Scientific Research Publishing

American Journal of Computational Mathematics, 2018, 8, 184-196
http://www.scirp.org/journal/ajcm
ISSN Online: 2161-1211
ISSN Print: 2161-1203

Completeness Problem of the Deep Neural Networks

Ying Liu[1], Shaohui Wang[2]

[1]Department of Engineering Technology, Savannah State University, Savannah, Georgia, USA
[2]Department of Mathematics, Savannah State University, Savannah, Georgia, USA
Email: liuy@savannahstate.edu, wangs@savannahstate.edu

How to cite this paper: Liu, Y. and Wang, S.H. (2018) Completeness Problem of the Deep Neural Networks. *American Journal of Computational Mathematics*, **8**, 184-196.
https://doi.org/10.4236/ajcm.2018.82014

Received: April 27, 2018
Accepted: June 26, 2018
Published: June 29, 2018

Abstract

Hornik, Stinchcombe & White have shown that the multilayer feed forward networks with enough hidden layers are universal approximators. Roux & Bengio have proved that adding hidden units yield a strictly improved modeling power, and Restricted Boltzmann Machines (RBM) are universal approximators of discrete distributions. In this paper, we provide yet another proof. The advantage of this new proof is that it will lead to several new learning algorithms. We prove that the Deep Neural Networks implement an expansion and the expansion is complete. First, we briefly review the basic Boltzmann Machine and that the invariant distributions of the Boltzmann Machine generate Markov chains. We then review the θ-transformation and its completeness, *i.e.* any function can be expanded by θ-transformation. We further review ABM (Attrasoft Boltzmann Machine). The invariant distribution of the ABM is a θ-transformation; therefore, an ABM can simulate any distribution. We discuss how to convert an ABM into a Deep Neural Network. Finally, by establishing the equivalence between an ABM and the Deep Neural Network, we prove that the Deep Neural Network is complete.

Keywords

AI, Universal Approximators, Boltzmann Machine, Markov Chain, Invariant Distribution, Completeness, Deep Neural Network

1. Introduction

Neural networks and deep learning currently provide the best solutions to many supervised learning problems. In 2006, a publication by Hinton, Osindero, and Teh [1] introduced the idea of a "deep" neural network, which first trains a simple supervised model; then adds on a new layer on top and trains the parameters

for the new layer alone. You keep adding layers and training layers in this fashion until you have a deep network. Later, this condition of training one layer at a time is removed [2] [3] [4] [5].

After Hinton's initial attempt of training one layer at a time, Deep Neural Networks train all layers together. Examples include TensorFlow [6], Torch [7], and Theano [8]. Google's TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and also used for machine learning applications such as neural networks [3]. It is used for both research and production at Google. Torch is an open source machine learning library and a scientific computing framework. Theano is a numerical computation library for Python. The approach using the single training of multiple layers gives advantages to the neural network over other learning algorithms.

One question is the existence of a solution for a given problem. This will often be followed by an effective solution development, *i.e.* an algorithm for a solution. This will often be followed by the stability of the algorithm. This will often be followed by an efficiency study of solutions. Although these theoretical approaches are not necessary for the empirical development of practical algorithms, the theoretical studies do advance the understanding of the problems. The theoretical studies will prompt new and better algorithm development of practical problems. Along the direction of solution existence, Hornik, Stinchcombe, & White [9] have shown that the multilayer feedforward networks with enough hidden layers are universal approximators. Roux & Bengio [10] have shown the same, Restricted Boltzmann machines are universal approximators of discrete distributions.

Hornik, Stinchcombe, & White [9] establish that the standard multilayer feedforward networks with hidden layers using arbitrary squashing functions are capable of approximating any measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.

Deep Belief Networks (DBN) are generative neural network models with many layers of hidden explanatory factors, recently introduced by Hinton, Osindero, and Teh, along with a greedy layer-wise unsupervised learning algorithm. The building block of a DBN is a probabilistic model called a Restricted Boltzmann machine (RBM), used to represent one layer of the model. Restricted Boltzmann machines are interesting because inference is easy in them and because they have been successfully used as building blocks for training deeper models. Roux & Bengio [10] proved that adding hidden units yield a strictly improved modeling power, and RBMs are universal approximators of discrete distributions.

In this paper, we provide yet another proof. The advantage of this proof is that it will lead to several new learning algorithms. We once again prove that Deep

Neural Networks are universal approximators. In our approach, Deep Neural Networks implement an expansion and this expansion is complete.

In this paper, a Deep Neural Network (DNN) is an Artificial Neural Network (ANN) with multiple hidden layers between the input and output layers. The organization of this paper is as follows.

