SN Computer Science (2020) 1:165
https://doi.org/10.1007/s42979-020-00170-7

ORIGINAL RESEARCH

Parallelized Training of Restricted Boltzmann Machines Using Markov-Chain Monte Carlo Methods

Pei Yang[1] · Srinivas Varadharajan[1] · Lucas A. Wilson[1] · Don D. Smith II[1] · John A. Lockman III[1] · Vineet Gundecha[1] · Quy Ta[1]

Received: 12 August 2019 / Accepted: 16 April 2020 / Published online: 12 May 2020

Abstract
Restricted Boltzmann machine (RBM) is a generative stochastic neural network that can be applied to collaborative filtering technique used by recommendation systems. Prediction accuracy of the RBM model is usually better than that of other models for recommendation systems. However, training the RBM model involves Markov-Chain Monte Carlo method, which is computationally expensive. In this paper, we have successfully applied distributed parallel training using Horovod framework to improve the training time of the RBM model. Our tests show that the distributed training approach of the RBM model has a good scaling efficiency. We also show that this approach effectively reduces the training time to little over 12 min on 64 CPU nodes compared to 5 h on a single CPU node. This will make RBM models more practically applicable in recommendation systems.

Keywords Restricted Boltzmann machine · Distributed training · Collaborative filtering · Recommendation systems

Introduction

The restricted Boltzmann machines (RBM) are generative stochastic neural networks that were first proposed in 1980s by Smolensky et al. [11] and intensively studied by Hinton et al. [6, 7, 9]. Theoretically, RBM has one visible layer and several hidden layers (one hidden layer in most cases), with mutual connections between neurons in different layers while connections between neurons within the same layer are prohibited, as is shown in Fig. 3. Connections between neurons are determined in a way such that the "energy" for the system—consisting of all these neurons—is minimal. As such, RBM is an energy-based bidirectional graphical model, whose principles and topologies are quite different from those of other neural networks such as multilayer perceptrons (MLP), convolutional neural networks (CNN), recurrent neural networks (RNN), etc.

One of the popular applications of RBM is collaborative filtering for recommendation system [8], where the algorithm needs to predict users' interest levels for products that they have not purchased based on the observed ratings for other products. RBM model outperforms other models for collaborative filtering (e.g., singular value decomposition (SVD) model [12]) by predicting with better accuracy [8]. Considering large data sets with number of users and products (typically more than 100,000), the number of ratings involved is at the scale of 10^{12} or even bigger, which also requires a large memory space to train RBM models. Also, the training algorithm involves Markov Chain Monte Carlo (MCMC) step, which is very computationally expensive. Hence, distributed training is a necessity in order to speed up the training process and practically leverage RBM models for recommendation problems in e-commerce, retails, online entertainment, etc.

One efficient algorithm to train the RBM is the contrastive divergence (CD) algorithm initially proposed by Hinton et al. [1]. The basic idea behind CD algorithm is to approximately draw samples from a joint distribution via sampling from a Markov chain with up to a limited number of steps. CD algorithm has been shown to work well even with just a few steps

This article is part of the topical collection "Software Challenges to Exascale Computing" guest edited by Amit Majumdar and Ritu Arora.

✉ Lucas A. Wilson
lucaswilson@acm.org

[1] Dell Technologies HPC & AI Innovation Lab, Dell Technologies, Austin, TX, USA

of the Markov chain [8]. However, for a large-scale training data, it still takes a long time for the RBM model to converge to a good solution. Hence, it is necessary to explore parallel training techniques to reduce the time to solution. This would make RBMs to be practically applied for collaborative filtering in recommendation systems. This would also be useful in recommendation systems where there is a need to retrain the model frequently, and where training is constrained by the available time. In this paper, we describe a parallelized training approach using the Horovod [10] framework to significantly scale-up RBM models with large-scale data sets for collaborative filtering in recommendation systems.

The paper is organized as follows. In Sect. 3, the model architecture and mathematical principles of RBM are presented. Section 4 details the learning algorithm for RBM. Sections 5 and 6 describe how to perform parallelized training of RBM model and how to make predictions with a trained RBM model. In Sect. 7, some experimental results with the MovieLens data set [5] are presented. In the end, Sect. 8 presents our conclusions.

Restricted Boltzmann Machine Versus Singular Value Decomposition

Alternative approaches to RBM have been used in the past for the purpose of collaborative filtering, and require less computational processing. While RBMs require more training effort, they can produce higher quality recommendations than other traditionally used approaches. One such approach is singular value decomposition (SVD) [14].

Before testing parallelized training of RBM model, we first train an RBM model with a small subset of the MovieLens data and compare its performance with that of a model produced using the SVD method. The small data set has 84,313 ratings for 9557 movies rated by 248 users. In this test, the first 30 ratings for each user is held from the data as test set and the remaining part is used as training set.

