American International Journal of Computer Science and Information Technology
Volume.7, Number 1;January-March, 2022;
ISSN: 2837-388X| Impact Factor:
https://zapjournals.com/Journals/index.php/aijcsit
Published By: Zendo Academic Publishing

"UNDERSTANDING LINEAR NEURONS: FUNDAMENTALS AND LEARNING TECHNIQUES"

Jessica Lauren Williams[1]

Article Info

Keywords: Neural Networks,Deep Learning, Supervised Learning, Tensor Flow,Training Algorithms

Abstract

Neural networks and deep learning have emerged as leading solutions for a myriad of supervised learning challenges. In 2006, Hinton, Osindero, and Teh introduced the concept of "deep" neural networks, a methodology that initiates with training a basic supervised model, subsequently augmenting it with additional layers and exclusively training the parameters of the new layer. This iterative process continues until a deep network is formed. Over time, the necessity for training one layer at a time has been transcended.

Contemporary Deep Neural Networks adopt a holistic approach, training all layers concurrently. Exemplary implementations include Tensor Flow, Torch, and Theano. Google's Tensor Flow, an open-source dataflow programming library, serves as a versatile tool for tasks spanning symbolic mathematics and machine learning applications, including neural networks. It is a staple for both research and production endeavors at Google. Torch, an open-source machine learning library and scientific computing framework, has established itself as an invaluable resource. Meanwhile, Theano, a numerical computation library for Python, contributes significantly to the numerical computations domain.

This unified training methodology across multiple layers imparts distinct advantages to neural networks, distinguishing them from other learning algorithms.

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.

[1] Department of Engineering Technology, Savannah State University, Savannah, Georgia 31404, USA

After Hinton‟s initial attempt of training one layer at a time, Deep Neural Networks train all layers together. Examples include Tensor Flow [30], Torch [31], and Theano [32]. Google‟s Tensor Flow 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.[6] 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 [33] have shown that the multilayer feedforward networks with enough hidden layers are universal approximators. Roux & Bengio [34] have shown the same, Restricted Boltzmann machines are universal approximators of discrete distributions. Hornik, Stinchcombe, & White [33] 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 feed forward 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 [34] proved that adding hidden units yield a strictly improved modeling power, and RBMs are universal approximators of discrete distributions.

In our earlier paper [15], we provide yet another proof. The advantage of this proof is that it will lead to multiple new learning algorithms. In our approach, Deep Neural Networks implement an expansion and this expansion is complete. In this paper, we will present a new algorithm based our earlier paper [15]. In addition to neural network algorithms, there are numerous learning algorithms. We select a few such algorithms below.

Principal Component Analysis [16-17] is a statistical procedure that uses an orthogonal transformation to convert a set of vectors into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables.

Sparse coding [18-19] minimizes the objective:

$L_{sc} = ||WH-X||_2^2 + \lambda ||H||_1$

Where, W is a matrix of transformation, H is a matrix of inputs, and X is a matrix of the outputs. λ implements a trade of between sparsity and reconstruction. Auto encoders [20-25] minimizes the objective: $L_{ae} = ||W \sigma (W^T X) - X||_2^2$

Where σ is some neural network functions. Note that L_{sc} looks almost like L_{ae} once we set $H=\sigma(W^T X)$. The difference is that 1) auto encoders do not encourage sparsity in their general form; 2) an auto encoder uses a model for finding the codes, while sparse coding does so by means of optimization.

K-means clustering [26-29] is a method of vector quantization which is popular for cluster analysis in data mining. K-means clustering aims to partition n observations into k clusters. Each observation belongs to the

cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into k clusters.

If we limit the learning architecture to one layer, all of these algorithms have some advantages for some applications. The deep learning architectures currently provide the best solutions to many supervised learning problems; in other words, two layers are better than one layer. 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. In this approach of completing one layer before moving into the next layer, one can argue that each layer can be any learning algorithm and the neural network is one of many candidates in each layer.

The advantage of Deep Neural Network is to train all layers together; for example, TensorFlow [30], Torch [31], and Theano [32]. 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 with single training of multiple layers gives advantages to the neural network over other learning algorithms.

If one accepts the assumption that deep learning currently provides the best solutions to many supervised learning problems, and if one further accepts that a single training of multiple layers is better than multiple training of single layers, then neural networks emerge as the only candidate. Therefore, it is important to study the neural network algorithm thoroughly. In this paper, we introduce the concepts of Linear neurons, and new learning algorithms based on Linear neurons, with an explanation of the reasons behind these algorithms.