In Section 2, we briefly review how to study the completeness problem of Deep Neural Networks (DNN). In this approach, given an input A, an output B, and a mapping from A to B, one can convert this problem to a probability distribution [3] [4] of (A, B): $p(a, b)$, $a \in A$, $b \in B$. If an input is $a \in A$ and an output is $b \in B$, then the probability $p(a, b)$ will be close to 1. One can find a Markov chain [11] such that the equilibrium distribution of this Markov chain, $p(a, b)$, realizes, as faithfully as possible, the given supervised training set.

In Section 3, the Boltzmann machines [3] [4] are briefly reviewed. All possible distributions together form a distribution space. All of the distributions, implemented by Boltzmann machines, define a Boltzmann Distribution Space, which is a subset of the distribution space [12] [13] [14]. Given an unknown function, one can find a Boltzmann machine such that the equilibrium distribution of this Boltzmann machine realizes, as faithfully as possible, the unknown function.

In Section 4, we review the ABM (Attrasoft Boltzmann Machine) [15] which has an invariant distribution. An ABM is defined by two features: 1) an ABM with n neurons has neural connections up to the n^{th} order; and 2) all of the connections up to n^{th} order are determined by the ABM algorithm [15]. By adding more terms in the invariant distribution compared to the second order Boltzmann Machine, ABM is significantly more powerful in simulating an unknown function. Unlike Boltzmann Machine, ABM's emphasize higher order connections rather than lower order connections. Later, we will discuss the relationships between the higher order connections and DNN.

In Section 5, we review θ-transformation [12] [13] [14].

In Section 6, we review the completeness of the θ-transformation [12] [13] [14]. The θ-transformation is complete; *i.e.* given a function, one can find a θ-transformation to convert it from the x-coordinate system to the θ-coordinate system.

In Section 7, we discuss how the invariant distribution of an ABM implements a θ-transformation [12] [13] [14], *i.e.* given an unknown function, one can find an ABM such that the equilibrium distribution of this ABM realizes precisely the unknown function. Therefore, an ABM is complete.

The next two sections are the new contributions of this paper. In section 8, we show that we can reduce an ABM to a DNN, *i.e.* we show that a higher order ANN can be replaced by a lower order ANN by increasing layers. We do not seek an efficient conversion from a higher order ANN to a lower order ANN with more layers. We will merely prove this is possible.

In Section 9, we prove that the DNN is complete, *i.e.* given an unknown function, one can find a Deep Neural Network that can simulate the unknown function.

2. Basic Approach for Completeness Problem

The goal of this paper is to prove that given any unknown function from A to B, one can find a DNN such that it can simulate this unknown function. It turns out that if we can reduce this from a discrete problem to a continuous problem, it will be very helpful. In this section, we introduce the basic idea of how to study the completeness problem.

The basic supervised learning [2] problem is: given a training set $\{A, B\}$, where $A=\{a_1,a_2,\cdots\}$ and $B=\{b_1,b_2,\cdots\}$, find a mapping from A to B. The first step is to convert this problem to a probability [3] [4]:

$$p=p(a,b), a\in A, b\in B.$$

If a_1 matches with b_1, the probability is 1 or close to 1. If a_1 does not match with b_1, the probability is 0 or close to 0. This can reduce the problem of inferencing a mapping from A to B, to inferencing a distribution function.

An irreducible finite Markov chain possesses a stationary distribution [11]. This invariant distribution can be used to simulate an unknown function. It is the invariant distribution of the Markov Chain which eventually allows us to prove that the DNN is complete.

3. Boltzmann Machine

A Boltzmann machine [3] [4] is a stochastic neural network in which each neuron has a certain probability to be 1. The probability of a neuron to be 1 is determined by the so called Boltzmann distribution. The collection of the neuron states:

$$x=(x_1,x_2,\cdots,x_n)$$

of a Boltzmann machine is called a configuration. The configuration transition is mathematically described by a Markov chain with 2^n configurations $x\in X$, where X is the set of all points, $(x_1,x_2,\cdots,x_n)$ When all of the configurations are connected, it forms a Markov chain. A Markov chain has an invariant distribution [11]. Whatever initial configuration a Boltzmann machine starts from, the probability distribution converges over time to the invariant distribution, $p(x)$. The configuration $x\in X$ appears with a relative frequency $p(x)$ over a long period of time.

The Boltzmann machine [3] [4] defines a Markov chain. Each configuration of the Boltzmann machine is a state of the Markov chain. The Boltzmann machine has a stable distribution. Let T be the parameter space of a family of Boltzmann machines. An unknown function can be considered as a stable distribution of a Boltzmann machine. Given an unknown distribution, a Boltzmann machine can be inferred such that its invariant distribution realizes, as faithfully as possible, the given function. Therefore, an unknown function is transformed into a specification of a Boltzmann machine.

