A Distributed Coordinate Descent Algorithm for Learning Factorization Machine

Kankan Zhao[1,2], Jing Zhang[1,2], Liangfu Zhang[1,2], Cuiping Li[1,2(✉)], and Hong Chen[1,2]

[1] Key Laboratory of Data Engineering and Knowledge Engineering, Beijing, China
[2] School of Information, Renmin University of China, Beijing, China
{zhaokankan,zhang-jing,liangfu_zhang,licuiping,chong}@ruc.edu.cn

Abstract. Although much effort has been made to implement Factorization Machine (FM) on distributed frameworks, most of them achieve bad model performance or low efficiency. In this paper, we propose a new distributed block coordinate descent algorithm to learn FM. In addition, a distributed pre-computation mechanism incorporated with an optimized Parameter Server framework is designed to avoid the massive repetitive calculations and further reduce the communication cost. Systematically, we evaluate the proposed distributed algorithm on three different genres of datasets for prediction. The experimental results show that the proposed algorithm achieves significantly better performance (3.8%–6.0% RMSE) than the state-of-the-art baselines, and also achieves a 4.6–12.3× speedup when reaching a comparable performance.

Keywords: FM · Block coordinate descent · Distributed framework · Pre-computation

1 Introduction

Although there exists some research on adapting FM to the distributed frameworks, the problem remains largely unsolved. As we know, most of them implemented the (stochastic) gradient descent (SGD) to optimize the FM model, which is limited by the appropriate learning rate. Especially when the dataset and model space are quite large, an inappropriate learning rate may waste a lot of time in searching the optimal solution. Although some solutions can adjust the learning rate adaptively, the slow convergence rate is intolerable. Comparing with gradient optimization, coordinate descent (CD) can avoid setting learning rate, which makes CD converge faster and perform better in convex model. Thus, a distributed algorithm for solving FM by the CD algorithm is worth studying.

In this paper, we propose a novel CD algorithm under the Parameter Server (PS) framework to learn FM model. It divides all the parameters into blocks by the specific blocking scheme, and then updates one block at a time when maintaining the other blocks unchanged. We call this method as Block Coordinate Descent (BCD). It achieves better model performance within the less time.

H. W. Lauw et al. (Eds.): PAKDD 2020, LNAI 12085, pp. 881–893, 2020.
https://doi.org/10.1007/978-3-030-47436-2_66

However, the high communication cost and massive repetitive calculations still affect the efficiency of framework. To address the above drawbacks, we design a distributed pre-computation mechanism incorporated with an optimized PS framework, by which we can avoid massive repetitive calculations and reduce the communication cost.

The main contributions of this paper can be listed as:

- We propose a novel distributed BCD method to optimize the FM under the PS. By dividing all the parameters into blocks according to the specific blocking scheme, and updating the parameters in a synchronous way, our proposed approach can achieve better accuracy and efficiency.
- We design a distributed pre-computation mechanism incorporated with the optimized PS framework, by which our approach can not only avoid massive repetitive calculations but also reduce the parameter exchanges.
- The experimental results show that our algorithm achieves better RMSE performance (3.8%–6.0%) than the state-of-the-art methods, and achieves a 4.6–12.3× speedup to obtain a comparable RMSE performance.

2 Preliminary

This section describes the FM model and its CD optimization.

2.1 Factorization Machine

Suppose the training set of a prediction problem is formulated by $D = \{(\mathbf{x}, y)\}$, where each pair $(\mathbf{x}, y)$ represents an instance $\mathbf{x}$ with p-dimension variables and its target value (or label) y, then a FM model of order $d = 2$ is defined as:

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{j=1}^{p} \mathbf{w}_j \mathbf{x}_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} \langle \mathbf{V}_j, \mathbf{V}_{j'} \rangle \mathbf{x}_j \mathbf{x}_{j'}, \tag{1}$$

where notation w_0 is the global bias, $\mathbf{w}_j$ models the weight of the j-th variable, and $\mathbf{V}_j$ represents a k-dimension interaction weight vector of j-th variable.