This approach advances the current state of understanding of neural networks by stating that neural network is a part of a larger and complete expansion, where completeness means any function can be expanded. Based on the above discussion regarding the importance of the neural network, this different perception matters; it has both practical and theoretical significance. An alternative to the direction of "deep", higher order is another direction. These two directions are equivalent [15]. In addition, both deep and higher order can be combined. In this paper, we review θtransformation and will introduce a new family of learning algorithms characterized by θ-transformation.

In section 2, we briefly review how to use probability distributions in a Supervised Learning Problem. 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 [4-8] of (A, B): p (a, b) , a ϵA, b ϵ B. If an input is a ϵA and an output is b ϵ B, then the probability p (a, b) will be close to 1. One can find a Markov chain [9] 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 [4-8] are briefly reviewed. Our discussion concentrates on the distribution space of the Boltzmann machine rather than the neural aspects. 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 [11-14]. 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. A natural question is whether such an approximation is possible. The answer is that this approximation is not yet a good approximation.

In section 4, we review the ABM (Attrasoft Boltzmann Machine) [10] 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 [10]. By adding more terms in the

invariant distribution compared to the second order Boltzmann Machine, ABM is significantly more powerful to simulate an unknown function. Unlike the Boltzmann Machine, ABM‟s emphasize higher order connections rather than lower order connections. The Boltzmann Machine and the ABM are at the opposite end of the neuron orders.

In section 5, we review θ-transformation [11-14].

In section 6, we review the completeness of the θ-transformation [11-14]. The θ-transformation is complete; i.e. given a function, one can find a θ-transformation by converting 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 [11-14], 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.

In section 8, we show that the ABM algorithm is derived from the θ-transformation. We call the neurons in the ABM algorithm the exponential neurons, because of its exponential generating function. We introduce a new algorithm, and we will call the neurons in the new algorithm, linear neurons, because of its generating functions. The new algorithm uses summation in expansion, while the ABM algorithm uses multiplication in expansion. Therefore, the new algorithm is more stable.

In section 9, we introduce a simple example to test the algorithm.

2. Basic Approach

The basic supervised learning [2] problem is: given a training set {A, B}, where A = {a_1, a_2, … } and B = {b_1, b_2, …}, find a mapping from A to B. It turns out that if we can reduce this from a discrete problem to a continuous problem, it is very helpful. The first step is to convert this problem to a probability [4-8]: p = p (a, b), a ϵ A, b ϵ 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 [9].

This invariant distribution can be used to simulate an unknown function.

3. Boltzmann Machine

A Boltzmann machine [4-8] 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 Boltzmann distribution. The collection of the neuron states: $x = (x_1, x_2, ..., x_n)$of a Boltzmann machine is called a configuration. The configuration transition is mathematically described by a Markov chain with 2^n configurations x ϵ X, where X is the set of all points, (x_1, x_2, ..., x_n). When all of the configurations are connected, it forms a Markov chain. A Markov chain has an invariant distribution [9]. Whatever initial configuration that a Boltzmann starts from, the probability distribution converges over time to the invariant distribution, p(x). The configuration x ε X appears with a relative frequency of p(x) over a long period of time.

A Boltzmann machine [4-8] defines a Markov chain. Each configuration of a Boltzmann machine is a state of the Markov chain. A 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 ε F, one can find a Boltzmann machine such that the equilibrium distribution of this

Boltzmann machine realizes, as faithfully as possible, the unknown function [4-8]. Therefore, the unknown, f, is encoded into a specification of a Boltzmann machine, t ϵ T. We call the mapping from F to T as a Boltzmann Machine Transformation: F □ T [11-14].

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 so 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 [4-8] is:

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

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

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

By rearranging the above distribution, we have: $p(x) = ec - \sum T_i x_i + \sum_{i<j} M_{ij}\, x_i x_j$

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

$$p(x) = ec - \sum T_i x_i + \sum_{i<j} M_{ij}\, x_i x_j + \sum_{i<j<k} M_{ijk}\, x_i x_j x_k.$$

An ABM [11-14] 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 [10]. By adding additional higher order terms to the invariant distribution, ABM is significantly more powerful to simulate an unknown function.