More formally, let F be the set of all functions. Let T be the parameter space of a family of Boltzmann machines. Given an unknown $f\in F$, one can find a

Boltzmann machine such that the equilibrium distribution of this Boltzmann machine realizes, as faithfully as possible, the unknown function [3] [4]. Therefore, the unknown, f, is encoded into a specification of a Boltzmann machine, $t \in T$. We call the mapping from F to T, a Boltzmann Machine Transformation: $F \to T$ [12] [13] [14].

Let T be the parameter space of a family of Boltzmann machines, and let F_T be the set of all functions that can be inferred by the Boltzmann Machines over T; obviously, F_T is a subset of F. It turns out that F_T is significantly smaller than F and that it is not a good approximation for F. The main contribution of the Boltzmann Machine is to establish a framework for inferencing a mapping from A to B.

4. Attrasoft Boltzmann Machines (ABM)

The invariant distribution of a Boltzmann machine [3] [4] is:

$$p(x) = b\mathrm{e}^{\sum_{i<j} M_{ij} x_i x_j} \tag{1}$$

If the threshold vector does not vanish, the distributions are:

$$p(x) = b\mathrm{e}^{\sum_{i<j} M_{ij} x_i x_j - \sum T_i x_i}. \tag{2}$$

By rearranging the above distribution, we have:

$$p(x) = \mathrm{e}^{c - \sum T_i x_i + \sum_{i<j} M_{ij} x_i x_j} \tag{3}$$

It turns out that the third order Boltzmann machines have the following type of distributions:

$$p(x) = \mathrm{e}^{c - \sum T_i x_i + \sum_{i<j} M_{ij} x_i x_j + \sum_{i<j<k} M_{ijk} x_i x_j x_k} \tag{4}$$

An ABM [12] [13] [14] is an extension of the higher order Boltzmann Machine to the maximum order. An ABM with n neurons has neural connections up to the n^{th} order. All of the connections up to the n^{th} order are determined by the ABM algorithm [15]. By adding additional higher order terms to the invariant distribution, ABM is significantly more powerful in simulating an unknown function.

By adding additional terms, the invariant distribution for an ABM is,

$$p(x) = \mathrm{e}^H,$$

$$H = \theta_0 + \sum \theta_1^{i_1} x_{i_1} + \sum \theta_2^{i_1 i_2} x_{i_1} x_{i_2} + \sum \theta_3^{i_1 i_2 i_3} x_{i_1} x_{i_2} x_{i_3} + \cdots + \theta_n^{12\cdots n} x_1 x_2 \cdots \tag{5}$$

ABM is significantly more powerful in simulating an unknown function. As more and more terms are added, from the second order terms to the n^{th} order terms, the invariant distribution space will become larger and larger. Like Boltzmann Machines of the last section, ABM implements a transformation, $F_B \to T$. Our goal ultimately that this ABM transformation is complete so that given any function $f \in F$, we can find an ABM, $t \in T$, such that the equilibrium distribution of this ABM realizes precisely the unknown function. We show that this is exactly the case.

5. θ-Transformation

5.1. Basic Notations

We first introduce some notations used in this paper [12] [13] [14]. There are two different types of coordinate systems: the x-coordinate system and the θ-coordinate system [12] [13] [14]. Each of these two coordinate systems has two representations, x-representation and θ-representation. An N-dimensional vector, p, is:

$$p = (p_0, p_1, \cdots, p_{N-1}),$$

which is the x-representation of p in the x-coordinate systems.

In the x-coordinate system, there are two representations of a vector:

- $\{p_i\}$ in the x-representation and
- $\left\{p_m^{i_1 i_2 \cdots i_m}\right\}$ in the θ-representation.

In the θ-coordinate system, there are two representations of a vector:

- $\{\theta_i\}$ in the x-representation and
- $\left\{\theta_m^{i_1 i_2 \cdots i_m}\right\}$ in the θ-representation.

The reason for two different representations is that the x-representation is natural for the x-coordinate system, and the θ-representation is natural for the θ-coordinate system.

The transformations between $\{p_i\}$ and $\left\{p_m^{i_1 i_2 \cdots i_m}\right\}$, and those between $\{\theta_i\}$ and $\left\{\theta_m^{i_1 i_2 \cdots i_m}\right\}$, are similar. Because of the similarity, in the following, only the transformation between $\{p_i\}$ and $\left\{p_m^{i_1 i_2 \cdots i_m}\right\}$ will be introduced. Let $N = 2^n$ be the number of neurons. An N-dimensional vector, p, is:

$$p = (p_0, p_1, \cdots, p_{N-1}).$$

Consider p_x, because $x \in \{0, 1, \cdots, N-1 = 2^n - 1\}$ is the position inside a distribution, then x can be rewritten in the binary form:

$$x = x_n 2^{n-1} + \cdots + x_2 2^1 + x_1 2^0.$$

Some of the coefficients x_i might be zero. In dropping those coefficients which are zero, we write:

$$x = x_{i_1} x_{i_2} \cdots x_{i_m} = 2^{i_m - 1} + \cdots + 2^{i_2 - 1} + 2^{i_1 - 1}.$$