For SVD method, the basic idea is to keep only a portion of singular values of the rating matrix

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{V}^T \tag{1}$$

and reconstruct $\boldsymbol{R}$ via

$$\boldsymbol{R}' = \boldsymbol{U}\boldsymbol{\Lambda}'\boldsymbol{V}^T. \tag{2}$$

Here Λ contains all the singular values of R and Λ' is obtained by truncating Λ and keeping only the first q leading singular values. Then, the predicted rating for movie m' by user n' is defined as

$$r'_{n'm'} = \boldsymbol{R}'_{n'm'}. \tag{3}$$

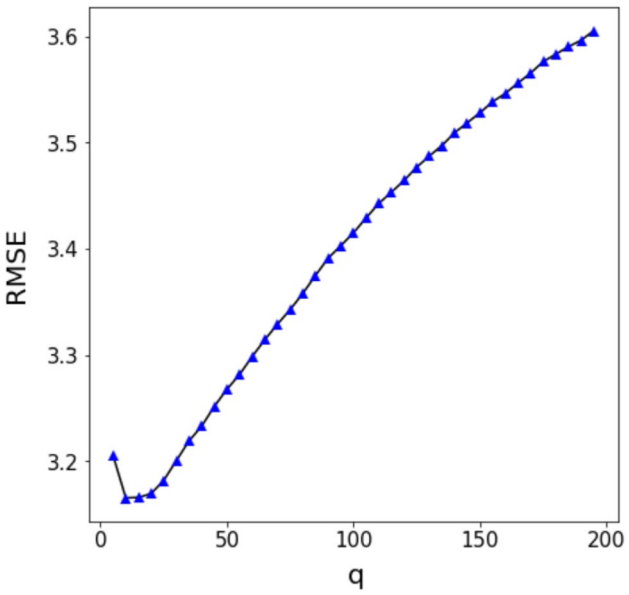

Fig. 1 RMSE for SVD models with different q values

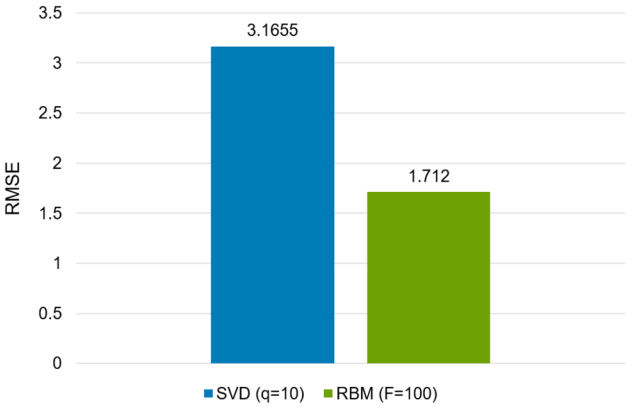

Fig. 2 Prediction accuracy comparison (SVD vs. RBM)

Readers may refer to [12] for more details on SVD method and its variations.

Since q is a hyper-parameter for SVD method, we tested SVD models with different values of q and computed their performance metric RMSE (Rooted Mean Square Error). The result is shown in Fig. 1. We can observe a distinct inflection point in the RMSE around $q = 10$, after which normalized error continues to asymptotically increase.

The comparison of prediction accuracy for SVD with $q = 10$ and RBM is shown in Fig. 2. The RBM model has 100 neurons in hidden layer ($F = 100$) and is trained with global batch size of 50 for 100 epochs with a learning rate of 0.001 on 2 processes in a single compute node. As is shown in Fig. 2, RMSE value for SVD model with the optimal q value ($q = 10$) is still way larger than that for RBM model.

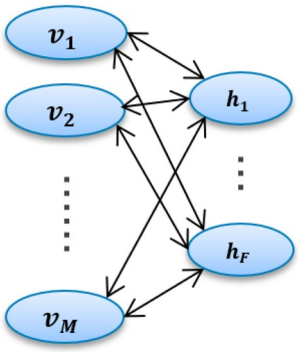

Fig. 3 RBM architecture

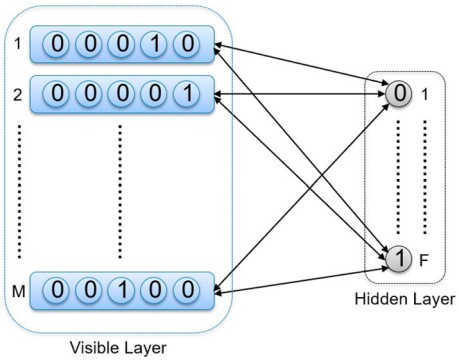

Fig. 4 RBM architecture (5-way softmax in visible layer)

In reality, RBM model produces more accurate predictions than SVD model in most cases.

While prediction accuracy of RBM model is usually higher than that of SVD model, training an RBM model could be computationally slow. This motivated us to explore on parallelized training of RBM model.

Restricted Boltzmman Machine for Collaborative Filtering

Usually, a RBM is a bidirectional network with one visible layer and one hidden layer. The neurons in visible and hidden layers are mutually connected, while connections between neurons within the same layers are restricted, as is shown in Fig. 3. If we try to predict the users' ratings for some products using the collaborative filtering, the visible layer represents the ratings in a 5-way 0's and 1's as shown in Fig. 4.