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

$p(x) = e^H$,

$H = \theta 0 + \sum \theta 1 i1\, xi1 + \sum \theta i2 i2\, xi1\, xi2$

$+ \sum \theta i1 i2 i33 xi1\, xi2\, xi3 + \ldots + \theta 12n\, ..nx1\, x2\, ...xn.$

ABM is significantly more powerful to simulate 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 in the last section, ABM implements a transformation, F_B□ T. We will show that this ABM transformation is complete so that given any function f ϵ F, we can find an ABM, t ϵ T, such that the equilibrium distribution of this ABM realizes precisely the unknown function.

5. Θ - Transformation

5.1 Basic Notations

We first introduce some notations used in this paper [11-14]. There are two different types of coordinate systems: the x-coordinate system and the θ-coordinate system [11-14]. Each of these two coordinate systems has two representations, x-representation and θ-representation. An N-dimensional vector, p, is:

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

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
- $\{ p_m^{i1i2\,..im} \}$ in the θ-representation.

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

- { θi } in the x-representation, and
- { θmi1i2 ..im } in the θ-representation.

The reason for the 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 $\{ p_m^{i1i2..im} \}$, and those between $\{ \theta_i \}$ and $\{ \theta_m^{i1i2..im} \}$, are similar. Because of this similarity, only the transformation between $\{p_i\}$ and $\{p_m^{i1i2..im}\}$ will be introduced. Let $N = 2^n$ be the number of neurons. An N-dimensional vector, p, is: $p = (p_0 , p_1 , ... p_{N-1})$.

Consider p_x, because $x \in \{ 0, 1, ... N - 1 = 2^n - 1 \}$ is the position inside a distribution, then x can be rewritten in the binary form: $x = x_n 2^{n-1} + ... + x_2 2^1 + x_1 2^0$.

Some of the coefficients, x_i, might be zero. In dropping those coefficients which are zero, we write: $x = x\,i1\,x\,i2\,...x\,im = 2im-1 + ... +2i2-1 + 2i1-1$.

This generates the following transformation:

$pim1i2...im = px = p2im-1 + ... +\_2i2-1 + 2i1-1$,

where

$1 \leq i_1 < i_2 <... < i_m \leq n$.

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

$\{p_0 , \ p^1{}_1 , \ p_1{}^2 , \ p_1{}^3,..., p^{12}{}_2 , p^{13}{}_2 , p^{23}{}_2 , ..., p^{123}{}_3 ,...\}$

The 0-th order term is $p0$, the first order terms are: $p11$, $p12$, $p13$,..., … The first few terms in the transformation between $\{ p_i \}$ and $\{ p_m^{i1i2..im} \}$ are:

$p_0 = p_0 , \quad p^1{}_1 = p_1 , \quad p_1{}^2 = p_2 ,$

$p^{12}{}_2 = p_3 , \quad p_1{}^3 = p_4 , \quad p^{13}{}_2 = p_5 ,$

$p^{23}{}_2 = p_6 , \quad p^{123}{}_3 = p_7 \quad p_1{}^4 = p_8 , ...$

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:

{ p0, p1, p2, p3, p4, p5, p6, p7},

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

$\{p0 , \ p11 , \ p12 , \ p13 , p122 , p132 , p232 , p1233 \}$.

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 xrepresentation and the second vector $\{p0 , \ p11 , \ p12 , \ p13, p122 , p132 , p232 , p1233 \}$ 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(θ). 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 xrepresentation q(x) to the θ-representation q(θ), and then q(θ) is transformed back to q'(x), then these two vectors are equal: q'(x) = q(x). The θ-transformation uses a function F, called a generating function. The function F is required to have the inverse:

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

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

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

Or $p(x) = (p0 ; p11 , ..., p1n ; p122 , ... , p2n-1,n ; p1233$

$, ... ,p12n ..n)$.