This generates the following transformation:

$$p_m^{i_1 i_2 \cdots i_m} = p_x = p_{2^{i_m-1}+\cdots+2^{i_2-1}+2^{i_1-1}} \text{ where}$$
$$1 \le i_1 < i_2 < \cdots < i_m \le n.$$

In this θ-representation, a vector p looks like:

$$\left\{p_0, p_1^1, p_1^2, p_1^3, \cdots, p_2^{12}, p_2^{13}, p_2^{23}, \cdots, p_3^{123}, \cdots\right\} \tag{6}$$

The 0-th order term is p_0, the first order terms are: $p_1^1, p_1^2, p_1^3, \cdots$, … The first few terms in the transformation between $\{p_i\}$ and $\left\{p_m^{i_1 i_2 \cdots i_m}\right\}$ are:

$$p_0 = p_0, \quad p_1^1 = p_1, \quad p_1^2 = p_2,$$
$$p_2^{12} = p_3, \quad p_1^3 = p_{14}, \quad p_2^{13} = p_5$$
$$p_2^{23} = p_6, \quad p_3^{123} = p_7, \quad p_1^4 = p_8, \cdots$$

The x-representation is the normal representation, and the θ-representation is

a form of binary representation.

Example Let $n = 3$, $N = 2^n = 8$, and consider an invariant distribution:

$$\{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7\},$$

where p_0 is the probability of state $x = 0, \cdots$. There are 8 probabilities for 8 different states, $x = \{0, 1, 2, \cdots, 7\}$. In the new representation, it looks like:

$$\{p_0, p_1^1, p_1^2, p_1^3, p_2^{12}, p_2^{13}, p_2^{23}, p_3^{123}\}.$$

Note that the relative positions of each probability are changed. The first vector, $\{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$, is in the x-representation and the second vector $\{p_0, p_1^1, p_1^2, p_1^3, p_2^{12}, p_2^{13}, p_2^{23}, p_3^{123}\}$ is in the θ-representation. These two representations are two different expressions of the same vector.

5.2. θ-Transformation

Denote a distribution by p, which has a x-representation in the x-coordinate system, $p(x)$, and a θ-representation in the θ-coordinate system, $p(\theta)$. When a distribution function, $p(x)$ is transformed from one coordinate system to another, the vectors in both coordinates represent the same abstract vector. When a vector q is transformed from the x-representation $q(x)$ to the θ-representation $q(\theta)$, and then $q(\theta)$ is transformed back to $q'(x)$, $q'(x) = q(x)$.

The θ-transformation uses a function F, called a generating function. The function F is required to have the inverse:

$$FG = GF = I, \quad G = F^{-1}. \tag{7}$$

Let p be a vector in the x-coordinate system. As already discussed above, it can be written either as:

$$p(x) = (p_0, p_1, \cdots, p_{N-1}) \tag{8}$$

Or

$$p(x) = \left(p_0; p_1^1, \cdots, p_1^n; p_2^{12}, \cdots, p_2^{n-1,n}; p_3^{123}, \cdots, p_n^{12\cdots n}\right). \tag{9}$$

The θ-transformation transforms a vector from the x-coordinate to the θ-coordinate via a generating function. The components of the vector p in the x-coordinate, $p(x)$, can be converted into components of a vector $p(\theta)$ in the θ-coordinate:

$$p(\theta) = \left(\theta_0; \theta_1^1, \cdots, \theta_1^n; \theta_2^{12}, \cdots, \theta_2^{n-1,n}; \theta_3^{123}, \cdots, \theta_n^{12\cdots n}\right), \tag{10}$$

Or

$$p(\theta) = (\theta_0, \theta_1, \cdots, \theta_{N-1}). \tag{11}$$

Let F be a generating function, which transforms the x-representation of p in the x-coordinate to a θ-representation of p in the θ-coordinate system. The θ-components are determined by the vector $F[p(x)]$ as follows:

$$F\left[p(x)\right] = \theta_0 + \sum \theta_1^{i_1} x_{i_1} + \sum \theta_2^{i_1 i_2} x_{i_1} x_{i_2} + \sum \theta_3^{i_1 i_2 i_3} x_{i_1} x_{i_2} x_{i_3} + \cdots + \theta_n^{12\cdots n} x_1 x_2 \cdots x_n \tag{12}$$

where

$$1 \le i_1 < i_2 < \cdots < i_m \le n.$$

Prior to the transformation, $p(x)$ is the x-representation of p in the x-coordinate; after transformation, $F[p(x)]$ is a θ-representation of p in the θ-coordinate system.

