LETTER **Communicated by Alberto Fachechi**

Full-Span Log-Linear Model and Fast Learning Algorithm

Kazuya Takabatake
k.takabatake@aist.go.jp
Shotaro Akaho
s.akaho@aist.go.jp
Human Informatics and Interaction Research Institute, National Institute of Advanced Industrial Science and Technology, Tsukuba, 305-8568, Japan

The full-span log-linear (FSLL) model introduced in this letter is considered an nth order Boltzmann machine, where n is the number of all variables in the target system. Let $X = (X_0, \ldots, X_{n-1})$ be finite discrete random variables that can take $|X| = |X_0| \ldots |X_{n-1}|$ different values. The FSLL model has $|X| - 1$ parameters and can represent arbitrary positive distributions of X. The FSLL model is a highest-order Boltzmann machine; nevertheless, we can compute the dual parameter of the model distribution, which plays important roles in exponential families in $O(|X| \log |X|)$ time. Furthermore, using properties of the dual parameters of the FSLL model, we can construct an efficient learning algorithm. The FSLL model is limited to small probabilistic models up to $|X| \approx 2^{25}$; however, in this problem domain, the FSLL model flexibly fits various true distributions underlying the training data without any hyperparameter tuning. The experiments showed that the FSLL successfully learned six training data sets such that $|X| = 2^{20}$ within 1 minute with a laptop PC.

1 Introduction

The main purpose of this letter is as follows:

- To introduce the full-span log-linear (FSLL) model and a fast learning algorithm
- To demonstrate the performance of the FSLL model with experiments

Boltzmann machines (Ackley, Hinton, & Sejnowski, 1985) are multivariate probabilistic models that are widely used in the field of machine learning. Here, we consider a Boltzmann machine with n binary variables $X = (X_0, \ldots, X_{n-1})(X_i = 0, 1)$. We address fully connected Boltzmann machines with no hidden variables and no temperature parameter. A Boltzmann machine represents the following distribution p_θ, which we refer to as the *model distribution*:

Neural Computation **34, 1398–1424** (2022)
https://doi.org/10.1162/neco_a_01496

$$p_\theta(x) = Z(\theta)^{-1} e^{l_\theta(x)}, \quad x_i \in \{0, 1\}, \quad Z(\theta) = \sum_{x \in X} e^{l_\theta(x)},$$

$$l_\theta(x) = \sum_{0 \le i < j < n} \theta_{ij} x_i x_j + \sum_{0 \le i < n} \theta_{in} x_i. \tag{1.1}$$

In Boltzmann machines, learning is achieved by minimizing $KL(p_d \| p_\theta)$, where $KL(* \| *)$ is the Kullback-Leibler(KL-) divergence and p_d is the empirical distribution of the training data. One straightforward method to minimize $KL(p_d \| p_\theta)$ is to use a gradient vector whose components are

$$\frac{\partial KL(p_d \| p_\theta)}{\partial \theta_{ij}} = \langle X_i X_j \rangle_{p_\theta} - \langle X_i X_j \rangle_{p_d} \tag{1.2}$$

(Ackley et al., 1985) and to apply the gradient descent method or quasi-Newton method (Dennis & Moré, 1977). In evaluating equation 1.2, the computational cost to evaluate the term $\langle X_i X_j \rangle_{p_\theta}$ is significant because we need to evaluate this term every time θ is modified.

One disadvantage of the Boltzmann machine is its insufficient ability to represent distributions. The Boltzmann machine has only $n(n+1)/2$ parameters, while the dimension of the function space spanned by the possible distributions of X is $|X| - 1$ (-1 comes from the constraint $\sum_{x \in X} p(x) = 1$).

One way to reduce this disadvantage is to introduce higher-order terms into the function $l_\theta(X)$. For example, third-order Boltzmann machines represent the following distribution p_θ (Sejnowski, 1986):

$$p_\theta(x) = Z(\theta)^{-1} e^{l_\theta(x)}, \quad x_i \in \{0, 1\}, \quad Z(\theta) = \sum_{x \in X} e^{l_\theta(x)},$$

$$l_\theta(x) = \sum_{0 \le i < j < k < n} \theta_{ijk} x_i x_j x_k + \sum_{0 \le i < j < n} \theta_{ijn} x_i x_j + \sum_{0 \le i < n} \theta_{inn} x_i.$$

Here, "order" means the number of variables on which a function depends. This definition of order is not limited to binary variables. For example, if $f(x_0, x_1, x_2)$ ignores x_2, that is, x_2 does not affect the value $f(x_0, x_1, x_2)$, the order of f is two. The kth-order Boltzmann machine has up to kth-order terms in l_θ. Since the nth-order Boltzmann machine has arbitrary order of terms, it can represent arbitrary positive distributions[1] of X.

However, introducing higher-order terms leads to an enormous increase in computational cost because the kth-order Boltzmann machine has $\sum_{i=1}^{k} \binom{n}{i}$ parameters.

[1] Distributions such that $\forall x,\ p(x) > 0$.

The FSLL model introduced in this letter can be considered an nth order Boltzmann machine,[2] where n is the number of all variables in the target system. The FSLL model has $|X| - 1$ parameters and can represent arbitrary positive distributions. Since the FSLL model is a highest-order Boltzmann machine, the learning of FSLL is expected to be very slow. However, we propose a fast learning algorithm. For example, this algorithm can learn a joint distribution of 20 binary variables within 1 minute with a laptop PC.

Since the FSLL model has full degrees of freedom, a regularization mechanism to avoid overfitting is essential. For this purpose, we used a regularization mechanism based on the minimum description length principle (Rissanen, 2007).

The remainder of this letter is organized as follows. In section 2, we present the FSLL model and its fast learning algorithm. In section 3, we demonstrate the performance of the FSLL model by experiment. In section 4, we discuss the advantages and disadvantages of the FSLL model. In section 5, we present the conclusions and extensions of the letter.

2 Full-Span Log-Linear Model

Before introducing the FSLL model, we define the notations used in this letter. A random variable is denoted by a capital letter, such as X, and the value that X takes is indicated by a lowercase letter, such as x. X also denotes the set of values that the variable X can take; thus, $|X|$ denotes the number of values that X can take. $\langle f \rangle_p$ denotes the expectation of $f(X)$ with distribution $p(X)$, that is, $\langle f \rangle_p = \sum_{x \in X} p(x) f(x)$. The differential operator $\partial / \partial \theta_y$ is abbreviated as ∂_y. $\mathcal{P}$ denotes the set of all distributions of X, and $\mathcal{P}+$ denotes all positive distributions of X.

2.1 Model Distribution. The FSLL model is a multivariate probabilistic model designed for a system that has n discrete finite variables $X_0, \ldots, X_{n-1}$, where X_i takes an integer value in $[0, |X_i|)$. The FSLL model has parameters $\theta = \{\theta_y\}$, where $y = (y_0, \ldots, y_{n-1})$ is a vector such that $y_i \in X_i$. The model distribution of the FSLL model is the following p_θ:

$$p_\theta(x) = Z(\theta)^{-1} e^{l_\theta(x)}, \quad x_i \in X_i, \quad Z(\theta) = \sum_{x \in X} e^{l_\theta(x)},$$

$$l_\theta(x) = \sum_{y \in X} \theta_y \Phi_y(x), \quad \Phi_y(x) = \prod_{i=0}^{n-1} \phi^i_{y_i}(x_i). \tag{2.1}$$

In equation 2.1, $\{\phi^i_{y_i}\}(y_i \in X_i)$ are $|X_i|$ linearly independent functions of X_i, which we refer to as the *local basis functions*, and $\{\Phi_y\}(y \in X)$ are $|X|$ functions of $X = (X_0, \ldots, X_{n-1})$, which we refer to as the *(global) basis functions*.

[2] Furthermore, the FSLL model is not limited to binary variables.

Using the following theorem recursively, we can prove that the global basis functions are linearly independent functions of X.

Theorem 1. *If* $\{f_i\}(i \in I)$ *are linearly independent functions of* X_0*, and* $\{g_j\}(j \in J)$ *are linearly independent functions of* X_1*, then* $\{f_i(X_0)g_j(X_1)\}(i \in I, j \in J)$ *are linearly independent functions of* (X_0, X_1)*.*

The proof is provided in the appendix.