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(θ) in the θcoordinate:

$p(\square) = (\square 0 ; \square 11 , ..., \square 1n ; \square 122 , ... , \square n2\text{-}1,n ; \square 1233 , ... , \square 12n ..n),$

Or $p(\square) = (\square_0 , \square_1 , ... \square_{N-1}).$

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

$F[p(x)] = \square 0 + \square\square 1i1\ xi1 + \square\ \square i21i2\ xi1\ xi2$

$+ \square\square i1i2i33\ \ xi1\ xi2\ xi3 + ... + \square 12n\ ..nx1\ x2\ ...xn$ Where

$1 \square i_1 < i_2 <... < i_m \square n.$

Prior to the transformation, p(x) is the x-representation of p in the x-coordinate; after transformation, F[p(x)] is a θrepresentation 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:

$X_0 = X_0 = 1, \quad X^1{}_1 = X_1 = x_1 , \quad X_1{}^2 = X_2 = x_2 \quad X^{12}{}_2 = X_3 = x_1 x_2 ,$

$X_1{}^3 = X_4 = x_3 , \quad X^{13}{}_2 = X_5 = x_1 x_3 , \quad X^{23}{}_2 = X_6 = x_2 x_3 , X^{123}{}_3 = X_7 = x_1 x_2 x_3 ,$

$X_1{}^4 = X_8 = x_1 x_2 x_3 x_4 ...$

then the vector can be written as:

$F[p(x)] = \square\square_J X_J .$

By using the assumption GF = I, we have:

$p(x) = G\{\square\square_J X_J\},$

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(θ), in the θ-coordinate system, is called θ-transformation [11-14].

The θ-transformation is determined by [11-14]:

$\square im1i2...im = F[\ pim1i2...im\ ]+ F[\ pim1...\text{-}2im\text{-}2\ ]$

$+ ... + F[\ pim3...\text{-}2im\ ] + F[\ p...m\text{-}4\ ] + ...$

$- \quad F[\ pim1...\text{-}1im\text{-}1\ ] - ... F[\ pim2...\text{-}1im\ ]$

$- \quad F[\ pim1...\text{-}3im\text{-}3\ ] - ....$

The inverse of the θ-transformation [11-14] is:

$pim1i2...im = G(\square 0 + \square 1i1 + \square 1i2 + ...+ \square 1im$

$+ \square i21i2 + \square i21i3 + ... + \square i2m\text{-}1im$

$+ ...+ \square im1i2...im).$

5.3 An Example

Let an ANN have 3 neurons:

(x1, x2, x3) and let a distribution be:

{ p0, p1, p2, p3, p4, p5, p6, p7}.

Assume that the generating functions are:

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

By θ-transformation, the components are [11-14]: p^1

, $\square_2 = \log p^2 ,$ — —

$\square_1 = \log p_0 , \ \square_1 = \log$

$p^3\ p^0$, $\frac{p0}{}$ $\square_4 = \frac{p0}{}$ log p^4 , $\square_5 =$ log $p^5 p^0$, $\square_6 =$ log $p^6 p^0$,

$\square_3 =$ log $\frac{p1\ p2}{p123} \frac{p0}{p1\ p2\ p3}$ $\frac{p1\ p4}{}$ $\frac{p2\ p4}{}$

$\square 7 = \square 1233 =$ log $p123\ p131 p231\ p1 =$ log $\frac{p\ p\ p\ p}{2\ 2\ 0} \frac{}{p73\ p51\ p26\ p40}$.

The inverse are:

$p_0 \square \exp(\square_0)$, $p_1 \square \exp(\square_0 \square\square_1)$, $p_2 \square \exp(\square_0 \square\square_2)$, $p_3 \square \exp(\square_0 \square\square_1 \square\square_2 \square\square_3)$,

$p_4 \square \exp(\square_0 \square\square_4)$, $p_5 \square \exp(\square_0 \square\square_1 \square\square_4 \square\square_5)$, $p_6 \square \exp(\square_0 \square\square_2 \square\square_4 \square\square_6)$, p_7

$\square \exp(\square_0 \square\square_1 \square\square_2 \square\square_3 \square\square_4 \square\square_5 \square\square_6 \square\square_7)$.

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 θ = {0.693, 1.252, 0.405, -0.271, 0, -0.336, 0.510, -0.462}.

Example

Let θ = {0, 0, 0, 0, 0, 0, 0, 2.302}, then p = {1, 1, 1, 1, 1, 1, 1, 10}.