There are N components in the x-coordinate and N components in the θ-coordinate. By introducing a new notation X:

$$X_0 = X_0 = 1, \quad X_1^2 = X_1 = x_1, \quad X_1^2 = X_2 = x_2, \quad X_2^{12} = X_3 = x_1 x_2,$$
$$X_1^3 = X_4 = x_3, \quad X_2^{13} = X_5 = x_1 x_3, \quad X_2^{23} = X_6 = x_2 x_3,$$
$$X_3^{123} = X_7 = x_1 x_2 x_3, \quad X_1^4 = X_8 = x_1 x_2 x_3 x_4 \cdots$$

then the vector can be written as:

$$F\left[p(x)\right] = \sum \theta_J X_J \tag{13}$$

By using the assumption $GF = I$, we have:

$$p(x) = G\left\{\sum \theta_J X_J\right\} \tag{14}$$

where J denotes the index in either of the two representations in the θ-coordinate system.

The transformation of a vector p from the x-representation, $p(x)$, in the x-coordinate system to a θ-representation, $p(\theta)$, in the θ-coordinate system is called θ-transformation [12] [13] [14].

The θ-transformation is determined by [12] [13] [14]:

$$\begin{aligned}\theta_m^{i_1 i_2 \cdots i_m} = {} & F\left[p_m^{i_1 i_2 \cdots i_m}\right] + F\left[p_{m-2}^{i_1 \cdots i_{m-2}}\right] + \cdots + F\left[p_{m-2}^{i_3 \cdots i_m}\right] + F\left[p_{m-4}^{\cdots}\right] + \cdots \\ & - F\left[p_{m-1}^{i_1 \cdots i_{m-1}}\right] - \cdots - F\left[p_{m-1}^{i_2 \cdots i_m}\right] - F\left[p_{m-3}^{i_1 \cdots i_{m-3}}\right] - \cdots\end{aligned} \tag{15}$$

The inverse of the θ-transformation [12] [13] [14] is:

$$p_m^{i_1 i_2 \cdots i_m} = G\left(\theta_0 + \theta_1^{i_1} + \theta_1^{i_2} + \cdots + \theta_1^{i_m} + \theta_2^{i_1 i_2} + \theta_2^{i_1 i_3} + \cdots + \theta_2^{i_{m-1} i_m} + \cdots + \theta_m^{i_1 i_2 \cdots i_m}\right). \tag{16}$$

5.3. An Example

Let an ANN have 3 neurons:

$$\left(x_1, x_2, x_3\right)$$

and let a distribution be:

$$\left(p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7\right).$$

Assume that the generating functions are:

$$F(y) = \log(y), \quad G(y) = \exp(y).$$

By θ-transformation, the components are [12] [13] [14]:

$$\theta_0 = \log p_0, \quad \theta_1 = \log \frac{p_1}{p_0}, \quad \theta_2 = \log \frac{p_2}{p_0},$$

$$\theta_3 = \log \frac{p_3 p_0}{p_1 p_2}, \quad \theta_4 = \log \frac{p_4}{p_0}, \quad \theta_5 = \log \frac{p_5 p_0}{p_1 p_4}, \quad \theta_6 = \log \frac{p_6 p_0}{p_2 p_4},$$

$$\theta_4 = \theta_3^{123} = \log \frac{p_3^{123} p_1^2 p_1^2 p_1^3}{p_2^{12} p_2^{13} p_2^{23} p_0} = \log \frac{p_7 p_1 p_2 p_4}{p_3 p_5 p_6 p_0}$$

The inverse are:

$$p_0 = \exp(\theta_0), p_1 = \exp(\theta_0 + \theta_1), p_2 = \exp(\theta_0 + \theta_2), p_3 = \exp(\theta_0 + \theta_1 + \theta_2 + \theta_3),$$

$$p_4 = \exp(\theta_0 + \theta_4), p_5 = \exp(\theta_0 + \theta_1 + \theta_4 + \theta_5), p_6 = \exp(\theta_0 + \theta_2 + \theta_4 + \theta_6),$$

$$p_7 = \exp(\theta_0 + \theta_1 + \theta_2 + \theta_3 + \theta_4 + \theta_5 + \theta_6 + \theta_7).$$

Because of the nature of the exponential function, the 0 probability is $0 = e^{-\infty}$, so the minimum of the probability, p_i, will be some very small value, ε, rather than 0 to avoid singularity.

Example

Let $p = \{2,7,3,8,2,5,5,6\}$, then

$\theta = \{0.693,1.252,0.405,-0.271,0,-0.336,0.510,-0.462\}$.

Example

Let $\theta = \{0,0,0,0,0,0,0,2.302\}$, then $p = \{1,1,1,1,1,1,1,10\}$.

Example

Let $\theta = \{2.302,-0.223,-1.609,1.832,-0.510,-0.875,1.763,-1.581\}$, then

$p = \{10,8,2,10,6,2,7,3\}$.