In the FSLL model, we determine the local basis functions as follows:

Case $|X_i| = 2^k$:

$$H_1 = (1), \quad H_{2^{k+1}} = \begin{pmatrix} H_{2^k} & H_{2^k} \\ H_{2^k} & -H_{2^k} \end{pmatrix}, \tag{2.2}$$

$$\begin{pmatrix} \phi_0^i(0) & \dots & \phi_0^i(|X_i|-1) \\ \vdots & & \vdots \\ \phi_{|X_i|-1}^i(0) & \dots & \phi_{|X_i|-1}^i(|X_i|-1) \end{pmatrix} = H_{2^k}$$

H_{2^k} above is referred to as the *Walsh–Hadamard matrix* (Pratt, Andrews, & Kane, 1969).

Else:

$$\phi_0^i \equiv 1, \quad \phi_j^i(l) = \begin{Bmatrix} 1 & (0 < j = l) \\ -1 & (0 < j \neq l) \end{Bmatrix} \tag{2.3}$$

Since $\Phi_0 \equiv 1$, an arbitrary θ_0 gives the same model distribution. Therefore, we determine as $\theta_0 \equiv 0$.

2.2 Learning Algorithm. Algorithm 1 presents the outline of the learning algorithm of the FSLL model. This algorithm is a greedy search to find a local minimum point of $cost(\theta)$, which monotonically decreases as the iteration progresses.

In line 7 of the algorithm, *candidate* denotes θ' derived from θ by applying one of the following modifications on the yth component:

Candidate derived by appending θ_y: If $\theta_y = 0$, then let $\theta'_y = \arg\min_{\theta_y} cost(\theta)$.

Candidate derived by adjusting θ_y: If $\theta_y \neq 0$, then let $\theta'_y = \arg\min_{\theta_y} cost(\theta)$.

Candidate derived by removing θ_y: If $\theta_y \neq 0$, then let $\theta'_y = 0$.

2.2.1 Cost Function. We use a cost function based on the minimum description length principle (Rissanen, 2007). Suppose that we send the

Algorithm 1: Learning Algorithm.

θ: parameters
p_θ: model distribution
p_d: empirical distribution of training data
$\bar{\theta}$: dual parameters of θ
ϵ: threshold to halt the loop
r_y: regularization term for y such that $\theta_y \neq 0$

```
function LEARN(training data)
    Compute d̄ from p_d (section 2.2.1)
    Compute r_y(Eq.(7)) for all y ∈ X
    θ ← 0, p_θ ←uniform distribution
    loop
        Compute θ̄ from p_θ (section 2.2.2)
        Evaluate all candidates θ' and find θ^1 = arg min_θ' cost(θ')
        if cost(θ) − cost(θ^1) < ε then return θ, p_θ
        end if
        θ ← θ^1
        Update p_θ
    end loop
end function
```

training data to a receiver by transmitting the parameters and compressed data. Since θ is a sparse vector, we transmit only indexes y, such that $\theta_y \neq 0$, and the value of θ_y. Moreover, the index $y = (y_0, \ldots, y_{n-1})$ is a sparse vector because higher-order basis functions are rarely used in the model distribution due to their expensive cost. Therefore, we transmit only indexes i, such that $y_i \neq 0$, and the values of $y_i \in [1, |X_i|)$ to transmit the sparse vector y. Then the description length[3] to transmit the sparse vector y becomes

$$\sum_{i:y_i \neq 0} \underbrace{\ln n}_{\text{to send } i \in [0,n)} + \underbrace{\ln(|X_i| - 1)}_{\text{to send } y_i \in [1,|X_i|)} = \sum_{i:y_i \neq 0} \ln n(|X_i| - 1),$$

and the description length to transmit all index vectors y, such that $\theta_y \neq 0$, becomes

$$\sum_{y:\theta_y \neq 0} \sum_{i:y_i \neq 0} \ln n(|X_i| - 1).$$

The minimum description length to transmit k parameters and the compressed data is estimated as (Rissanen, 2007)

[3] We use "nat" as the description length unit; thus, we use "ln" instead of "log."

$$-N\left\langle \ln p_\theta \right\rangle_{p_d} + \frac{k}{2}\ln N,$$

where k is the number of nonzero parameters and N is the number of samples in the training data. The total description length to transmit y, θ_y and the compressed data becomes

$$\begin{aligned} &-N\left\langle \ln p_\theta \right\rangle_{p_d} + \frac{k}{2}\ln N + \sum_{y:\theta_y\neq 0}\sum_{i:y_i\neq 0}\ln n(|X_i|-1) \\ &= -N\left\langle \ln p_\theta \right\rangle_{p_d} + \sum_{y:\theta_y\neq 0}\left(\frac{\ln N}{2} + \sum_{i:y_i\neq 0}\ln n(|X_i|-1)\right). \end{aligned} \tag{2.4}$$

We divide equation 2.4 by N and add $\left\langle \ln p_d \right\rangle_{p_d}$ to create information-geometric quantity and obtain the following cost function:

$$cost(\theta, N) = KL(p_d \| p_\theta) + r(\theta, N), \quad r(\theta, N) = \sum_{y:\theta_y\neq 0} r_y(N),$$

$$r_y(N) = \frac{1}{N}\left(\frac{\ln N}{2} + \sum_{i:y_i\neq 0}\ln n(|X_i|-1)\right). \tag{2.5}$$

2.2.2 Fast Algorithm to Compute $\bar{\theta}$. The following vector $\bar{\theta}$, referred to as the *dual parameter* of θ, plays important roles in the exponential families (Amari, 2016):

$$\begin{aligned} \bar{\theta}_y(\theta) &= \partial_y \ln Z(\theta) \\ &= \left\langle \Phi_y \right\rangle_{p_\theta} \quad \text{(see appendix for derivation).} \end{aligned} \tag{2.6}$$

We identified an algorithm to compute $\bar{\theta}$ from p_θ in $O(|X|\log|X|)$ time. This algorithm borrowed ideas from the multidimensional discrete Fourier transform (DFT) (Smith, 2010). Here, we consider a two-dimensional (2D-)DFT[4] for $|X_0| \times |X_1|$ pixels of data. The 2D-DFT transforms f into F by the following equation:

$$F(y_0, y_1) = \sum_{x_0, x_1} f(x_0, x_1)\Phi_{y_0 y_1}(x_0, x_1),$$

$$\Phi_{y_0 y_1}(x_0, x_1) = \exp\left(\frac{2\pi i}{|X_0|}y_0 x_0\right)\exp\left(\frac{2\pi i}{|X_1|}y_1 x_1\right).$$

[4] Not 2D-FFT but 2D-DFT.

To derive the value of $F(y_0, y_1)$ for a specific (y_0, y_1), $O(|X_0||X_1|)$ time is required; therefore, it appears that $O(|X_0|^2|X_1|^2)$ time is required to derive $F(y_0, y_1)$ for all y_0, y_1. However, 2D-DFT is usually realized in the following dimension-by-dimension manner:

$$F(y_0, y_1) = \underbrace{\sum_{x_1} \underbrace{\left(\sum_{x_0} f(x_0, x_1) \exp\left(\frac{2\pi i}{|X_0|} y_0 x_0 \right) \right)}_{\text{DFT by } X_0} \exp\left(\frac{2\pi i}{|X_1|} y_1 x_1 \right)}_{\text{DFT by } X_1}. \quad (2.7)$$

In equation 2.7, the DFT by X_0 for all $(y_0, x_1) \in X_0 \times X_1$ requires $O(|X_0||X_1|)$ time, and the DFT by X_1 for all $y_0 \in X_0, y_1 \in X_1$ requires $O(|X_0||X_1|^2)$ time. Therefore, the entire 2D DFT requires $O(|X_0||X_1|(|X_0| + |X_1|))$ time that is smaller than $O(|X_0|^2|X_1|^2)$. The key here is that the basis function $\Phi_{y_0 y_1}(x_0, x_1)$ is a product of two univariate functions as follows:

$$\Phi_{y_0 y_1}(x_0, x_1) = \phi^0_{y_0}(x_0)\phi^1_{y_1}(x_1),$$
$$\phi^0_{y_0}(x_0) = \exp\left(\frac{2\pi i}{|X_0|} y_0 x_0 \right), \quad \phi^1_{y_1}(x_1) = \exp\left(\frac{2\pi i}{|X_1|} y_1 x_1 \right).$$

We apply this principle to compute $\bar{\theta}$.