Example

Let θ = {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 a 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 $p_i >= \varepsilon$, 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:

$p(x) = e^H$,

$H = \square 0 + \square\square 1i1\ xi1 + \square\ \square i2 1i2\ xi1\ xi2$

$+ \square\square i1i2i33 xi1\ xi2\ xi3 + ... + \square 12n\ ..nx1\ x2\ ...xn.$

An ABM implements a θ-transformation [11-14] with:

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

Furthermore, the "connection matrix" element can be calculated as follows [11-14]:

$pi1i2...im\ pi1...im\text{-}2\ ...\ pi3...im\ p...$

$\square i1i2...im =$ log $mpi1...im\text{-}1\ ...m\text{-}2pim2\text{-}...1im\ \overline{pmim1\text{-}...\text{-}23im\text{-}3\ ...m\text{-}4}\ .\ m\ m\text{-}1$

The reverse problem is as follows: given an ABM, the invariant distribution can be calculated as follows [11-14]:

$pim1i2...im = \exp(\square 0 + \square 1i1 + \square 1i2 + ... + \square 1i2$

$+ \square i21i2 + \square i21i3 + ... + \square i2m\text{-}1im + ... + \square im1i2...im).$

Therefore, an ABM can realize a θ-expansion, which in turn can approximate any distribution. ABM is complete [11-14]. Write the above equation:

$pim1i2...im = \exp(\square 0)\exp(\square 1i1)\exp(\square 1i2)\ ...\exp(\square 1i2)$

$\exp(\square i21i2)\exp(\square i21i3)\ ...\ \exp(\square i2m\text{-}1im)\ ...\exp(\square im1i2...im).$

We call the neurons in the ABM algorithm the exponential neurons, because of its exponential generating function. The ABM algorithm uses multiplication expansion, which raises the question of stability. Therefore, we expect to improve this algorithm.

8. A New Algoritm

If we can convert the multiplication expansion to addition expansion, then the performance will be more stable. Let :

G = X, F = X, from section

5, we have:

$\square im1i2...im = pim1i2...im \square pim1...-2im-2 \square ... \square pim3...-2im \square p...m-4 \square pim1...-1im-1 \square ...pim2...-1im \square pim1...-3im-3 \square ... pim1i2...im = \square 0 + \square 1i1 + \square 1i2 + ... + \square 1im + \square i21i2 + \square i21i3 + ... + \square i2m-1im + ... + \square im1i2...im$.

We call the neurons in the new algorithm as linear neurons, because of its generating functions. The new algorithm uses summation in expansion.

For simplicity, we will continue to call the cell "neurons". Because G = F = I, we call these neurons identify neurons. Example: let an ANN have 3 neurons, (x1, x2, x3) and let a distribution be:

{ p0, p1, p2, p3, p4, p5, p6, p7},Then, $p_0 \square (\square_0)$, $p_1 \square (\square_0 \square\square_1)$, $p_2 \square (\square_0 \square\square_2)$, $p_3 \square (\square_0 \square\square_1 \square\square_2 \square\square_3)$, $p_4 \square (\square_0 \square\square_4)$, $p_5 \square (\square_0 \square\square_1 \square\square_4 \square\square_5)$, $p_6 \square (\square_0 \square\square_2 \square\square_4 \square\square_6)$, $p_7 \square (\square_0 \square\square_1 \square\square_2 \square\square_3 \square\square_4 \square\square_5 \square\square_6 \square\square_7)$.

This is the starting point for the new algorithm. When the expansion is complete, it has the advantage of accuracy. When the expansion uses addition, it has the advantage of stability. As we will show below, it also has a third advantage of fast training (low time complexity).

The L_1 –distance between two configurations is:

$d(x', x) = |x'_1 - x_1| + |x'_2 - x_2| + \ldots$

For example, d (111, 111) = 0; d (111, 110) = 1.

We now introduce a new algorithm, the Linear neuron learning algorithm, which can be summarized into a single formula:

$\square im1i2...im \square 2^{(D \square d(xim1i2...im, x))}$, *if* $0 \square d(xim1i2...im, x) \square D$

$\square im1i2...im \square 0$, *if* $d(xim1i2...im, x) \square D$

Where $d(x^i{}_m{}^{1i2...im}, x)$ is the distance between a neuron configuration, x, and a training neuron configuration, $x^i{}_m{}^{1i2...im}$, and D is called a connection radius. Beyond this radius, all connections are 0.

The linear neurons learning algorithm is: Step

1. The First Assignment (d=0)

The first step is to assign the first connection matrix element for training vector, $x = x^i{}_m{}^{1i2...im}$. We will assign: $\theta x = \square im1i2...im = 2D$, while D is the radius of the connection space.

