders

allowing errors between real input and output vectors is interesting and important future work.

Another drawback of this work is that the proposed design methods are somewhat ad hoc. However, most of the designed architectures are based on existing ones for binary vectors using linear threshold activation functions, and the main purpose of this article is to extend such existing ones to a more practical setting (i.e., real vectors using ReLU activation functions). Therefore, this work can be considered an important step toward understanding data compression mechanisms of autoencoders in more practical settings. Of course, many other practical activation functions are known (Apicella et al., 2021; Dubey et al., 2022). Therefore, important future work is to develop more general design methods that can be applied to many practical activation functions.

Acknowledgments

T.A. was partially supported by grants-in-aid 22H00532 and 22K19830 from JSPS, Japan; W.C. was partially supported by Hong Kong RGC GRF grant 17301519, IMR, and Hung Hing Ying Physical Sciences Research Fund, HKU.

References

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, *9*(1), 147–169.

Akutsu, T., & Melkman, A. A. (2023). On the size and width of the decoder of a Boolean threshold autoencoder. *IEEE Transactions on Neural Networks and Learning Systems*, (PP)*99*, 1–8. 10.1109/TNNLS.2023.3342818

Apicella, A., Donnarumma, F., Isgró, F., & Prevete, R. (2021). A survey on modern trainable activation functions. *Neural Networks*, *138*, 14–32. 10.1016/j.neunet.2021.01.026

Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, *27* (pp. 37–49).

Baldi, P., & Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, *2*(1), 53–58. 10.1016/0893-6080(89)90014-2

Delalleau, O., & Bengio, Y. (2011). Shallow vs. deep sum-product networks. In S. Hanson, J. Cowan, & C. Giles (Eds.), *Advances in neural information processing systems*, *24* (pp. 666–674). MIT Press.

Doersch, C. (2016). *Tutorial on variational autoencoders*. arXiv:1606.05908.

Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2022). Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, *503*, 92–108. 10.1016/j.neucom.2022.06.111

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, Sánchez-Lengeling, Sheberla, . . . Aspuru-Guzik, A. (2018). Automatic chemical design

using a data-driven continuous representation of molecules. *ACS Central Science*, *4*(2), 268–276.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507. 10.1126/science.1127647

Kärkkäinen, T., & Hänninen, J. (2023). Additive autoencoder for dimension estimation. *Neurocomputing*, *551*, 126520.

Kumano, S., & Akutsu, T. (2022). Comparison of the representational power of random forests, binary decision diagrams, and neural networks. *Neural Computation*, *34*(4), 1019–1044. 10.1162/neco_a_01486

Lee, S., & Jo, J. (2021). Information flows of diverse autoencoders. *Entropy*, *23*(7), 862.

Melkman, A. A., Guo, S., Ching, W-K., Liu, P., & Akutsu, T. (2023). On the compressive power of Boolean threshold autoencoders. *IEEE Transactions on Neural Networks and Learning Systems*, *34*(2), 92–931. 10.1109/TNNLS.2021.3104646

Montufar, G. F., Pascanu, R., Cho, K., & Bengio, Y. (2014). On the number of linear regions of deep neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *27* (pp. 2924–2932). Curran.

Tapia, N. I., & Estévez, P. A. (2020). On the information plane of autoencoders. In *Proceedings of the 2020 International Joint Conference on Neural Networks* (pp. 1–8).

Tschannen, M., Bachem, O., & Lucic, M. (2018). *Recent advances in autoencoder-based representation learning*. arXiv:1812.05069.

Ubaru, S., & Saad, Y. (2016). Fast methods for estimating the numerical rank of large matrices. In *Proceedings of the 33rd International Conference on Machine Learning* (pp. 468–477).

Vershynin, R. (2020). Memory capacity of neural networks with threshold and rectified linear unit activations. *SIAM Journal on Mathematics of Data Science*, *2*(4), 1004–1033. 10.1137/20M1314884

Yu, S., & Príncipe, J. C. (2019). Understanding autoencoders with information theoretic concepts. *Neural Networks*, *117*, 104–123. 10.1016/j.neunet.2019.05.003

Yun, C., Sra, S., & Jadbabaie, A. (2019). Small ReLU networks are powerful memorizers: A tight analysis of memorization capacity. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*, *32* (pp. 15532–15543). Curran.

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. In *Proceedings of the International Conference on Learning Representations*.

Received April 3, 2024; accepted October 2, 2024.