Suppose, we have N users and M products. The N users have rated a portion of the M products, with rating values between 1 and K ($K = 5$ for the case in Fig. 4). For example, if a visible neuron is in the state of [0, 0, 0, 1, 0], it suggests that the user has provided a rating value of 4 for this product. The visible layer has M neurons, with each corresponding to one of the M products. The states of hidden neurons are binary (0 or 1).

RBM is an energy-based model. For the system $(\boldsymbol{V}, \boldsymbol{H})$ $(\boldsymbol{V} = \{v_i^k\}, \boldsymbol{H} = \{h_j\})$, the "energy" is defined as

$$E(\boldsymbol{V}, \boldsymbol{H}) = -\sum_{i=1}^{m}\sum_{j=1}^{F}\sum_{k=1}^{K} v_i^k W_{ij}^k h_j - \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k b_i^k - \sum_{j=1}^{F} h_j c_j \tag{4}$$

where W_{ij}^k models the interactions between visible and hidden layers, while b_i^k and c_j are bias terms for visible and hidden layers and F denotes the number of neurons in the hidden layer. The joint probability distribution is

$$p(\boldsymbol{V}, \boldsymbol{H}) = \frac{\exp(-E(\boldsymbol{V}, \boldsymbol{H}))}{Z} \tag{5}$$

where $Z = \sum_{\boldsymbol{V}}\sum_{\boldsymbol{H}} p(\boldsymbol{V}, \boldsymbol{H})$ is the normalization factor. It can be shown that [8, 13]

$$p(v_i^k = 1|\boldsymbol{H}) = \frac{\exp(b_i^k + \sum_{j=1}^{F} h_j W_{ij}^k)}{\sum_{l=1}^{K} \exp(b_i^l + \sum_{j=1}^{F} h_j W_{ij}^l)} \tag{6}$$

and

$$p(h_j = 1|\boldsymbol{V}) = \sum(c_j + \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k W_{ij}^k) \tag{7}$$

where $\sum(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function. Also, in [13] it was shown that

$$p(\boldsymbol{V}; \Theta) = \frac{f(\boldsymbol{V}; \Theta)}{Z} \tag{8}$$

where $\Theta = (W_{ij}^k, b_i^k, c_j)$ are the parameters for RBM model, and

$$\begin{aligned} f(\boldsymbol{V}; \Theta) &= \sum_{h_1, h_2, \ldots, h_F} \exp(-E(\boldsymbol{V}, \boldsymbol{H})) \\ &= \sum_{j'=1}^{F}\sum_{h_{j'}=0}^{1} \exp\Big(\sum_{i=1}^{m}\sum_{j=1}^{F}\sum_{k=1}^{K} v_i^k W_{ij}^k h_j \\ &\quad + \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k b_i^k + \sum_{j=1}^{F} h_j c_j\Big) \end{aligned} \tag{9}$$

where $\boldsymbol{H} = (h_1, h_2, \ldots, h_F)$. A detailed description of RBM can be found in [13].

Learning Algorithm for RBM

Suppose, we have observed data $\{\boldsymbol{V}_n\}_{n=1}^N$, then the likelihood with respect to such data is

$$\begin{aligned} L(\{\boldsymbol{V}_n\}_{n=1}^N, \Theta) &= \Pi_{n=1}^N p(\boldsymbol{V}_n;\Theta) \\ &= \Pi_{n=1}^N \frac{f(\boldsymbol{V}_n;\Theta)}{Z(\Theta)}. \end{aligned} \tag{10}$$

To maximize $L(\{\boldsymbol{V}_n\}_{n=1}^N, \Theta)$ is equivalent to minimizing the following objective function

$$\begin{aligned} G(\{\boldsymbol{V}_n\}_{n=1}^N;\Theta) &= -\frac{1}{N}\log(L(\{\boldsymbol{V}_n\}_{n=1}^N;\Theta)) \\ &= \log(Z(\Theta)) - \frac{1}{N}\sum_{n=1}^N \log(f(\boldsymbol{V}_n;\Theta)). \end{aligned} \tag{11}$$

Learning via Gradient Descent

The gradient of G with respect to Θ is

$$\begin{aligned} \frac{\partial G}{\partial \Theta} &= \frac{\partial \log(Z(\Theta))}{\partial \Theta} - \frac{1}{N}\sum_{n=1}^N \frac{\partial \log(f(\boldsymbol{V}_n;\Theta))}{\partial \Theta} \\ &= \frac{\partial \log(Z(\Theta))}{\partial \Theta} - \left\langle \frac{\partial \log(f(\boldsymbol{V};\Theta))}{\partial \Theta} \right\rangle_{\boldsymbol{V}\in\{\boldsymbol{V}_n\}_{i=n}^N}. \end{aligned} \tag{12}$$