Step 2. The Rest of the Assignment

The next step is to assign the rest of the weight:

$\square im1i2...im \square 2^{(D \square d(xim1i2...im, x))}$, *if* $0 \square d(xim1i2...im, x) \square D$

$\square im1i2...im \square 0$, *if* $d(xim1i2...im, x) \square D$

Step 3. Modification

The algorithm uses bit "1" to represent a pattern or a class; so, the input vectors or the output vectors cannot be all 0"s; otherwise, these coefficients are 0. Step 4. Retraining

Repeat the last three steps for all training patterns; if there is an overlap, take the maximum values:

$\square im1i2...im$ (t+1) = max {$\square im1i2...im$ (t), $\square im1i2...im$ }.

If we set D = 2, then the possible elements are: 2^{D-d} = 4, 2, 1, and 0. There is one term for d = 0; there are no more than n terms for d = 1; and there are no more than n(n-1)/2 terms for d = 2. Assume there are m training vectors, the time complexity for training is:

$T = O(m n^2)$.

In general,

$T = O(m n^D)$.

The advantage of this new algorithm is completeness (accuracy), stability, and fast speed.

9. An Example

In this section, we will first use the linear neuron algorithm; then we will use the square and power neuron algorithms. The example is to identify simple digits in Figure 1 [1, 2, 13]. Each digit is converted into 7 bits: 0, 1, …, 6. Figure 2 shows the bit location.

Figure 1. An example.

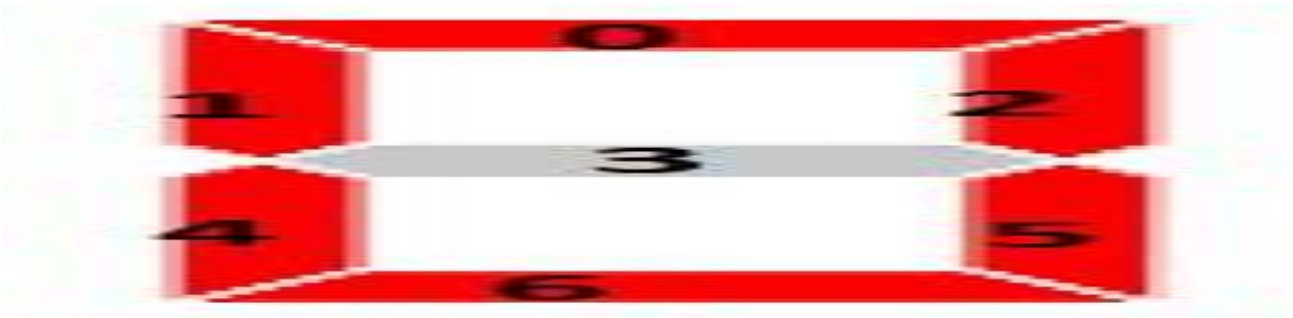

Figure 2. Input bit assignments.

The 10 input vectors have 7 bits:

I0 = (1, 1, 1, 0, 1, 1, 1),

I1 = (0, 0, 1, 0, 0, 1, 0),

I2 = (1, 0, 1, 0, 1, 0, 1),

The 10 output vectors have 10 bits:

O0 = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0),

O1 = (0, 1, 0, 0, 0, 0, 0, 0, 0, 0),

O2 = (0, 0, 1, 0, 0, 0, 0, 0, 0, 0),

The 10 training vectors have 17 bits:

T0 = (I0, O0) = ((1, 1, 1, 0, 1, 1, 1), (1, 0, 0, 0, 0, 0, 0, 0, 0, 0)), T1 = (I1, O1) = ((0, 0, 1, 0, 0, 1, 0), (0, 1, 0, 0, 0, 0, 0, 0, 0, 0)),

…

We will set the radius: D = 2; the possible elements are: 2^{D-d} = 4, 2, 1, and 0. We will work out a few examples. First, we rewrite T1 as:

x_3^{259} = ((0, 0, 1, 0, 0, 1, 0), (0, 1, 0, 0, 0, 0, 0, 0, 0, 0)).