Here, we consider how to derive $\bar{\theta}$ from p_θ by the following equation:

$$\begin{aligned} \langle \Phi_y \rangle_{p_\theta} &= \sum_x p_\theta(x)\Phi_y(x) \\ &= \sum_x p_\theta(x) \prod_i \phi^i_{y_i}(x_i) \\ &= \sum_{x_{n-1}} \left(\cdots \left(\sum_{x_1} \left(\sum_{x_0} p_\theta(x)\phi^0_{y_0}(x_0) \right) \phi^1_{y_1}(x_1) \right) \cdots \right) \phi^{n-1}_{y_{n-1}}(x_{n-1}). \end{aligned} \quad (2.8)$$

We evaluate the right side of equation 2.8 from the innermost parenthesis to the outermost parenthesis; that is, we determine the following function, $g^i : X \to \mathbb{R}$ by the following recurrence sequence:

$$\begin{aligned} &g^0(x) = p_\theta(x), \\ &g^{i+1}(y_0, \ldots, y_{i-1}, y_i, x_{i+1}, \ldots, x_{n-1}) \\ &\quad = \sum_{x_i} g^i(y_0, \ldots, y_{i-1}, x_i, x_{i+1}, \ldots, x_{n-1})\phi^i_{y_i}(x_i). \end{aligned} \quad (2.9)$$

Then the equation $\bar{\theta}_y = g^n(y)$ holds. Since $O(|X||X_i|)$ time is required to obtain g^{i+1} from g^i, we can obtain g^n from g^0 in $O(|X| \sum_i |X_i|)$ time. In the case where $|X_i| = k$, the computational cost to derive g^n from g^0 becomes $O(|X|kn)$. Moreover, considering k as a constant, we obtain the computational cost as $O(|X|n) = O(|X| \log |X|)$.

We can use the same algorithm to obtain the vector $\bar{d}$ from p_d. In this case, let $g^0 = p_d$.

2.2.3 Acceleration by Walsh-Hadamard Transform. In cases where $|X_i|$ is large, we can accelerate the computation of $\bar{\theta}$ by using Walsh-Hadamard transform (WHT) (Fino & Algazi, 1976).

Let us recall the 2D-DFT in section 2.2.2. If $|X_0|, |X_1|$ are powers of two, we can use the fast Fourier transform (FFT) (Smith, 2010) for DFT by X_0 and DFT by X_1 in equation 2.7 and can reduce the computational cost to $O(|X_0||X_1| \log |X_0||X_1|)$. We can apply this principle to computing $\bar{\theta}$.

Here, we fix the values of $y_0, \ldots, y_{i-1}, x_{i+1}, \ldots, x_{|X_i|-1}$(only the ith component is omitted) in equation 2.9. Then we obtain

$$g^{i+1}(\ldots, \underset{i}{y_i}, \ldots) = \sum_{x_i} g^i(\ldots, \underset{i}{x_i}, \ldots)\phi^i_{y_i}(x_i).$$

Using matrix notation, we obtain

$$\begin{pmatrix} g^{i+1}(\ldots, \underset{i}{0}, \ldots) \\ \vdots \\ g^{i+1}(\ldots, \underset{i}{|X_i|-1}, \ldots) \end{pmatrix} = \begin{pmatrix} \phi^i_0(0) & \ldots & \phi^i_0(|X_i|-1) \\ \vdots & & \vdots \\ \phi^i_{|X_i|-1}(0) & \ldots & \phi^i_{|X_i|-1}(|X_i|-1) \end{pmatrix} \begin{pmatrix} g^i(\ldots, \underset{i}{0}, \ldots) \\ \vdots \\ g^i(\ldots, \underset{i}{|X_i|-1}, \ldots) \end{pmatrix}. \quad (2.10)$$

We refer to this transform $\mathbb{R}^{|X_i|} \to \mathbb{R}^{|X_i|}$ as the *local transform*. The local transform usually requires $O(|X_i|^2)$ time; however, if the matrix in equation 2.10 is a Walsh-Hadamard matrix (see equation 2.2), using Walsh-Hadamard transform (WHT) (Fino & Algazi, 1976), we can perform the local transform in $O(|X_i| \log |X_i|)$ time.[5]

Since the number of all combinations of $y_0, \ldots, y_{i-1}, x_{i+1}, \ldots, x_{n-1}$ (the ith component is omitted) is $|X|/|X_i|$, we can obtain the function

[5] For $k \leq 4$, directly multiplying the Hadamard matrix is faster than the WHT in our environment; therefore, we use the WHT only in cases where $k \geq 5$.

g^{i+1} from g^i in $O(|X|\log|X_i|)$ time. Moreover, g^n is obtained from g^0 in $O(\sum_i |X|\log|X_i|) = O(|X|\log|X|)$ time.

2.2.4 $O(1)$ Algorithm to Evaluate Candidate. In line 7 of algorithm 1, all three types of candidates—appending, adjusting, and removing—are evaluated for all $y \in X$. Here, we consider the following equation:

$$\partial_y \bar{\theta}_y = \left\langle \Phi_y^2 \right\rangle_{p_\theta} - \bar{\theta}_y^2 \quad \text{(see the appendix for the derivation)} \tag{2.11}$$

$$= 1 - \bar{\theta}_y^2 \quad (\because \Phi_y^2 \equiv 1). \tag{2.12}$$

Considering $\bar{\theta}_y$ as a univariate function of θ_y and equation 2.12 as an ordinary differential equation, we obtain the following general solution:

$$\bar{\theta}_y(\theta_y) = \tanh(\theta_y - c) \quad \text{(see the appendix for the derivation)}, \tag{2.13}$$

where c is a constant determined by a boundary condition. For example, if $\bar{\theta}_y(\theta_y^0) = \bar{\theta}_y^0$, then c is given by $c = \theta_y^0 - \tanh^{-1}\bar{\theta}_y^0$ and equation 2.13 becomes

$$\bar{\theta}_y(\theta_y) = \tanh(\theta_y - \theta_y^0 + \tanh^{-1}\bar{\theta}_y^0). \tag{2.14}$$

Here, we define the follwing line $A_y(\theta^0)$ in $\mathcal{P}$:

$$A_y(\theta^0) = \{\theta | \forall y' \neq y,\ \theta_y = \theta_y^0\}. \tag{2.15}$$

Equation 2.14 demonstrates that if $\bar{\theta}_y^0$ is known, we can derive any $\bar{\theta}_y$ at a point on the line $A_y(\theta^0)$ in $O(1)$ time.

Here, the gradient vector of $KL(p_d \| p_\theta)$ is given by

$$\partial_y KL(p_d \| p_\theta) = \bar{\theta}_y - \bar{d}_y,$$
$$\bar{d}_y = \left\langle \Phi_y \right\rangle_{p_d} \quad \text{(see the appendix for the derivation)}. \tag{2.16}$$

Therefore, for $\theta \in A_y(\theta^0)$, we can obtain $KL(p_d \| p_\theta) - KL(p_d \| p_{\theta^0})$ by integrating equation 2.16 as follows:

$$KL(p_d \| p_\theta) - KL(p_d \| p_{\theta^0}) = \int_{\theta_y^0}^{\theta_y} \bar{\theta}_y(u) - \bar{d}_y \, du$$

$$= \frac{1+\bar{d}_y}{2} \ln \frac{1+\bar{\theta}_y^0}{1+\bar{\theta}_y} + \frac{1-\bar{d}_y}{2} \ln \frac{1-\bar{\theta}_y^0}{1-\bar{\theta}_y} \tag{2.17}$$

(see the appendix for the derivation).

Then we can evaluate $\Delta(\theta') = cost(\theta') - cost(\theta^0)$ for three types of candidates θ' as follows:

Appending θ_y: By equation 2.16, $KL(p_d \| p_\theta)$ is minimized when $\bar{\theta}_y = \bar{d}_y$. Therefore,

$$\Delta = \frac{1+\bar{d}_y}{2} \ln \frac{1+\bar{\theta}^0_y}{1+\bar{d}_y} + \frac{1-\bar{d}_y}{2} \ln \frac{1-\bar{\theta}^0_y}{1-\bar{d}_y} + r_y. \tag{2.18}$$

Adjusting θ_y: By equation 2.16, $KL(p_d \| p_\theta)$ is minimized when $\bar{\theta}_y = \bar{d}_y$. Therefore,

$$\Delta = \frac{1+\bar{d}_y}{2} \ln \frac{1+\bar{\theta}^0_y}{1+\bar{d}_y} + \frac{1-\bar{d}_y}{2} \ln \frac{1-\bar{\theta}^0_y}{1-\bar{d}_y}.$$

Removing θ_y: θ_y becomes 0. Therefore,

$$\Delta = \frac{1+\bar{d}_y}{2} \ln \frac{1+\bar{\theta}^0_y}{1+\bar{\theta}_y(0)} + \frac{1-\bar{d}_y}{2} \ln \frac{1-\bar{\theta}^0_y}{1-\bar{\theta}_y(0)} - r_y.$$

Among the three types of candidates—appending, adjusting, and removing—the group of candidates by appending θ_y has almost $|X|$ candidates because θ_y is a very sparse vector. Therefore, it is important to reduce the computational cost to evaluate candidates by appending. Evaluating Δ in equation 2.18 is an expensive task for a central processing unit (CPU) because it involves logarithm computation.[6]

Δ in equation 2.18 has the following lower bound $\underline{\Delta}$:

$$\begin{aligned} \Delta &\geq \underline{\Delta} \\ &= -\frac{(\bar{\theta}^0_y - \bar{d}_y)^2}{1-(\bar{\theta}^0_y)^2} + r_y \quad \text{(see the appendix for the derivation).} \end{aligned} \tag{2.19}$$

Evaluating $\underline{\Delta}$ is much faster than evaluating Δ. In the candidate evaluation, the candidate with the lower cost is the "winner." If a candidate's lower bound is greater than the champion's cost—the lowest cost ever found—the candidate has no chance to win; therefore, we can discard the candidate without precise evaluation of Δ.