The second term is the expectation of $\log(f(\boldsymbol{V};\Theta))$ with respect to observed data $\{\boldsymbol{V}_n\}_{n=1}^N$. For the first term, we have

$$\begin{aligned} \frac{\partial \log(Z(\Theta))}{\partial \Theta} &= \frac{1}{Z(\Theta)}\frac{\partial Z(\Theta)}{\partial \Theta} \\ &= \frac{1}{Z(\Theta)}\sum_{\boldsymbol{V}} \frac{\partial f(\boldsymbol{V},\Theta)}{\partial \Theta} \\ &= \sum_{\boldsymbol{V}} \frac{f(\boldsymbol{V};\Theta)}{Z(\Theta)}\frac{1}{f(\boldsymbol{V},\Theta)}\frac{\partial f(\boldsymbol{V},\Theta)}{\partial \Theta} \\ &= \sum_{\boldsymbol{V}} p(\boldsymbol{V};\Theta)\frac{\partial \log f(\boldsymbol{V},\Theta)}{\partial \Theta} \\ &= \left\langle \frac{\partial \log f(\boldsymbol{V};\Theta)}{\partial \Theta} \right\rangle_{p(\boldsymbol{V};\Theta)}. \end{aligned} \tag{13}$$

That is, $\frac{\partial \log(Z(\Theta))}{\partial \Theta}$ is the expectation of $\frac{\partial \log f(\boldsymbol{V},\Theta)}{\partial \Theta}$ with respect to distribution $p(\boldsymbol{V};\Theta)$. From Eq. 9, we have

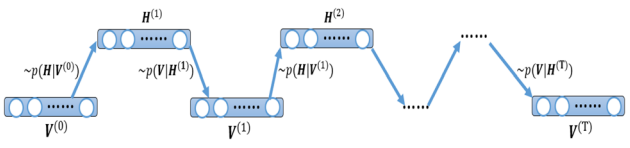

Fig. 5 Gibbs sampling

$$\begin{aligned} &\frac{\partial \log f(\boldsymbol{V};\Theta)}{\partial W_{ij}^k} \\ &= \frac{1}{f}\frac{\partial f(\boldsymbol{V};\Theta)}{\partial W_{ij}^k} \\ &= \frac{1}{f}\sum_{j'=1}^F \sum_{h_{j'}=0}^1 v_i^k h_j exp(\sum_{i=1}^m \sum_{j=1}^F \sum_{k=1}^K v_i^k W_{ij}^k h_j \\ &\quad + \sum_{i=1}^m \sum_{k=1}^K v_i^k b_i^k + \sum_{j=1}^F h_j c_j) \\ &= \frac{1}{f} v_i^k h_j f \\ &= v_i^k h_j. \end{aligned} \tag{14}$$

Similarly, it can be shown that

$$\frac{\partial \log f(\boldsymbol{V};\Theta)}{\partial b_i^k} = v_i^k, \tag{15}$$

$$\frac{\partial \log f(\boldsymbol{V};\Theta)}{\partial c_j} = h_j. \tag{16}$$

Then, the gradient descent learning algorithm for RBM is

$$W_{ij}^k \leftarrow W_{ij}^k + \eta(\langle v_i^k h_j\rangle_{\text{data}} - \langle v_i^k h_j\rangle_{model}), \tag{17}$$

$$b_i^k \leftarrow b_i^k + \eta(\langle v_i^k\rangle_{\text{data}} - \langle v_i^k\rangle_{model}), \tag{18}$$

$$c_j \leftarrow c_j + \eta(\langle h_j\rangle_{\text{data}} - \langle h_j\rangle_{model})) \tag{19}$$

where η is learning rate and $\langle\cdot\rangle_{\text{data}}$, $\langle\cdot\rangle_{model}$ are the expectations corresponding to observed data and the true probability distribution from RBM model, respectively.

Contrastive Divergence Algorithm with MCMC

Usually, $p(\boldsymbol{V};\Theta)$ is intractable since $Z(\Theta)$ is unknown. Hence, it is infeasible to analytically compute $\langle v_i^k h_j\rangle_{model}$, $\langle v_i^k\rangle_{model}$ and $\langle h_j\rangle_{model}$ in the learning algorithm (17), (18) and (19). In practice, Monte Carlo method is applied to compute these expectations approximately, which uses sample mean from a large-size sampling set for the joint distribution $p(\boldsymbol{V},\boldsymbol{H})$ to estimate the theoretical expectations. Since $p(\boldsymbol{V},\boldsymbol{H})$ is also unknown, it is also infeasible to draw samples from it directly.

To resolve this difficulty, Hinton et al. [6] proposed the contrastive divergence algorithm which utilizes Gibbs sampling technique. It is a MCMC algorithm, to draw samples that asymptotically follow the joint distribution $p(\boldsymbol{V},\boldsymbol{H})$.