6. θ-Transformation Is Complete

Because the θ-transformation is implemented by normal function, $FG = GF = I$, as long as there is no singular points in the transformation, any distribution function can be expanded. For example, in the last section, we require $p_i \geq \varepsilon$, which is a predefined small number.

7. ABM Is Complete

An ABM with n neurons has neural connections up to the n^{th} order. The invariant distribution is:

$$p(x) = e^H,$$

$$H = \theta_0 + \sum \theta_1^{i_1} x_{i_1} + \sum \theta_2^{i_1 i_2} x_{i_1} x_{i_2} + \sum \theta_3^{i_1 i_2 i_3} x_{i_1} x_{i_2} x_{i_3} + \cdots + \theta_n^{12\cdots n} x_1 x_2 \cdots x_n. \quad (17)$$

An ABM implements a θ-transformation [12] [13] [14] with:

$$F(y) = \log(y), G(y) = \exp(y). \quad (18)$$

Furthermore, the "connection matrix" element can be calculated as follows [6] [7] [8]:

$$\theta_m^{i_1 i_2 \cdots i_m} = \log \frac{p_m^{i_1 i_2 \cdots i_m} p_{m-2}^{i_1 \cdots i_{m-2}} \cdots p_{m-2}^{i_3 \cdots i_m} p_{m-4}^{\cdots}}{p_{m-1}^{i_1 \cdots i_{m-1}} \cdots p_{m-1}^{i_2 \cdots i_m} p_{m-3}^{i_1 \cdots i_{m-3}} \cdots} \quad (19)$$

The reverse problem is as follows: given an ABM, the invariant distribution can be calculated as follows [12] [13] [14]:

$$p_m^{i_1 i_2 \cdots i_m} = \exp\left(\theta_0 + \theta_1^{i_1} + \theta_1^{i_2} + \cdots + \theta_1^{i_m} + \theta_2^{i_1 i_2} + \theta_2^{i_1 i_3} + \cdots + \theta_2^{i_{m-1} i_m} + \cdots + \theta_m^{i_1 i_2 \cdots i_m}\right). \quad (20)$$

Therefore, an ABM can realize a θ-expansion, which in turn can approximate any distribution. ABM is complete [12] [13] [14].

8. Reduction of ANN from Higher Order to Lower Order

In this section, we show that we can reduce a higher order ANN to a lower order ANN by introducing more layers. We start with the base case, three neurons; then we go to the inductive step of the mathematical induction.

8.1. Third Order ABM

Assume that we have a first ANN with three neurons in one layer, $x=\{x_1,x_2,x_3\}$; the higher order distribution is:

$$p(x)=\mathrm{e}^{H},$$

$$H=\theta_0+\sum_{i_1=1}^{3}\theta_1^{i_1}x_{i_1}+\sum_{i_1<i_2,i_1=1}^{3}\theta_2^{i_1i_2}x_{i_1}x_{i_2}+\theta_3^{123}x_1x_2x_3 \tag{21}$$

There is only one third order term in the above distribution for 3 neurons.

We simulate the first network with a second ANN with two layers, $x=\{x_1,x_2,x_3\}$ and $y=\{y_1,y_2,y_3,y_4\}$. The transition from the first layer to the second layer is:

$$y_1=x_1,\ y_2=x_2,\ y_3=x_3,\ y_4=x_1*x_2.$$

Let $\{y_1,y_2,y_3\}$ be the same network as the first ANN without the higher order term. The neuron, y_4, has 4 additional connections, which will be defined as follows:

$$\theta_1^4=0,\ \theta_2^{34}=\theta_3^{123},\theta_2^{14}=\theta_2^{24}=0.$$

The second order distribution of $\{y_1,y_2,y_3,y_4\}$ is:

$$p(y)=\mathrm{e}^{H},$$

$$H=\theta_0+\sum_{i_1=1}^{4}\theta_1^{i_1}y_{i_1}+\sum_{i_1<i_2,i_1=1}^{4}\theta_2^{i_1i_2}y_{i_1}y_{i_2}.$$

There are only connections up to the second order. Separating y4, we have:

$$\sum_{i_1=1}^{4}\theta_1^{i_1}y_{i_1}=\sum_{i_1=1}^{3}\theta_1^{i_1}x_{i_1}+\theta_1^4y_4,$$

$$\sum_{i_1<i_2,i_1=1}^{4}\sum\theta_2^{i_1i_2}y_{i_1}y_{i_2}=\sum_{i_1<i_2,i_1=1}^{3}\theta_2^{i_1i_2}x_{i_1}x_{i_2}+\theta_2^{34}y_3y_4+\theta_2^{14}y_1y_4+\theta_2^{24}y_2y_4.$$