Algorithm 2 presents the details of line 7 of algorithm 1. In line 5, if $\underline{\Delta}(\theta'_y) \geq \Delta_{\theta^1}$, the candidate θ' is discarded and the evaluation of $\Delta(\theta'_y)$ is

[6] Logarithm computation is 30 times slower than addition or multiplication in our environment.

Algorithm 2: Details of Line 7 of Algorithm 1.

```
1:  Δ_{θ^1} ← 0
2:  for all y ∈ X do
3:      if θ_y = 0 then
4:          θ' ←candidate by appending θ_y
5:          if Δ(θ') < Δ_{θ^1} then
6:              if Δ(θ') < Δ_{θ^1} then θ^1_y ← θ'_y, Δ_{θ^1} ← Δ(θ'), y^1 ← y
7:              end if
8:          end if
9:      else
10:         θ' ←candidate by adjusting θ_y
11:         if Δ(θ') < Δ_{θ^1} then θ^1_y ← θ'_y, Δ_{θ^1} ← Δ(θ'), y^1 ← y
12:         end if
13:         θ' ←candidate by removing θ_y
14:         if Δ(θ') < Δ_{θ^1} then θ^1_y ← θ'_y, Δ_{θ^1} ← Δ(θ'), y^1 ← y
15:         end if
16:     end if
17: end for
```

skipped. This skipping effectively reduces the computational cost of evaluating candidates.[7]

2.2.5 Updating θ and p_θ. Let θ^0 denote θ before updating and θ^1 denote θ after updating in line 10 of algorithm 1. The θ^1 differs from θ^0 only at the y'th component. Therefore,

$$l_{\theta^1}(x) = l_{\theta^0}(x) + (\theta^1_{y'} - \theta^0_{y'})\Phi_{y'}(x)$$

and

$$\begin{aligned} p_{\theta^1}(x) &= Z(\theta^1)^{-1} \exp l_{\theta^1}(x) \\ &= Z(\theta^1)^{-1} \exp\left(l_{\theta^0}(x) + (\theta^1_{y'} - \theta^0_{y'})\Phi_{y'}(x)\right) \\ &= \frac{Z(\theta^0)}{Z(\theta^1)} p_{\theta^0}(x) \exp\left((\theta^1_{y'} - \theta^0_{y'})\Phi_{y'}(x)\right). \end{aligned} \tag{2.20}$$

Summing equation 2.20 for all $x \in X$, we obtain

$$1 = \frac{Z(\theta^0)}{Z(\theta^1)} \sum_x p_{\theta^0}(x) \exp\left((\theta^1_{y'} - \theta^0_{y'})\Phi_{y'}(x)\right).$$

[7] In our environment, this skipping makes the evaluation of candidates more than 10 times faster.

Algorithm 3: Details of Lines 10–11 of Algorithm 1.

1: $sum \leftarrow 0,\ c_+ \leftarrow \exp(\theta^1_{y'} - \theta_{y'}),\ c_- \leftarrow 1/c_+$
2: $\theta_{y^1} \leftarrow \theta^1_{y^1}$
3: **for all** $x \in X$ **do**
4: **if** $\Phi_{y^1}(x) = 1$ **then** $p_\theta(x) \leftarrow c_+ p_\theta(x)$
5: **else** $p_\theta(x) \leftarrow c_- p_\theta(x)$
6: **end if**
7: $sum \leftarrow sum + p_\theta(x)$
8: **end for**
9: **for all** $x \in X$ **do** $p_\theta(x) \leftarrow p_\theta(x)/sum$
10: **end for**

Therefore,

$$p_{\theta^1}(x) = \frac{Z(\theta^0)}{Z(\theta^1)} p_{\theta^0}(x) \exp\left((\theta^1_{y'} - \theta^0_{y'})\Phi_{y'}(x)\right),$$

$$\frac{Z(\theta^0)}{Z(\theta^1)} = \left(\sum_x p_{\theta^0}(x) \exp\left((\theta^1_{y'} - \theta^0_{y'})\Phi_{y'}(x)\right)\right)^{-1}. \tag{2.21}$$

Algorithm 3 presents the details of lines 10 and 11 in algorithm 1. This algorithm requires $O(|X|)$ time. It should be noted that the expensive exponential computation $\exp\left((\theta^1_{y'} - \theta^0_{y'})\Phi_{y'}(x)\right)$ is not used in the for-loop.

2.2.6 Memory Requirements. In the FSLL model, most of the memory consumption is dominated by four large tables for $p_\theta, \bar{\theta}, \bar{d}$, and r, and each stores $|X|$ floating point numbers. However, θ does not require a large amount of memory because it is a sparse vector.

For example, if the FSLL model is allowed to use 4 GB of memory, it can handle up to 26 binary variables, 16 three-valued variables($|X_i| = 3$), and 13 four-valued variables.

2.3 Convergence to Target Distribution. One major point of interest about algorithm 1 is whether the model distribution converges to a target distribution at the limit of $N \to \infty$ or not, where N is the number of samples in the training data. As a result, we can guarantee this convergence.

Let us define the following symbols:

$$\begin{aligned} A(\theta^t) &= \cup_y A_y(\theta^t) \quad // \ A_y \text{ is defined by equation 2.15,} \\ m_y(\theta^t, N) &= \min_{\theta \in A_y(\theta^t)} f_N(\theta) \quad // \text{ minimum in line } A_y \\ m(\theta^t, N) &= \min_{\theta \in A(\theta^t)} f_N(\theta) \quad // \text{ minimum in all lines } A_y \\ &= \min_y m_y(\theta^t, N), \end{aligned} \tag{2.22}$$

$$
\begin{aligned}
Y_{\min} &= \{y | m_y(\theta^t, N) = m(\theta, N)\}, \\
\arg m_y(\theta^t, N) &= \{\theta | \theta \in A_y(\theta^t),\ f_N(\theta) = m_y(\theta^t, N)\}, \\
\arg m(\theta^t, N) &= \{\theta | \theta \in A(\theta^t),\ f_N(\theta) = m(\theta^t, N)\}, \\
&= \cup_{y \in Y_{\min}} \arg m_y(\theta^t, N). \qquad (2.23)
\end{aligned}
$$

Then, algorithm 1 iteratively selects θ^{t+1} from $\arg m(\theta^t, N)$.

We first consider the case where the cost function is a continuous function of θ with no regularization term. Here, if and only if $f(\theta) = \min_{\theta' \in A(\theta)} f(\theta')$, we say that θ is an *axis minimum* of f. Then the following theorem holds (Beck, 2015):

Theorem 2. *Let $f : \mathbb{R}^{|X|} \to \mathbb{R}$ be a continuous cost function such that $B = \{\theta | f(\theta) \leq f(\theta^0)\}$ is a bounded close set. Then any accumulation point of $\{\theta^t\}(t \in [0, \infty))$ is an axis minimum of f.*

The proof is provided in the appendix. Here, if $f(\theta)$ has an unique axis minimum at $\theta = a$, the following corollary is derived from theorem 2:

Corollary 1. *Let f be a function satisfying the condition in theorem 2. If f has a unique axis minimum at $\theta = \theta_{\min}$, then $\theta_{\min}$ is also the global minimum and $\lim_{t \to \infty} \theta^t = \theta_{\min}$.*

The proof is provided in the appendix.

Let q be a positive distribution. By corollary 1, in the case where the cost function is $f(\theta) = KL(q \| p_\theta)$, where q is a positive distribution,[8] the equation $\lim_{t \to \infty} KL(q \| p_{\theta^t}) = 0$ holds.