The first connection element (0-distance) is: $\square_3^{259}$ □ 4. There are two coefficients for d = 1: $\square^{59}_2$ □ $\square^{29}_2$ □ 2. T1 generates

3 coefficients. $\square^{25}{}_2 \square$ 0, due to step 3 of the algorithm; in this case, the output vector is all 0"s.
Next, we rewrite T7 as:
$x^1{}_4{}^{,2,5,14}$ = ((1, 0, 1, 0, 0, 1, 0), (0, 0, 0, 0, 0, 0, 0, 1, 0, 0)).
The first connection element (0-distance) is: $\square^1{}_4{}^{,2,5,14} \square$ 4. There are 3 coefficients for d=1: $\square^1{}_3{}^{,2,14} \square \square^1{}_3{}^{,5,14} \square \square_3{}^{2,5,14} \square$ 2.
There are 3 coefficients for d = 2: $\square^1{}_2{}^{,14} \square \square^2{}_2{}^{,14} \square \square^5{}_2{}^{,14} \square$1. T7 generates 7 coefficients.
Last, we rewrite T4 as:
$x^1{}_5{}^{,2,3,5,11}$ = ((0, 1, 1, 1, 0, 1, 0), (0, 0, 0, 0, 1, 0, 0, 0, 0, 0)),
the first connection element (0-distance) is: $\square^1{}_5{}^{,2,3,5,11} \square$4 . There are 4 coefficients for d=1: $\square$14,2,3,11 $\square \square$14,2,5,11 $\square \square$14,3,5,11 $\square \square$24,3,5,11 $\square$ 2. There are 6 coefficients for d = 2: $\square$13,2,11 $\square \square$12,3,11 $\square$... $\square$1. T4 generates 11 coefficients.
After training the linear neuron algorithm with {T0, T1, …, T9}, all of the connection coefficients, $\square^{i}_{m}{}^{1i2...im}$, are calculated. For example, the probability is $p_3{}^{259}$, if the input is „1" and the output is in class 1; the probability is $p_3{}^{258}$, if the input is „1" and the output is in class 0; the probability is $p_3{}^{2,5,10}$, if the input is „1" and the output is in class 2; …

Section 7 provides the formula to calculate the probability of each (input, output) pair:
$pim1i2...im = \square 0 + \square 1i1 + \square 1i2 +$
$... + \square 1im$
$+ \square i21i2 + \square i21i3 + ... + \square i2m\text{-}1im$
$+ ... + \square im1i2...im$.
For example, $p3259 = \square 0 + \square 12 + \square 15 + \square 19 +$
$\square 225 + \square 292 + \square 592 + \square 3259$.
The character recognition results are given in Figure. 3, where the first column is input, then the next 10 columns are output. The output probability is not normalized in Figure 3. The relative probability for (input = 0, output = 0) is 31; those for (input = 0, output = 1) are 8; those for (input = 0, output = 2) are 6; … . So if the input is digit 0, the output is identified as 0. In this problem, the output is a single identification, so the largest weight determines the digit classification. In each case, all input digits are classified correctly.

Input	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	P_9
0	31	8	6	6	5	6	7	13	8	7
1	0	8	0	0	1	0	0	4	0	0
2	1	2	24	6	1	1	1	4	1	1
3	1	8	6	24	5	6	1	13	1	7
4	0	8	0	1	18	1	0	4	0	1
5	1	2	1	6	5	24	7	4	1	7
6	7	2	6	6	5	24	31	4	8	7
7	0	8	0	1	1	0	0	13	0	0
8	**31**	**8**	**24**	**24**	**18**	**24**	**31**	**13**	**39**	**31**
9	7	8	6	24	18	24	7	13	8	31

Figure 3. The results from the new linear neuron algorithm without normalization

10. Conclusion

In conclusion, we have introduced a new algorithm. 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 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 observed that θ-transformation defines a whole family of neurons. The θ-neurons generally emphasize higher order neurons rather than lower order neurons. The ABM uses exponential neurons. In this paper, we use linear neurons. We discussed that the ABM algorithm is only the first algorithm in a family of new algorithms based on the θ-transformation. We introduced the simplest algorithm in this family. We also discussed the advantages of this algorithm: accuracy, stability, and low time complexity.

Reference

Hinton, G. E., Osindero, S. and Teh, Y., "A fast learning algorithm for deep belief nets," Neural Computation 18, pp 1527-1554.

Edgar Sanches-Sinencio and Clifford Lau, Editors, Artificial Neural Networks, IEEE Press, 1992.

Ying Liu and Shaohui Wang, Completeness Problem of the Deep Neural Networks, submitted.

S. Amari, K. Kurata, and H Nagaoka, "Information geometry of Boltzmann machine," IEEE Trans., Neural Network, Vol. 3, No. 2, pp. 260 - 271.