Using the conditions:

$$\theta_1^4=0,\ \theta_2^{34}=\theta_3^{123},\theta_2^{14}=\theta_2^{24}=0,$$

and substituting y-neurons by x-neurons:

$$\theta_1^4y_4=0,\ \theta_2^{34}y_3y_4=\theta_3^{123}x_1x_2x_3,\ \theta_2^{14}y_1y_4=0,\ \theta_2^{24}y_2y_4=0,$$

we have:

$$H=\theta_0+\sum_{i_1=1}^{3}\theta_1^{i_1}x_{i_1}+\sum_{i_1<i_2,i_1=1}^{3}\theta_2^{i_1i_2}x_{i_1}x_{i_2}+\theta_3^{123}x_1x_2x_3.$$

So in this base case, we have proven that the invariant distribution of a higher order ANN can be precisely duplicated by the invariant distribution of a lower order ANN with multiple layers.

8.2. Higher Order ABM

We now show how to reduce the n[th] order term to the $(n-1)$ order term.

Consider a first ANN has neural connections up to the n[th] order. There is only one n[th]-order term:

$$\theta_n^{12\cdots n} x_1 x_2 \cdots x_n .$$

The rest of the connections are at most $(n-1)$ order.

We simulate the first network with a second ANN with additional layers, which has an additional layer $y = \{y_1, y_2, y_3, \cdots, y_n, y_{n+1}\}$ added on the top of the first ANN. This second ANN has neural connections up to the $(n-1)$ order. The transitions from the first layer to the second layer are defined as:

$$y_1 = x_1, y_2 = x_2, \cdots, y_n = x_n, y_{n+1} = x_1 * x_2.$$

Based on the same approach in the last section, we can reduce this term from the order of n to the order of $n-1$. The neuron, y_{n+1}, has many additional connections. With the exception of the following term:

$$\theta_{n-1}^{34\cdots n+1} = \theta_n^{12\cdots n},$$

all other connections will be defined as 0:

$$\theta_1^{n+1} = 0, \theta_2^{1n+1} = \theta_2^{2n+1} = \cdots = 0, \cdots, \theta_3^{12n+1} = \theta_3^{13n+1} = \cdots = 0, \cdots$$

The $(n-1)$ order distribution of $\{y_1, y_2, y_3, \cdots, y_n, y_{n+1}\}$ is:

$$p(y) = \mathrm{e}^H,$$

$$H = \theta_0 + \sum_{i_1=1}^{n+1} \theta_1^{i_1} y_{i_1} + \sum_{i_1<i_2, i_1=1}^{n+1} \theta_2^{i_1 i_2} y_{i_1} y_{i_2} + \cdots + \sum_{i_1<i_2, i_1=1}^{n+1} \theta_{n-1}^{i_1 i_2 \cdots} y_{i_1} y_{i_2} + \cdots$$

Separating y_{n+1} from the other first order term, substituting y-neurons by x-neurons, and using the condition $\theta_1^{n+1} = 0$, we have:

$$\sum_{i_1=1}^{n+1} \theta_1^{i_1} y_{i_1} = \sum_{i_1=1}^{n} \theta_1^{i_1} x_{i_1} + \theta_1^{n+1} y_{n+1} = \sum_{i_1=1}^{n} \theta_1^{i_1} x_{i_1}$$

Separating y_{n+1} from the other second order term, substituting y-neurons by x-neurons, and using the conditions, $\theta_2^{1n+1} = \theta_2^{2n+1} = \cdots = 0$, we have:

$$\sum_{i_1<i_2<, i_1=1}^{n+1} \theta_2^{i_1 i_2} y_{i_1} y_{i_2} = \sum_{i_1<i_2, i_1=1}^{n} \theta_2^{i_1 i_2} y_{i_1} y_{i_2} + \theta_2^{1n+1} y_1 y_{n+1} + \theta_2^{2n+1} y_2 y_{n+1} + \cdots$$

$$= \sum_{i_1<i_2<, i_1=1}^{n} \theta_2^{i_1 i_2} y_{i_1} y_{i_2}$$

$$\cdots$$

Separating y_{n+1} from the $(n-1)$ order term, substituting y-neurons by x-neurons, and using the condition:

$$\theta_{n-1}^{34\cdots n+1} = \theta_n^{12\cdots n}, \ \theta_{n-1}^{12\cdots n+1} = \theta_{n-1}^{13\cdots n+1} = \cdots = 0,$$

we have:

$$\sum_{i_1<i_2<,i_1=1}^{n+1}\theta_{n-1}^{i_1i_2\cdots}y_{i_1}y_{i_2}\cdots=\sum_{i_1<i_2<,i_1=1}^{n}\theta_{n-1}^{i_1i_2\cdots}x_{i_1}x_{i_2}\cdots+\theta_n^{12\cdots n}x_1x_2\cdots x_n$$

Combine all of the above terms, we have:

$$p(x)=\mathrm{e}^H$$

$$H=\theta_0+\sum\theta_1^{i_1}x_{i_1}+\sum\theta_2^{i_1i_2}x_{i_1}x_{i_2}+\sum\theta_3^{i_1i_2i_3}x_{i_1}x_{i_2}x_{i_3}+\cdots+\theta_n^{12\cdots n}x_1x_2\cdots x_n \quad (22)$$

Now, we have proven that the invariant distribution of an n^{th} order ANN can be precisely duplicated by the invariant distribution of a $(n-1)$ order ANN with multiple layers.