Then we extend the cost function to $f_N(\theta) = f(\theta) + r(\theta, N)$, where $f : \mathbb{R}^{|X|} \to \mathbb{R}$ is a continuous function having the unique global minimum at $\theta = \theta_{\min}$, and $r(\theta, N)$ is a regularization term in equation 2.5. The following theorem holds:

Theorem 3. *Let f be a function satisfying the conition in theorem 2. Then*

$$\lim_{N \to \infty} \lim_{t \to \infty} f(\theta^t(N)) = \min_\theta f(\theta)$$

holds.

Here, do not confuse $\lim_{N \to \infty} \lim_{t \to \infty} f(\theta^t(N))$ with $\lim_{t \to \infty} \lim_{N \to \infty} f(\theta^t(N))$.

$$\lim_{t \to \infty} \lim_{N \to \infty} f(\theta^t(N)) = \min_\theta f(\theta)$$

[8] Positivity is needed to keep B bounded.

is trivial by corollary 1, while

$$\lim_{N\to\infty}\lim_{t\to\infty} f(\theta^t(N)) = \min_{\theta} f(\theta)$$

needs a proof. The proof is provided in the appendix. In the case where p_d is a positive distribution for sufficiently large N and $f(\theta) = KL(p_d|p_\theta)$, theorem 3 guarantees

$$\lim_{N\to\infty}\lim_{t\to\infty} KL(p_d\|p_{\theta^t(N)}) = 0.$$

3 Experiments

In this section, to demonstrate the performance of the FSLL model, we compare a full-span log-linear model that we refer to as *FL* with two Boltzmann machines that we refer to as *BM-DI* and *BM-PCD*.

3.1 Full-Span Log-Linear Model FL. FL is a full-span log-linear model that has been described in section 2. The model distribution of FL is given by equation 2.1. The cost function is given by equation 2.5. The learning algorithm is algorithm 1 described in section 2.2. The threshold to finish the cost minimization is determined as $\epsilon = 10^{-4}$.

3.2 Boltzmann Machine BM-DI. BM-DI (Boltzmann machine with direct integration) is a fully connected Boltzmann machine having no hidden variables and no temperature parameter. To examine the ideal performance of the Boltzmann machine, we do not use the Monte Carlo approximation in BM-DI to evaluate equation 1.2. The model distribution of BM-DI is given by equation 1.1. The cost function of BM-DI is $KL(p_d\|p_\theta)$ and has no regularization term.

To minimize the cost function with fewer evaluations of the gradient vector, we used a pseudo-Newton method, the *Broyden-Fletcher-Goldfarb-Shanno (BFGS)* algorithm (Nocedal & Wright, 2006), implemented in the Java statistical analysis tool (JSAT; Raff, 2017).

3.3 Boltzmann Machine BM-PCD. BM-PCD (Boltzman machine with persistent contrastive divergence method) is similar to BM-DI; however, BM-PCD uses the persistent contrastive divergence method (Tieleman, 2008), a popular Monte Carlo method in Boltzmann machine learning. BM-PCD has some hyperparameters. We tested various combinations of these hyperparameters and determined them as learning rate = 0.01, number of Markov chains = 100, and length of Markov chains = 10,000.

3.4 Training Data. We prepared six training data sets. These data sets are artificial; therefore, their true distributions $p_*(X)$ are known. Each is an

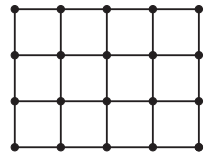

Figure 1: Graphical structure of Ising5 × 4S/L.

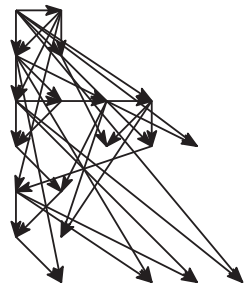

Figure 2: Graphical structure of BN20-37S/L.

independent and identically distributed (i.i.d.) data set drawn from its true distribution.

3.4.1 Ising5 × 4S, Ising5 × 4L. These data sets were drawn from the distribution represented by the 2D Ising model (Newman & Barkema, 1999) with 5 × 4 nodes (see Figure 1).

Every X_i takes the value 0 or 1. Ising5 × 4S has 1000 samples, while Ising5 × 4L has 100,000 samples. The true distribution is represented as

$$p_*(x) = Z^{-1} \exp\left(\frac{1}{2}\sum_{\langle i,j\rangle} s_i s_j\right), \quad Z = \sum_x \exp\left(\frac{1}{2}\sum_{\langle i,j\rangle} s_i s_j\right),$$

$$s_i = \left\{\begin{array}{ll} 1 & x_i = 1 \\ -1 & x_i = 0 \end{array}\right\},$$

where $\langle i, j\rangle$ denotes the adjacent variables in Figure 1. Boltzmann machines can represent p_*.

3.4.2 BN20-37S, BN20-37L. These data sets were drawn from the distribution represented by the Bayesian network with 20 nodes and 37 edges (see Figure 2).

X_0 has no parents, X_1 has X_0 as a parent, and the other $X_i (i \geq 2)$ individually has two parents $Y_{i0}, Y_{i1} \in \{X_0, \ldots, X_{i-1}\}$. The graphical structure and

contents of the conditional distribution table of each node are determined randomly. Every X_i takes the value 0 or 1. BN20-37S has 1000 samples, while BN20-37L has 100,000 samples. The true distribution is represented as

$$p_*(x) = \prod_i p_*(x_i|y_{i0}, y_{i1})$$
$$= \exp\left(\sum_i \ln p_*(x_i|y_{i0}, y_{i1})\right).$$

Since $\ln p_*(x_i|y_{i0}, y_{i1})$ are third-order terms, third-order Boltzmann machines can represent p_*.

3.4.3 BN20-54S, BN20-54L. These data sets were drawn from the distribution represented by the following Bayesian network with 20 nodes and 54 edges. X_0 has no parents, X_1 has X_0 as a parent, X_2 has X_0, X_1 has parents, and the other $X_i (i \geq 3)$ individually has three parents: $Y_{i0}, Y_{i1}, Y_{i2} \in \{X_0, \ldots, X_{i-1}\}$. The graphical structure and contents of the conditional distribution table of each node are determined randomly. Every X_i takes the value zero or one. BN20-54S has 1000 samples, and BN20-54L has 100,000 samples. The true distribution is represented as

$$p_*(x) = \prod_i p_*(x_i|y_{i0}, y_{i1}, y_{i2})$$
$$= \exp\left(\sum_i \ln p_*(x_i|y_{i0}, y_{i1}, y_{i2})\right).$$

Since $\ln p_*(x_i|y_{i0}, y_{i1}, y_{i2})$ are fourth-order terms, fourth-order Boltzmann machines can represent p_*.

3.5 Experimental Platform. All experiments were conducted on a laptop PC (CPU: Intel Core i7-6700K @4 GHz; memory: 64 GB; operating system: Windows 10 Pro). All programs were written in and executed on Java 8.

3.6 Results. Table 1 represents performance comparisons between FL BM-DI and BM-PCD. We evaluated the accuracy of the learned distribution by $KL(p_*\|p_\theta)$. Figure 3 illustrates the comparison of $KL(p_*\|p_\theta)$.

For Ising5 $\times$ 4S/L, a performance difference between FL and BMs (BM-DI and BM-PCD) was not remarkable because both FL and BMs could represent the true distribution p_*. The fact that $KL(p_d\|p_\theta) \gg KL(p_*\|p_\theta)$ implies that overfitting to p_d was successfully suppressed. FL used fewer basis functions than BMs used, which implies that some basis functions of BM were useless to represent p_*. Regarding the accuracy of the model distribution,

Table 1: Performance Comparison between FL and BM.

Data	Model	$KL(p_d \| p_\theta)$	$KL(p_* \| p_\theta)$	#Basis	Time
Ising5 × 4S	FL	2.501 nat	0.012 nat	31	5 sec
	BM-DI	2.424	0.087	210	13
	BM-PCD	2.504	0.094	210	3
Ising5 × 4L	FL	0.476	0.004	37	9
	BM-DI	0.473	0.002	210	12
	BM-PCD	0.528	0.053	210	3
BN20-37S	FL	4.355	0.317	39	5
	BM-DI	4.746	0.863	210	17
	BM-PCD	4.803	0.903	210	3
BN20-37L	FL	0.697	0.026	105	12
	BM-DI	1.422	0.750	210	19
	BM-PCD	1.477	0.806	210	3
BN20-54S	FL	3.288	0.697	41	5
	BM-DI	3.743	1.301	210	23
	BM-PCD	3.826	1.338	210	3
BN20-54L	FL	0.430	0.057	192	23
	BM-DI	1.545	1.166	210	21
	BM-PCD	1.620	1.242	210	3

Notes: p_d: empirical distribution of training data. p_*: true distribution. #Basis: number of used ($\theta_y \neq 0$) basis functions. Time: CPU time for learning(median of three trials).