Figure 5 illustrates the Gibbs sampling algorithm. The algorithm generates a Markov chain with known conditional probabilities $p(\boldsymbol{H}|\boldsymbol{V})$ and $p(\boldsymbol{V}|\boldsymbol{H})$. When the chain is long enough, samples at the end of the chain will be theoretically close enough to true samples drawn from the unknown joint distribution $p(\boldsymbol{V},\boldsymbol{H})$. In reality, only a few Gibbs steps are enough to generate qualified samples that are needed to estimate the expectations in the learning algorithm (17), (18) and (19). Readers may refer to [4] for more details on MCMC and Gibbs sampling.

Contrastive Divergence Algorithm for Training RBM

The contrastive divergence learning algorithm for training RBM via T-step Gibbs sampling is summarized in Eqs. (20), (21) and (22).

$$W_{ij}^k \leftarrow W_{ij}^k + \eta(\langle v_i^k h_j\rangle_{\text{data}} - \langle v_i^k h_j\rangle_{\text{T}-stepGibbssamples}), \tag{20}$$

$$b_i^k \leftarrow b_i^k + \eta(\langle v_i^k\rangle_{\text{data}} - \langle v_i^k\rangle_{\text{T}-stepGibbssamples}), \tag{21}$$

$$c_j \leftarrow c_j + \eta(\langle h_j\rangle_{\text{data}} - \langle h_j\rangle_{\text{T}-stepGibbssamples}). \tag{22}$$

Parallelized Training

The learning algorithms (20), (21) and (22) are parallel by nature. Suppose, we have a batch of training data $\{(v_m)_i^k, (h_m)_j\}_{m=1}^N$ where $(v_m)_i^k$ and $(h_m)_j$ are v_i^k and h_j for the m^{th} data point in the training set. If N can be evenly divided into P parts with $N = Pn$, then

$$\begin{aligned}
\langle v_i^k h_j\rangle_{\text{data}} &= \frac{1}{N}\sum_{m=1}^{N}(v_m)_i^k(h_m)_j \\
&= \frac{1}{Pn}\sum_{m=1}^{Pn}(v_m)_i^k(h_m)_j \\
&= \frac{1}{P}[\frac{1}{n}\sum_{m=1}^{n}(v_m)_i^k(h_m)_j \\
&\quad + \frac{1}{n}\sum_{m=n+1}^{2n}(v_m)_i^k(h_m)_j + \cdots \\
&\quad + \frac{1}{n}\sum_{m=(P-1)n+1}^{Pn}(v_m)_i^k(h_m)_j)].
\end{aligned}$$

Similarly, it can be shown that other formulas for computing expectations in algorithms (20), (21) and (22) are parallel with respect to training data. So, we can distribute the computations in the algorithms over P processes, as is illustrated in Fig. 6.

We are performing distributed training of the RBM with the Horovod framework developed by Uber [10]. Horovod uses a disrtibuted optimizer strategy which wraps standard `tf.Optimizer`. This wrapper then uses the MPI allreduce or allgather operation (based on whether encoding is as dense tensors or sparse `IndexSlice`, see [2]) to accumulate gradient values before applying gradients to model weights. In essence, Horovod is increasing the effective batch size, and performing a lock-step backpropagation with the accumulated gradients from all MPI processes.

Inference

After an RBM model is trained (i.e., W_{ij}^k, b_i^k and c_j have been learned from training data), we can make prediction for a user's potential rating for a given item via $p(\boldsymbol{V};\boldsymbol{\Theta})$. Let $\boldsymbol{V}^{\text{obs}} = \{(v_m)_i^k\}_{i\in\mathscr{I},k\in\mathscr{K}}$ be the observed ratings for user m, where $\mathscr{I}, \mathscr{K}$ are sets for indices i, k for which a rating of k for item i is observed (for example, if a rating 4 is observed for item 221, then 4 is an element in $\mathscr{K}$ and 221 is an element in $\mathscr{I}$). Given $\boldsymbol{V}^{\text{obs}}$, the probability that a user will rate item i' with score k' is:

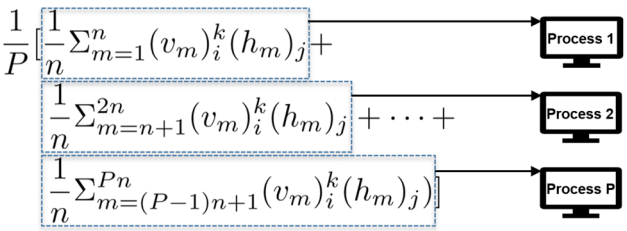

Fig. 6 Parallel computing of gradients

$$
\begin{aligned}
&p((v)_{i'}^{k'} = 1|\boldsymbol{V}^{\text{obs}}) \\
&= \frac{1}{p(\boldsymbol{V}^{\text{obs}})} p(\{(v)_{i'}^{k'}\} \bigcup \boldsymbol{V}^{\text{obs}}) \\
&\propto p(\{(v)_{i'}^{k'}\} \bigcup \boldsymbol{V}^{\text{obs}}) \\
&= \frac{1}{Z} \sum_{\boldsymbol{H}} \exp[\sum_{j=1}^{F} \sum_{i \in \{i'\} \bigcup \mathscr{I}} \sum_{k \in \{k'\} \bigcup \mathscr{K}} v_i^k W_{ij}^k h_j \\
&\quad + \sum_{i \in \{i'\} \bigcup \mathscr{I}} \sum_{k \in \{k'\} \bigcup \mathscr{K}} v_i^k b_i^k \\
&\quad + \sum_{j=1}^{F} h_j c_j] \\
&\propto \sum_{\boldsymbol{H}} \exp[\sum_{i \in \mathscr{I}} \sum_{k \in \mathscr{K}} + v_{i'}^{k'} b_{i'}^{k'}] \\
&\Pi_{j=1}^{F} \exp[\sum_{i \in \mathscr{I}} \sum_{k \in \mathscr{K}} v_i^k W_{ij}^k h_j + v_{i'}^{k'} W_{i'j}^{k'} h_j + h_j c_j] \\
&= \exp[v_{i'}^{k'} b_{i'}^{k'}] \\
&\prod_{j=1}^{F} \sum_{h_j=0}^{1} \exp[\sum_{i \in \mathscr{I}} \sum_{k \in \mathscr{K}} v_i^k W_{ij}^k h_j + v_{i'}^{k'} W_{i'j}^{k'} h_j + h_j c_j] \\
&= \exp[v_{i'}^{k'} b_{i'}^{k'}] \\
&\prod_{j=1}^{F} (1 + \exp[g(\{v_i^k\}_{i \in \mathscr{I}, k \in \mathscr{K}}; \{W_{ij}^k\}) + v_{i'}^{k'} W_{i'j}^{k'} + c_j]) \\
&= S(k'; i', \boldsymbol{V}^{\text{obs}})
\end{aligned}
$$

where $g(\{v_i^k\}_{i \in \mathscr{I}, k \in \mathscr{K}}; \{W_{ij}^k\}) = \sum_{i \in \mathscr{I}} \sum_{k \in \mathscr{K}} v_i^k W_{ij}^k$. Then the predicted rating that user m will give to item i' is the one with the highest S value, i.e.,

$$k_0 = \text{argmax}(\{S(k'; i', \boldsymbol{V}^{\text{obs}})\}_{k' \in \{1, \ldots, K\}}). \tag{23}$$

Readers can refer to [8] for more details about making predictions with RBM model.

Experiments

In this section, we test parallelized training of RBM model with the MovieLens data [5]. The data set has 27,753,444 ratings for $M = 53{,}889$ movies by $N = 283{,}228$ users. A piece of this data is shown in Table 1.

The rating matrix $\boldsymbol{R} \in \mathbf{R}^{N \times M}$ is defined as

$$\boldsymbol{R} = (r_{nm})_{n \in \{1, \ldots, N\}; m \in \{1, \ldots, M\}} \tag{24}$$

with r_{nm} being the rating score of user n for movie m. If r_{nm} is not observed yet, then, we let $r_{nm} = 0$ in the rating matrix.

Table 1 MovieLens data samples

UserId	MovieId	Rating
1	1	4.0
1	3	4.0
1	6	4.0
1	47	5.0
1	50	5.0

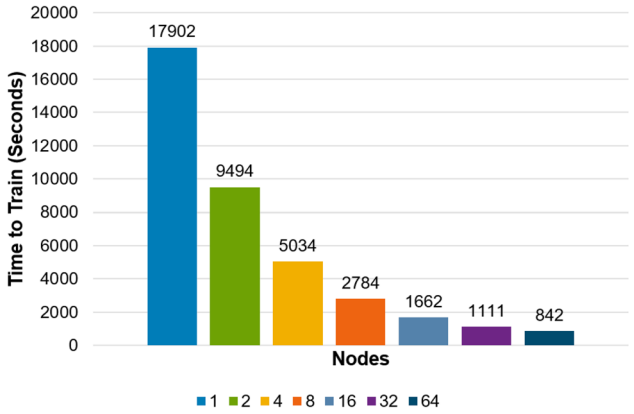

Fig. 7 Time to solution (strong scaling)

Parallelized Training of RBM Model

The data we used to test parallelized training of RBM model are obtained by selecting observed ratings from users who have rated at least 100 different movies. The selected data set contains 21,595,144 ratings for 53,324 movies by 68,342 users. Similarly, 30 of the ratings from each user is held as test data. The tests were run on the Zenith supercomputer at Dell EMC HPC & AI Innovation Lab [3].

Strong Scaling

To test the performance of strong scaling for RBM model, we fix the global batch size to be 512. Then, we train the model for one epoch with $1, 2, 4, \ldots, 64$ nodes with 1 process per node. This means that each MPI process will have a local batch size of $512, 256, 128, \ldots, 8$, respectively for each test. Time-to-train and scaled speedup are shown in Figs. 7 and 8, respectively.