Give each $\theta \in (w_0, \mathbf{w}, \mathbf{V})$, FM can be represented as a linear combination of two functions g_θ and h_θ that are independent of θ. For example, when $\theta = \mathbf{w}_m$, then $h_\theta = \mathbf{x}_m$ and $g_\theta = w_0 + \sum_{i=1 \& i \neq m}^{p} \mathbf{w}_i \mathbf{x}_i + \sum_{i=1}^{p} \sum_{j=i+1}^{p} \langle \mathbf{V}_i, \mathbf{V}_j \rangle \mathbf{x}_i \mathbf{x}_j$.

$$\hat{y}(\mathbf{x}) = g_\theta(\mathbf{x}) + \theta h_\theta(\mathbf{x}) \tag{2}$$

2.2 Learning FM with Coordinate Descent

Suppose we are updating θ_j^t to θ_j^{t+1}, the least square loss function is defined as:

$$\sum_{(\mathbf{x}, y) \in D} (y - \hat{y}(\mathbf{x}|\Theta))^2 + \lambda_\Theta \|\theta_j^t\|^2 \tag{3}$$

By minimizing the above loss function, the optimal θ_j^{t+1} can be obtained as:

$$\theta_j^{t+1} = \frac{\sum_{(\mathbf{x},y)\in D}(y - \hat{y}(\mathbf{x}|\Theta) + \theta_j^t h_{\theta_j^t}(\mathbf{x}))h_{\theta_j^t}(\mathbf{x})}{\lambda_\Theta + \sum_{(\mathbf{x},y)\in D} h^2_{\theta_j^t}(\mathbf{x})} \tag{4}$$

It is clear that whenever updating a parameter, CD enumerates all the instances containing the related non-zero variables, and calculate $y - \hat{y}(\mathbf{x})$ and $h_\theta(\mathbf{x})$ for each instance, which is very time consuming. To improve the training efficiency, [13] proposed a pre-computation mechanism under single-machine FM. To reduce the complexity of computing $y - \hat{y}(\mathbf{x})$, the term $e(\mathbf{x}, y)$ is defined as:

$$e(\mathbf{x}, y) = y - \hat{y}(\mathbf{x}) \tag{5}$$

By storing $e(\mathbf{x}, y)$ in a vector $\mathbf{e} \in \mathbf{R}^n$ (n is the number of instances) and pre-computing it at the beginning, the computation of error terms can be done in constant time $O(1)$. After changing θ_j^t to θ_j^{t+1}, $e(\mathbf{x}, y)$ can be updated by:

$$e(\mathbf{x}, y) \leftarrow e(\mathbf{x}, y) + (\theta^{t+1} - \theta^t)h_\theta(\mathbf{x}) \tag{6}$$

To reduce the complexity of calculating $\mathbf{V}_{jf}$, we reformulate $h_{\mathbf{V}_{jf}}$ as:

$$h_{\mathbf{V}_{jf}}(\mathbf{x}) = \mathbf{x}_j q(\mathbf{x}, f) - \mathbf{x}_j^2 \mathbf{V}_{jf}, \quad q(\mathbf{x}, f) = \sum\nolimits_{j'=1}^{p} \mathbf{V}_{j'f}\mathbf{x}_{j'} \tag{7}$$

The term $q(\mathbf{x}, f)$ can be pre-computed for each instance and stored in a matrix $Q \in \mathbf{R}^{n\times k}$. By pre-computing $q(\mathbf{x}, f)$, $h_\theta(\mathbf{x})$ can be computed in constant time. When updating $\mathbf{V}_{jf}^t$ to $\mathbf{V}_{jf}^{t+1}$, the corresponding $q(\mathbf{x}, f)$ is updated as:

$$q(\mathbf{x}, f) \leftarrow q(\mathbf{x}, f) + (\mathbf{V}_{jf}^{t+1} - \mathbf{V}_{jf}^t)x_i \tag{8}$$

3 Distributed BCD Framework

In this section, we first introduce how to infer FM by BCD, and then give the details of the propose distributed BCD framework.

3.1 Learning FM with BCD

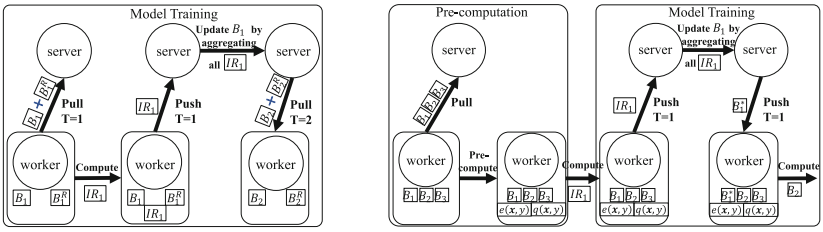
CD updates one coordinate each time while fixing others unchanged. However, if we directly extend it to the distributed platform without any changes, the model training will be very low-efficiency.