BM-PCD is less accurate than FL and BM-DI. This disadvantage becomes noticeable when the model distribution is close to the true distribution. Even when large training data are given, some error remains in the model distribution of BM-PCD (e.g., Ising5 × 4L).

For BN20-37S/L and BN20-54S/L, FL outperformed BMs because only FL could represent p_*. To fit p_*, FL adaptively selected 39 basis functions for BN20-37S and 105 basis functions for BN20-37L from $|X| - 1 = 2^{20} - 1$ basis functions. This fact implies that FL constructed a more complex model to fit p_* as the training data increased. Furthermore, a comparison of $KL(p_* \| p_\theta)$ revealed that the accuracy of the model distribution was remarkably improved as the size of training data increased in FL. In contrast, BMs could not fit p_* even if a large training data set was supplied.

Figure 4 illustrates the CPU time to learn the training data sets. BM-PCD was the fastest, and FL was faster than BM-DI for five out of six training data sets. The learning time of BM-PCD is constant because we used a fixed length (10,000) of Markov chains. FL had $|X| - 1 = 2^{20} - 1$ basis functions, while BM-DI had $n(n+1)/2 = 210$ basis functions. Nevertheless, FL was faster than BM-DI.

For BN20-54L, FL takes a long time to learn because it uses 192 basis functions to construct the model distribution. Using the 192 basis functions, FL successfully constructed a model distribution that fit p_*, while BMs failed.

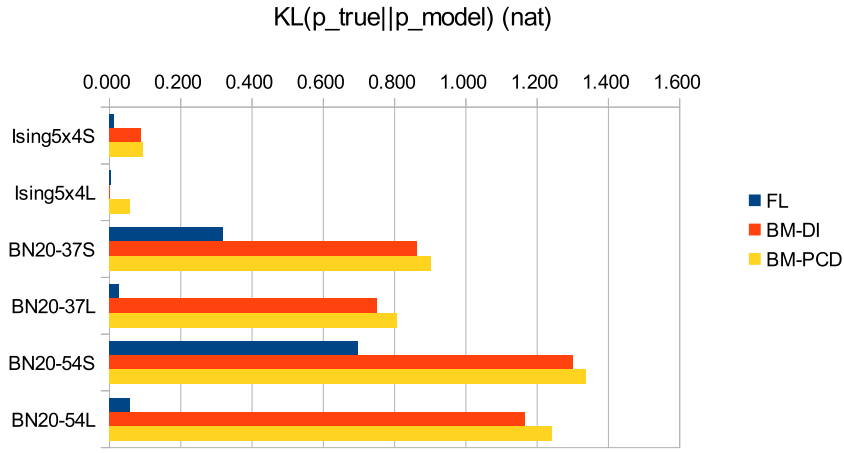

Figure 3: Comparing accuracy of model by $KL(p_* \| p_\theta)$ (lower is better).

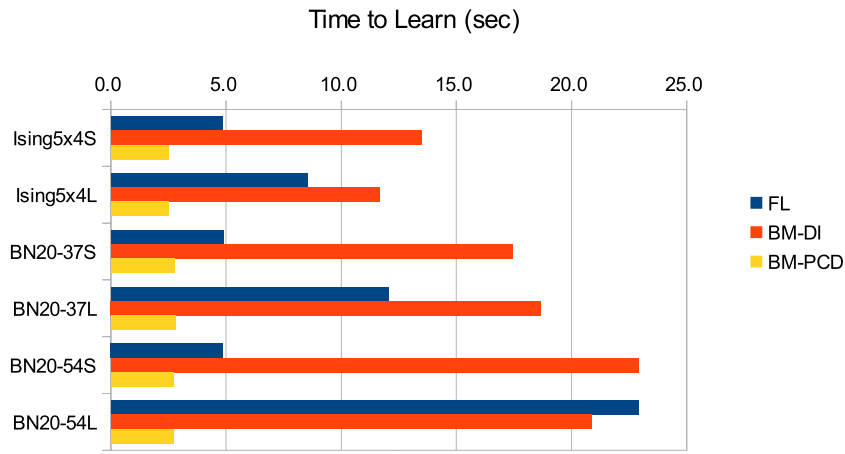

Figure 4: Comparing CPU time to learn (lower is better).

4 Discussion

The major disadvantage of the FSLL model is that it is not feasible for large problems due to memory consumption and learning speed. If we use a typical personal computer, the problem size should be limited as $|X| \lesssim 2^{25}$. However, as far as we use the FSLL model in this problem domain, the model has the following theoretical and practical advantages.

The first advantage is that the FSLL model can represent arbitrary distributions of X. Furthermore, it is guaranteed that the model distribution

converges to any target distribution as the limit of the training data size is infinity.

Here, we view learning machines from an information geometry perspective. The dimension of the distribution space $\mathcal{P}+$ (all positive distributions of X) is $|X|-1$, and a learning machine having M parameters spans an M-dimensional manifold in $\mathcal{P}+$ to represent its model distribution (we refer to this manifold as the *model manifold*).

A learning machine with $M < |X|-1$ parameters cannot represent arbitrary distributions in $\mathcal{P}$. Moreover, if $M < |X|-1$, there is no guarantee that the true distribution is close to the model manifold, and if the model manifold is remote from the true distribution, the machine's performance will be poor. This poor performance is not improved even if infinite training data are given.

The FSLL model extends the manifold's dimension to $|X|-1$ by introducing higher-order factors. The model manifold becomes $\mathcal{P}$ itself; thus, there is no more expansion. Therefore, we refer to the model as the *full-Span log-linear model*. The main interest of this letter is that as far as the problem size is $|X| \lesssim 2^{25}$, the FSLL model becomes a feasible and practical model.

For example, suppose that we construct a full-span model by adding hidden nodes into a Boltzmann machine having 20 visible nodes. The number of parameters of the Boltzmann machine is $E+n$, where E is the number of edges and n is the number of nodes. Therefore, it is not practical to construct the full-span model for 20 visible nodes because it requires $\approx 2^{20}$ edges.

The second advantage is that the FSLL model has no hyperparameters; therefore, no hyperparameter tuning is needed. For example, if we use a Boltzmann machine with hidden nodes that learns the true distribution with contrastive divergence methods, we need to determine hyperparameters such as the learning rate, mini-batch size, and the number of hidden and graphical structure of nodes. The FSLL model, however, automatically learns the training data without human participation.

5 Conclusion and Extension

Suppose that we let the FSLL model learn training data consisting of 20 binary variables. The dimension of the function space spanned by possible positive distributions is $2^{20}-1$. The FSLL model has $2^{20}-1$ parameters and can fit arbitrary positive distributions. The basis functions of the FSLL model have the following properties:

- Each basis function is a product of univariate functions.
- The basis functions take values 1 or -1.

The proposed learning algorithm exploited these properties and realized fast learning.

Our experiments demonstrated the following:

- The FSLL model could learn the training data with 20 binary variables within 1 minute with a laptop PC.
- The FSLL model successfully learned the true distribution underlying the training data; even higher-order terms that depend on three or more variables existed.
- The FSLL model constructed a more complex model to fit the true distribution as the training data increased; however, the learning time became longer.

In this letter, we have presented a basic version of the FSLL model; however, we can extend it as follows (Takabatake & Akaho, 2014, 2015):

- Introducing L_1 regularization (Andrew & Gao, 2007)
- Introducing hidden variables

Appendix: Proofs and Derivations of Equations

A.1 Proof of Theorem 1. For brevity and clarity of expression, we use predicate logic notation here. The statement "$\{f_i(X_0)g_j(X_1)\}(i \in I,\ j \in J)$ are linearly independent functions of (X_0, X_1)" is equivalent to the following proposition:

$$\forall x_0 \forall x_1 \left[\sum_{i,j} a_{ij} f_i(x_0) g_j(x_1) = 0 \right] \Rightarrow \forall i \forall j \left[a_{ij} = 0 \right].$$

This proposition is proved as follows:

$$\begin{aligned}
&\forall x_0 \forall x_1 \left[\sum_{i,j} a_{ij} f_i(x_0) g_j(x_1) = 0 \right] \\
&\Rightarrow \forall x_0 \left[\forall x_1 \left[\sum_j \left(\sum_i a_{ij} f_i(x_0) \right) g_j(x_1) = 0 \right] \right] \\
&\Rightarrow \forall x_0 \left[\forall j \left[\sum_i a_{ij} f_i(x_0) = 0 \right] \right] \quad \because \{g_j\} \text{ are linearly independent} \\
&\Rightarrow \forall i \forall j \left[a_{ij} = 0 \right]. \quad \because \{f_i\} \text{ are linearly independent.}
\end{aligned}$$