As can be seen from Figs. 7 and 8, parallelizing the training scaled close to ideal out to 8 nodes (2^3), and began to taper off after that. This is due to the nature of strong scaling. Since strong scaling keeps the total problem size fixed (in

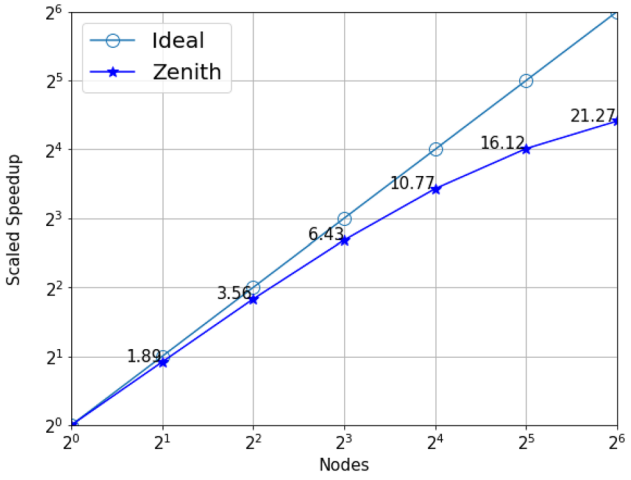

Fig. 8 Scaled speedup for strong scaling (1 epoch)

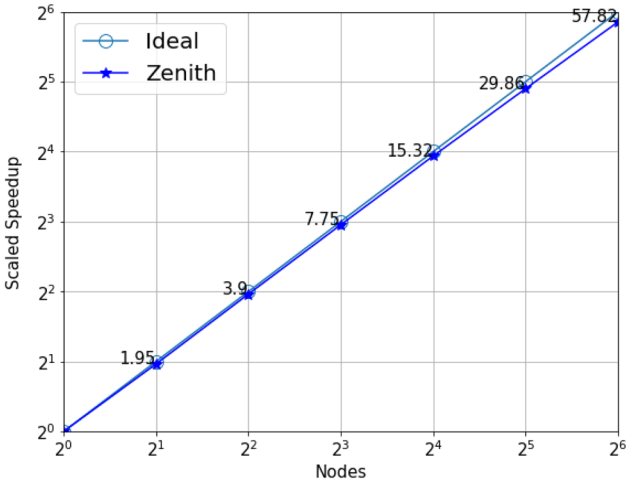

Fig. 10 Scaled speedup for weak scaling (1 epoch)

this case, the global batch size, which is fixed at 512 recommendations), larger process counts are dramatically reducing the number of recommendations processed per MPI process. At 512 nodes, each node is processing only 8 recommendations before global communication occurs, meaning the ratio of computation to communication is too low to maintain efficient parallel performance.

Weak Scaling

For weak scaling test, the batch size for each node is fixed to 100 recommendations. We run the test for one epoch for $1, 2, 4, \ldots, 64$ nodes. This means that the global batch (for the entire run) is $100, 200, 400, \ldots, 6400$ recommendations, respectively. Time-to-train and scaled speedup are shown in Figs. 9 and 10 respectively.

Time to train on 1 node is slightly lower for our weak scaling tests than for our strong scaling tests (see Figs. 7 and 8, respectively). This is due to the larger batch size with the strong scaling test on one node. Scaling is extremely efficient for the weak scaling case, staying very close to ideal performance (linear scaling, see Fig. 10). While batch size can have an effect on resulting model accuracy, this effect is data set dependent and many models can be trained with very large batches [2].

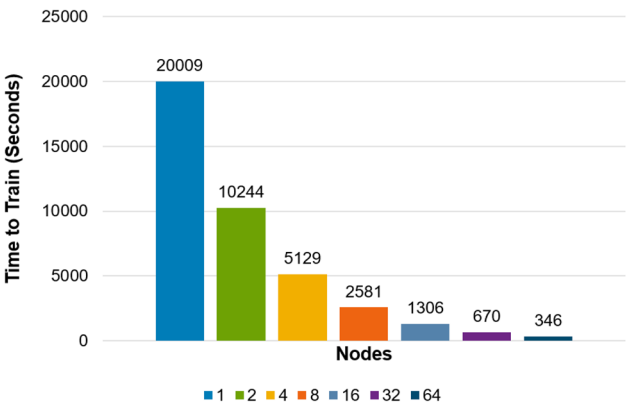

Fig. 9 Time to solution (weak scaling)

Prediction Accuracy

We trained an RBM model with global batch size 512 over 8 processes for 100 epochs. The model has 100 neurons in the hidden layer. The trained model was then applied to make predictions for 10, 963 ratings in the test data set. The RMSE value is about 1.62. In future work, we will train RBM models with larger global batch size and test the prediction accuracy for them.

In order to evaluate whether the accuracy of the resulting model is affected by the parallelization process, we also trained RBM model with global batch size 128 with multiple processes ranging from 1 to 8 for 100 epochs. The network has 100 neurons in the hidden layer. The data set used was a subset of 100,000 recommendations extracted from the full movie lens data. We chose not to use the full data set as the time required to fully train the network using a single process would have been prohibitively high. For 100 epochs, the time to train the single-process model using the full data set would have been approximately 20 days (assuming 17,902 seconds per epoch—see Fig. 7).