Cihan H. Dagli, Editor, "Intelligent Engineering Systems Through Artificial Neural networks," Vol. 5, ASME Press, pp. 757 - 850, 1995.

William Byrne, "Alternating Minimization and Boltzmann Machine Learning," IEEE Trans., Neural Network, Vol. 3, No. 4, pp. 612 - 620.

N. H. Anderson and D. M. Titterington, " Beyonf the Binary Boltzmann Machine," IEEE Trans., Neural Network, Vol. 6, No. 5, pp. 1229 - 1236.

C. T. Lin and C. S. G. Lee, "A Multi-Valued Boltzmann Machine," IEEE Trans., SMC, Vol. 25, No. 4, pp. 660 - 668.

W. Feller, An introduction to probability theory and its application, John Wiley and Sons, 1968.

Ying Liu, US Patent No. 7,773,800, http://www.google.com/patents/US7773800, Feb, 2002.

Ying Liu, "Image Compression Using Boltzmann machines," Proc. SPIE, Vol. 2032, pp. 103 - 117, July, 1993.

Ying Liu, "Boltzmann Machine for Image Block Coding," Proc. SPIE. Vol. 2424, pp. 434 - 447, Feb., 1995.

Ying Liu, "Character and Image Recognition and Image Retrieval Using the Boltzmann Machine," pp. 706 - 715, Proc. SPIE. Vol. 3077, April 1997.

Ying Liu, "Two New Classes of Boltzmann Machines," Proc. SPIE, Vol. 1966, pp.162 - 175, April, 1993.

Liu, Y. and Wang, S. (2018) Completeness Problem of the Deep Neural Networks. American Journal of Computational Mathematics, 8, 184-196. doi: 10.4236/ajcm.2018.82014.

Jolliffe I.T. Principal Component Analysis, Series: Springer Series in Statistics, 2nd ed., Springer, NY, 2002, XXIX, 487 p. 28 illus. ISBN 978-0-387-95442-4.

H., & Williams, L.J. "Principal component analysis.". Wiley Interdisciplinary Reviews: Computational Statistics, 2: 433– 459, (2010).

Olshausen, Bruno A. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images." Nature 381.6583 (1996): 607-609.

Gupta, N; Stopfer, M, "A temporal channel for information in sparse sensory coding.". Current Biology 24 (19): 2247–56 (October 2014)..

Bengio, Y. (2009). "Learning Deep Architectures for AI" (PDF). Foundations and Trends in Machine Learning 2.

Modeling word perception using the Elman network, Liou, C.-Y., Huang, J.-C. and Yang, W.-C., Neurocomputing, Volume 71, 3150–3157 (2008).

Autoencoder for Words, Liou, C.-Y., Cheng, C.-W., Liou, J.-W., and Liou, D.-R., Neurocomputing, Volume 139, 84–96 (2014).

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, "Learning internal representations by error propagation" (1986).

Bourlard, H.; Kamp, Y. (1988). "Auto-association by multilayer perceptrons and singular value decomposition". Biological Cybernetics 59 (4–5): 291–294.

Hinton & Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks (Science, 28 July 2006).

MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability 1. University of California Press. pp. 281–297.

Steinhaus, H. (1957). "Sur la division des corps matériels en parties". Bull. Acad. Polon. Sci. (in French) 4 (12): 801–804.

A. B. Lloyd, S. P. (1957). "Least square quantization in PCM". Bell Telephone Laboratories Paper. Published in journal much later: Lloyd., S. P. (1982). "Least squares quantization in PCM" (PDF). IEEE Transactions on Information Theory 28 (2): 129–137.

E.W. Forgy (1965). "Cluster analysis of multivariate data: efficiency versus interpretability of classifications". Biometrics 21: 768–769. TensorFlow, https://www.tensorflow.org/. Torch, http://torch.ch/. Theano, http://deeplearning.net/software/theano/introduction.html.

K. Hornik, M. Stinchcombe, and H. White (1989) Multilayer Feedforward Networks are Universal Approximators, Neural Networks, Vol. 2, pp. 359-366.

Nicolas Le Roux and Yoshua Bengio (2008) Representational Power of Restricted Boltzmann Machines and Deep Belief Networks, Neural Computation, Volume 20, Issue 6, pp.1631-1649.