□

A.2 Derivation of Equation 2.6.

$$\begin{aligned}
\bar{\theta}_y(\theta) &= \partial_y \ln Z \\
&= \frac{1}{Z} \partial_y Z
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{Z} \sum_x \partial_y e^{l_\theta(x)} \\
&= \frac{1}{Z} \sum_x \Phi_y(x) e^{l_\theta(x)} \\
&= \sum_x p_\theta(x) \Phi_y(x) \\
&= \left\langle \Phi_y \right\rangle_{p_\theta} .
\end{aligned}
$$

A.3 Derivation of Equation 2.11.

$$
\begin{aligned}
\partial_y \bar{\theta}_y &= \partial_y \left\langle \Phi_y \right\rangle_{p_\theta} \\
&= \sum_x \Phi_y(x) \partial_y p_\theta(x) \\
&= \sum_x \Phi_y(x) p_\theta(x) \partial_y \ln p_\theta(x) \\
&= \sum_x \Phi_y(x) p_\theta(x) \partial_y (l_\theta(x) - \ln Z) \\
&= \sum_x \Phi_y(x) p_\theta(x) (\Phi_y(x) - \partial_y \ln Z) \\
&= \sum_x p_\theta(x) \Phi_y(x)^2 - \bar{\theta}_y \sum_x p_\theta(x) \Phi_y(x) \\
&= \left\langle \Phi_y^2 \right\rangle_{p_\theta} - \bar{\theta}_y^2 .
\end{aligned}
$$

A.4 Derivation of Equation 2.13.

$$
\begin{aligned}
1 &= \frac{\partial \bar{\theta}_y}{1 - \bar{\theta}_y^2} \quad \text{(by equation 2.12)} \\
&= \frac{1}{2} \left(\frac{\partial_y \bar{\theta}_y}{1 + \bar{\theta}_y} + \frac{\partial_y \bar{\theta}_y}{1 - \bar{\theta}_y} \right) \\
&= \frac{1}{2} \left(\partial_y \ln(1 + \bar{\theta}_y) - \partial_y \ln(1 - \bar{\theta}_y) \right) . \qquad \text{(A.1)}
\end{aligned}
$$

Integrating equation A.1, we obtain

$$
\begin{aligned}
\frac{1}{2} \ln \frac{1 + \bar{\theta}_y}{1 - \bar{\theta}_y} &= \int 1 d\theta_y \\
&= \theta_y - c, \qquad \text{(A.2)}
\end{aligned}
$$

where c is a constant of integration. Since the left side of equation A.2 equals $\tanh^{-1}(\bar{\theta}_y)$, we obtain the equation $\bar{\theta}_y = \tanh(\theta_y - c)$.

A.5 Derivation of Equation 2.16.

$$\begin{aligned}
\partial_y KL(p_d \| p_\theta) &= \partial_y \left(\langle \ln p_d \rangle_{p_d} - \langle \ln p_\theta \rangle_{p_d} \right) \\
&= -\partial_y \langle \ln p_\theta \rangle_{p_d} \\
&= -\partial_y \langle l_\theta - \ln Z \rangle_{p_d} \\
&= -\langle \partial_y l_\theta \rangle_{p_d} + \partial_y \ln Z \\
&= -\langle \Phi_y \rangle_{p_d} + \bar{\theta} \\
&= \bar{\theta}_y - \bar{d}.
\end{aligned}$$

A.6 Derivation of Equation 2.17.

$$\begin{aligned}
\int_{\theta_y^0}^{\theta_y} \bar{\theta}_y(u) - \bar{d}_y du &= \int_{\theta_y^0}^{\theta_y} \tanh(u-c)du - \bar{d}_y(\theta_y - \theta_y^0) \quad \text{(by equation 2.13)} \\
&= [\ln \cosh(u-c)]_{u=\theta_y^0}^{\theta_y} - \bar{d}_y(\tanh^{-1}\bar{\theta}_y - \tanh^{-1}\bar{\theta}_y^0) \\
&= \left[\frac{1}{2} \ln \cosh^2(u-c) \right]_{u=\theta_y^0}^{\theta_y} - \bar{d}_y \left(\frac{1}{2} \ln \frac{1+\bar{\theta}_y}{1-\bar{\theta}_y} - \frac{1}{2} \ln \frac{1+\bar{\theta}_y^0}{1-\bar{\theta}_y^0} \right) \\
&= \left[\frac{1}{2} \ln \frac{1}{1-\tanh^2(u-c)} \right]_{u=\theta_y^0}^{\theta_y} - \frac{\bar{d}_y}{2} \ln \frac{1+\bar{\theta}_y}{1-\bar{\theta}_y} \frac{1-\bar{\theta}_y^0}{1+\bar{\theta}_y^0} \\
&= \frac{1}{2} \left(\ln \frac{1}{1-\bar{\theta}_y^2} - \ln \frac{1}{1-(\bar{\theta}_y^0)^2} \right) - \frac{\bar{d}_y}{2} \ln \frac{1+\bar{\theta}_y}{1-\bar{\theta}_y} \frac{1-\bar{\theta}_y^0}{1+\bar{\theta}_y^0} \\
&= \frac{1}{2} \ln \frac{1+\bar{\theta}_y^0}{1+\bar{\theta}_y} \frac{1-\bar{\theta}_y^0}{1-\bar{\theta}_y} - \frac{\bar{d}_y}{2} \ln \frac{1+\bar{\theta}_y}{1-\bar{\theta}_y} \frac{1-\bar{\theta}_y^0}{1+\bar{\theta}_y^0} \\
&= -\frac{1+\bar{d}_y}{2} \ln(1+\bar{\theta}_y) - \frac{1-\bar{d}_y}{2} \ln(1-\bar{\theta}_y) \\
&\quad + \frac{1+\bar{d}_y}{2} \ln(1+\bar{\theta}_y^0) + \frac{1-\bar{d}_y}{2} \ln(1-\bar{\theta}_y^0) \\
&= \frac{1+\bar{d}_y}{2} \ln \frac{1+\bar{\theta}_y^0}{1+\bar{\theta}_y} + \frac{1-\bar{d}_y}{2} \ln \frac{1-\bar{\theta}_y^0}{1-\bar{\theta}_y}.
\end{aligned}$$

A.7 Derivation of Equation 2.19.

$$
\begin{aligned}
\Delta &= \frac{1+\bar{d}_y}{2}\ln\frac{1+\bar{\theta}_y^0}{1+\bar{d}_y} + \frac{1-\bar{d}_y}{2}\ln\frac{1-\bar{\theta}_y^0}{1-\bar{d}_y} + r_y \\
&= \frac{1+\bar{d}_y}{2}\left(-\ln\frac{1+\bar{d}_y}{1+\bar{\theta}_y^0}\right) + \frac{1-\bar{d}_y}{2}\left(-\ln\frac{1-\bar{d}_y}{1-\bar{\theta}_y^0}\right) + r_y \\
&\geq \frac{1+\bar{d}_y}{2}\left(1-\frac{1+\bar{d}_y}{1+\bar{\theta}_y^0}\right) + \frac{1-\bar{d}_y}{2}\left(1-\frac{1-\bar{d}_y}{1-\bar{\theta}_y^0}\right) + r_y \\
&\quad \because \frac{1+\bar{d}_y}{2} \geq 0, \quad \frac{1-\bar{d}_y}{2} \geq 0, \quad -\ln x \geq 1-x \\
&= \frac{1+\bar{d}_y}{2}\frac{\bar{\theta}_y^0-\bar{d}_y}{1+\bar{\theta}_y^0} - \frac{1-\bar{d}_y}{2}\frac{\bar{\theta}_y^0-\bar{d}_y}{1-\bar{\theta}_y^0} + r_y \\
&= \frac{\bar{\theta}_y^0-\bar{d}_y}{2}\left(\frac{1+\bar{d}_y}{1+\bar{\theta}_y^0} - \frac{1-\bar{d}_y}{1-\bar{\theta}_y^0}\right) + r_y \\
&= \frac{\bar{\theta}_y^0-\bar{d}_y}{2}\frac{(1+\bar{d}_y)(1-\bar{\theta}_y^0)-(1-\bar{d}_y)(1+\bar{\theta}_y^0)}{(1+\bar{\theta}_y^0)(1-\bar{\theta}_y^0)} + r_y \\
&= \frac{\bar{\theta}_y^0-\bar{d}_y}{2}\frac{2\bar{d}_y-2\bar{\theta}_y^0}{1-(\bar{\theta}_y^0)^2} + r_y \\
&= -\frac{(\bar{\theta}_y^0-\bar{d}_y)^2}{1-(\bar{\theta}_y^0)^2} + r_y \\
&= \underline{\Delta}.
\end{aligned}
$$

A.8 Proof of Theorem 2. Since B is a bounded closed set, $\{\theta^t\}$ has one or more accumulation point(s) in B.