The results shown in Fig. 11 clearly indicate that even as we parallelize the RBM model the resulting model accuracy does not exhibit significant variation. RMSE after 100 epochs for single process non-distributed RBM model is 1.27 where as RMSE for 8 process distributed

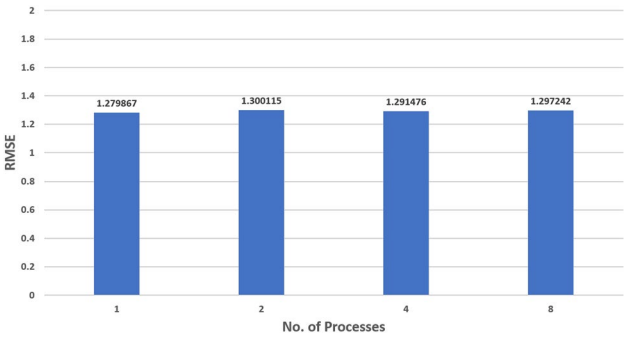

Fig. 11 RMSE comparison for multiple processes (100 epochs)

RBM model is 1.29, with a standard deviation across all four runs of $\sigma = 0.007757151$.

Conclusion and Future Work

In this paper, we studied the principles of RBM model and parallelized training with Horovod framework for it. As is shown in the paper, parallelized training can significantly shorten the training time. Only in this way, RBM models can be practically applied for collaborative filtering in recommendation systems.

Experiments using our technique to train a Restricted Boltzmann Machine with the MovieLens data set showed that both strong and weak scaling could be maintained out to 64 compute nodes while producing quality models in accordance with the scale of the data set used. Future work will focus on training at greater scale using larger data sets.

Going forward, we also intend to explore additional data sets in order to show better generality of the approach, as well as evaluate the performance on other hardware architectures, such as GPUs and special-purpose neural network chips.

Source Code Available

The code for this research is available on GitHub at: https://github.com/dellemc-hpc-ai/rbm-recommendation.

Compliance with Ethical Standards

Conflicts of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Carreira-Perpiñán MÁ, Hinton GE. On contrastive divergence learning. In: AISTATS 2005.
2. Cavdar D, Codreanu V, Karakus C, Lockman JA, Podareanu D, Saletore V, Sergeev A, Smith DD, Suthichai V, Ta Q, Varadharajan S. Densifying assumed-sparse tensors. In: International conference on high performance computing. Cham: Springer; 2019. p. 23–39. https://doi.org/10.1007/978-3-030-20656-7_2.
3. Dell EMC: HPC & AI Innovation Lab. 2019. https://www.dellemc.com/en-us/solutions/high-performance-computing/HPC-AI-Innovation-Lab.htm. Accessed 2 Nov 2019.
4. Gilks WR, Richardson S, Spiegelhalter D, editors. Markov Chain Monte Carlo in practice. New York: Chapman and Hall/CRC; 1996. https://doi.org/10.1201/b14835.
5. Harper FM, Konstan JA. The MovieLens datasets: history and context. ACM Trans Interact Intell Syst. 2015;5(4):19.
6. Hinton G. 2017. https://www.cs.toronto.edu/~hinton/csc321/readings/boltz321.pdf. Accessed 1 Nov 2019.
7. Hinton GE, Sejnowski TJ. Learning and relearning in Boltzmann machines. In: Rumelhart DE, McClelland JL, editors. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. Cambridge: MIT Press; 1986. p. 282–317.
8. Salakhutdinov R, Mnih A, Hinton G. Restricted Boltzmann machine for collaborative filtering. In: Proceedings of the 24th international conference on machine learning 2007.
9. Salakhutdinov R, Hinton G. Deep Boltzmann machines. In: van Dyk D, Welling M, editors. Proceedings of the twelth international conference on artificial intelligence and statistics, proceedings of machine learning research, vol. 5. PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA; 2009. pp. 448–455. http://proceedings.mlr.press/v5/salakhutdinov09a.html.
10. Sergeev A, Balso MD. Horovod: fast and easy distributed deep learning in TensorFlow. arXiv preprint arXiv:1802.05799 2018.
11. Smolensky P. Parallel distributed processing: explorations in the microstructure of cognition. In: Information processing in dynamical systems: foundations of harmony theory, vol. 1. Cambridge: MIT Press; 1986. pp. 194–281. http://dl.acm.org/citation.cfm?id=104279.104290.
12. Xian Z, Li Q, Li G, Li L. New collaborative filtering algorithms based on SVD++ and differential privacy. Math Probl Eng. 2017;2017:1–14. https://doi.org/10.1155/2017/1975719.
13. Yu H. A gentle tutorial on restricted Boltzmann machine and contrastive divergence 2017. https://doi.org/10.13140/RG.2.2.26119.60326.
14. Zhang S, Wang W, Ford J, Makedon F, Pearlman J. Using singular value decomposition approximation for collaborative filtering. In: Seventh IEEE international conference on e-commerce technology (CEC'05). IEEE; 2005. pp. 257–264.