We have shown both reduction from $n=3$ to $n=2$ and reduction from arbitrary n to $n-1$.

9. Completeness of Deep Neural Networks

Given an unknown function, one can find a θ-expansion to convert it from the x-coordinate system to the θ-coordinate system using the following generating function:

$$F(y)=\log(y),\ G(y)=\exp(y).$$

This expansion can be represented by an ABM.

ABM is a higher order ANN which can be expanded into multiple layers with only second order terms. Therefore, a Deep Neural Network can realize a θ-expansion, which in turn can approximate any distribution. Therefore, the Deep Neural Network is complete.

We stress that we merely provide the proof for the existence of a solution for simulating any mapping from A to B. We did not seek for an effective implementation of a Deep Neural Network from supervised training data, (A, B). We have merely proved that this is possible.

10. Conclusions

Hornik, Stinchcombe, & White have shown that the multilayer feed forward networks with enough hidden layers are universal approximators. Roux & Bengio have shown the same. In this paper, we provide yet another proof. The advantage of this proof is that it will lead to several new learning algorithms, which will be presented in our future publications.

In conclusion, we have studied the completeness problem of the Deep Neural Networks. We have reviewed the Attrasoft Boltzmann Machine (ABM). An ABM with n neurons has neural connections up to the n^{th} order. We have reviewed the θ-transformation and shown that the θ-transformation is complete, *i.e.* any function can be expanded by θ-transformation. We have further shown that the invariant distribution of the ABM is a θ-transformation; therefore, an ABM can simulate any distribution. An ABM with n neurons has neural connections up to the n^{th} order. We have shown how to convert an ABM to a Deep

Neural Network. Finally, by establishing the equivalence between an ABM and the Deep Neural Network, we have proven that the Deep Neural Network is complete. In other words, the Deep Neural Network can simulate any mapping from A to B.

Acknowledgements

I would like to thank Gina Porter for proof reading this paper.

References

[1] Hinton, G.E., Osindero, S. and Teh, Y. (2006) A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, **18**, 1527-1554. https://doi.org/10.1162/neco.2006.18.7.1527

[2] Bengio, Y. (2009) Learning Deep Architectures for AI (PDF). *Foundations and Trends in Machine Learning*, **2**, 1-127. https://doi.org/10.1561/2200000006

[3] Amari, S., Kurata, K. and Nagaoka, H. (1992) Information Geometry of Boltzmann Machine. *IEEE Transactions on Neural Networks*, **3**, 260-271. https://doi.org/10.1109/72.125867

[4] Byrne, W. (1992) Alternating Minimization and Boltzmann Machine Learning. *IEEE Transactions on Neural Networks*, **3**, 612-620. https://doi.org/10.1109/72.143375

[5] Coursera (2017). https://www.coursera.org/

[6] TensorFlow (2017). https://www.tensorflow.org/

[7] Torch (2017). http://torch.ch/

[8] Theano (2017). http://deeplearning.net/software/theano/introduction.html

[9] Hornik, K., Stinchcombe, M. and White, H. (1989) Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks*, **2**, 359-366. https://doi.org/10.1016/0893-6080(89)90020-8

[10] Le Roux, N. and Bengio, Y. (2008) Representational Power of Restricted Boltzmann Machines and Deep Belief Networks. *Neural Computation,* **20**, 1631-1649. https://doi.org/10.1162/neco.2008.04-07-510

[11] Feller, W. (1968) An Introduction to Probability Theory and Its Application. John Wiley and Sons, New York.

[12] Liu, Y. (1993) Image Compression Using Boltzmann Machines. *Proceedings of SPIE*, **2032**, 103-117. https://doi.org/10.1117/12.162027

[13] Liu, Y. (1995) Boltzmann Machine for Image Block Coding. *Proceedings of SPIE*, **2424**, 434-447. https://doi.org/10.1117/12.205245

[14] Liu, Y. (1997) Character and Image Recognition and Image Retrieval Using the Boltzmann Machine. *Proceedings of SPIE*, **3077**, 706-715. https://doi.org/10.1117/12.271533

[15] Liu, Y. (2002) US Patent No. 7, 773, 800. http://www.google.com/patents/US7773800