As an assumption of a proof by contradiction, assume that a is an accumulation point of $\{\theta^t\}$ and the proposition

$$\exists y \exists b \in A_y(a), \quad f(b) < f(a)$$

holds. Let $\eta^j \in A_y(\theta^j)$ be the point such that $\eta_y^j = b_y$. Since $f(a) - f(b) > 0$, $f(\theta)$ is continuous at $\theta = b$, and a is an accumulation point of $\{\theta^t\}$, the proposition

$$\exists \theta_j, \quad f(\eta^j) < f(a)$$

holds ($\because$ in Figure 5, $f(\eta^j) < f(a)$ for sufficiently small δ).

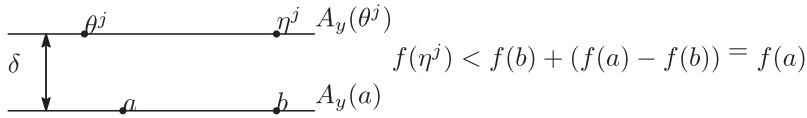

Figure 5: Existence of θ^j such that $f(\eta^j) < f(a)$.

Here, $\eta^j \in A_y(\theta^j)$, and the inequality

$$f(\theta^{j+1}) = \min_{\theta \in \cup_y A_y(\theta^j)} f(\theta) \leq f(\eta^j)$$

holds. Therefore, the inequality

$$f(\theta^{j+1}) \leq f(\eta^j) < f(a)$$

holds. Since $f(\theta^t)$ monotonically decreases as t increases, the proposition

$$\forall t \geq j+1, \quad f(\theta^t) \leq f(\eta^j) < f(a)$$

holds. Since $f(\theta)$ is continuous at $\theta = a$, no sub-sequence of $\{\theta^t\}$ can converge to a, that is, a is not an accumulation point of $\{\theta^t\}$. This fact contradicts the assumption we made at the beginning of this proof. □

A.9 Proof of Corollary 1. By theorem 2, any accumulation point of $\{\theta^t\}$ is an axis minimum of f; however, the axis minimum of f is unique; therefore, $\theta = a$ is a unique accumulation point of $\{\theta^t\}$, and $\theta^t \to a (t \to \infty)$. □

A.10 Proof of Theorem 3. Let us define the following symbols:

$$m_y(\theta^t, \infty) = \min_{\theta \in A_y(\theta^t)} f(\theta),$$

$$\arg m_y(\theta^t, \infty) = \{\theta | \theta \in A_y(\theta^t), f(\theta) = m_y(\theta^t, \infty)\}.$$

Figure 6 illustrates the sectional view of $f_N(\theta)$ along the line $A_y(\theta^t)$. As shown in the figure, $f_N(\theta)$ has a gap with depth r_y (see equation 2.5) at $\theta_y = 0$. Here, we can ignore the gap at $\theta_y = 0$ for sufficiently large N, that is, the proposition

$$\exists N' \forall N, \quad N > N' \Longrightarrow \arg m_y(\theta^t, N) = \arg m_y(\theta^t, \infty)$$

holds. Moreover, the proposition

$$\exists N' \forall N \forall y, \quad N > N' \Longrightarrow \arg m_y(\theta^t, N) = \arg m_y(\theta^t, \infty) \tag{A.3}$$

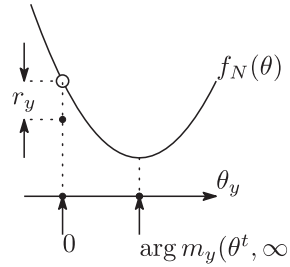

Figure 6: View of $f_N(\theta)$.

holds. By equation 2.23, the proposition

$$\exists N' \forall N, \quad N > N' \Longrightarrow \arg m(\theta^t, N) = \arg m(\theta^t, \infty)$$

holds. Here, we compare a sequence $\{\theta^t(N)\}$ with the cost function f_N and a sequence $\{\theta^t(\infty)\}$ with the cost function $f(= f_\infty)$. By corollary 1, $\lim_{t\to\infty} f(\theta^t(\infty)) = \min_\theta f(\theta)$, that is,

$$\forall \epsilon > 0 \exists t' \forall t, \quad t \geq t' \Longrightarrow f(\theta^t(\infty)) < \min_\theta f(\theta) + \epsilon \tag{A.4}$$

holds. Here, let $T(N)$ be the smallest t such that $m(\theta^{t+1}, N) \neq m(\theta^{t+1}, \infty)$. Then $\lim_{N\to\infty} T(N) = \infty$, that is,

$$\forall t' \exists N, \quad T(N) \geq t' \tag{A.5}$$

holds. By equations A.4 and A.5, the proposition

$$\forall \epsilon > 0 \exists N, \forall t, \quad t \geq T(N) \Longrightarrow f(\theta^t(\infty)) < \min_\theta f(\theta) + \epsilon \tag{A.6}$$

holds, and therefore the proposition

$$\forall \epsilon > 0 \exists N, \quad f(\theta^{T(N)}(\infty)) < \min_\theta f(\theta) + \epsilon \tag{A.7}$$

also holds. Here, by the definition of $T(N)$, $\theta^{T(N)}(\infty) = \theta^{T(N)}(N)$. Therefore, we can modify equation A.7 as

$$\forall \epsilon > 0 \exists N, \quad f(\theta^{T(N)}(N)) < \min_\theta f(\theta) + \epsilon. \tag{A.8}$$

Since $f(\theta^t(N))$ monotonically decreases as t grows,

$$\forall \epsilon > 0 \exists N \forall t, \quad t \geq T(N) \Longrightarrow f(\theta^t(N)) < \min_{\theta} f(\theta) + \epsilon.$$

Using the notation of lim, we obtain

$$\lim_{N \to \infty} \lim_{t \to \infty} f(\theta^t(N)) = \min_{\theta} f(\theta). \qquad \square$$

Acknowledgments

This research is supported by KAKENHI 17H01793.

References

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, *9*(1), 147–169. 10.1207/s15516709cog0901_7

Amari, S. (2016). *Information geometry and its applications*. Berlin: Springer.

Andrew, G., & Gao, J. (2007). Scalable training of L_1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning* (pp. 33–40). New York: ACM.

Beck, A. (2015). On the convergence of alternating minimization for convex programming with applications to iteratively reweighted least squares and decomposition schemes. *SIAM Journal on Optimization*, *25*(1), 185–209. 10.1137/13094829X

Dennis Jr., J. E., & Moré, J. J. (1977). Quasi-Newton methods, motivation and theory. *SIAM Review*, *19*(1), 46–89. 10.1137/1019005

Fino, B. J., & Algazi, V. R. (1976). Unified matrix treatment of the fast Walsh-Hadamard transform. *IEEE Transactions on Computers*, *25*(11), 1142–1146. 10.1109/TC.1976.1674569

Newman, M., & Barkema, G. (1999). *Monte Carlo methods in statistical physic*. New York: Oxford University Press.

Nocedal, J., & Wright, S. (2006). *Numerical optimization*. New York: Springer Science & Business Media.

Pratt, W. K., Andrews, H. C., & Kane, J. (1969). Hadamard transform image coding. In *Proceedings of the IEEE*, *57*(1), 58–68. 10.1109/PROC.1969.6869

Raff, E. (2017). JSAT: Java statistical analysis tool, a library for machine learning. *Journal of Machine Learning Research*, *18*(23), 1–5.

Rissanen, J. (2007). *Information and complexity in statistical modeling*. Berlin: Springer.

Sejnowski, T. J. (1986). Higher-order Boltzmann machines. In *AIP Conference Proceedings* (vol. 151, pp. 398–403). Melville, NY: American Institute for Physics. 10.1063/1.36246

Smith, W. W. (2010). *Handbook of real-time fast Fourier transforms*. Piscataway, NJ: IEEE.

Takabatake, K., & Akaho, S. (2014). Basis functions for fast learning of log-linear models. *IEICE Technical Report*, *114*(306), 307–312. (in Japanese)

Takabatake, K., & Akaho, S. (2015). Full-span log-linear model with L_1 regularization and its performance. *IEICE Technical Report*, *115*(323), 153–157. (in Japanese)

Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine Learning* (pp. 1064–1071). New York: ACM.

Received August 25, 2021; accepted January 22, 2022.