To overcome the above drawback, we propose a new CD method named BCD. BCD divides all coordinates into multiple blocks according to the specific scheme. During the training process, all the workers update the same block simultaneously while keeping the other blocks unchanged. The main problem for learning FM under BCD method is how to divide three types of FM model parameters $(w_0, \mathbf{w}, \mathbf{V})$ into blocks? Different blocking schemes may affect the

model performance significantly because they correspond to the different orders of the parameter updating in FM. To simplify the blocking process, we combine the global w_0 with $\mathbf{w}$ to form an extended $\mathbf{w}$. Then, according to the different combinations of parameter types, we can determine two types of blocking schemes. For the first scheme, each block contains not only $\mathbf{w}$, but also $\mathbf{V}$. In other words, we divide the parameters according to the order of the variables where each variable j correspond to $\mathbf{w}_j$ and $\mathbf{V}_j$, i.e., a block of parameters can be $\{\mathbf{w}_i, \mathbf{V}_i, \mathbf{w}_{i+1}, \mathbf{V}_{i+1}..., \mathbf{w}_j, \mathbf{V}_j\}$. We name it as **Mixed scheme**. For the second scheme, each block contains either $\mathbf{w}$ or $\mathbf{V}$. Or, put another way, we first update all blocks which include $\mathbf{w}$ parameters and then update the blocks with $\mathbf{V}$ parameters, i.e., a block of parameters can be $\{\mathbf{w}_i, \mathbf{w}_{i+1}, ..., \mathbf{w}_j\}$ or $\{\mathbf{V}_i, \mathbf{V}_{i+1}..., \mathbf{V}_j\}$. We name this scheme as **Separate scheme**.

3.2 Distributed BCD Under Standard PS

After analyzing the detail of inferring FM by BCD algorithm, we propose adapting BCD-based FM to a specific distributed environment. As far as we know, two types of distributed platforms, Map-Reduce (Spark) platform and PS framework, are popular and always used to the machine learning tasks. Generally speaking, PS framework is more efficient than Map-Reduce platform. Therefore, we propose a distributed BCD to learn FM under the PS framework.



(a) BCD under syn-PS (b) BCD under optimized PS

Fig. 1. The distributed BCD framework with pre-computation mechanism or not.

As mentioned in Sect. 3.1, the key idea of BCD is to update a block of parameters while maintaining all the other parameters unchanged. Thus, we can distribute BCD algorithm under synchronous PS framework. Before the model training, the server first initializes the model parameters and assigns the dataset to each worker randomly. Specific to each epoch, the workers update blocks in turns. In the update of each block, the workers firstly pull the valid parameters of current block and some other related parameters (the parameters which are co-occurrence with the parameters of the current block) from the server, and then calculate the intermediate results by the update rule. When all the workers pushed intermediate results to the server, the server updates the corresponding parameters by aggregating all intermediate results. To make the algorithm

more understandable, we illustrate the learning framework of distributed BCD in Fig. 1(a). In the figure, notation B_j denotes the valid parameters of block j, B_j^R denotes the related parameters which are co-occurrence with the parameters of block j, and IR_j denotes the intermediate results for block j.

However, we also observe some drawbacks of this approach. First, when updating a block, in addition to the current block, the workers always need to pull some other related parameters. The high communication cost affects the efficiency of the model training. Second, there are only part of parameters changed when calculating $\hat{y}(\mathbf{x})$ and $h_{\mathbf{V}}(\mathbf{x})$ in Eq. (4). But we completely re-calculate these two terms for updating each parameter. The massive repetitive calculations are very time consuming. To overcome these problems, we propose a distributed pre-computation mechanism under an optimized PS framework in the next section.

4 Distributed Precomputation

In this section, we design a distributed pre-computation mechanism incorporated with an optimized PS framework to solve this problem.

As a variant of CD method, our distributed BCD also has the massive repetitive calculations problem. To address this problem, we propose a distributed pr-computation mechanism to further improve the efficiency of our distributed BCD framework. The key idea is to pre-compute $e(\mathbf{x}, y)$ and $q(\mathbf{x}, f)$ for each instance at the beginning, store these pre-computation terms and the valid parameters (corresponding to non-zero variables of the training instance allocated to it) in each worker, and then modify them incrementally with new parameters.

However, two problems need to be solved when we implement the pre-computation to the distributed BCD framework discussed in Sect. 3.2. First, the workers in the above distributed framework pull the needed parameters before updating the block while the pull operation in the pre-computation mechanism is not necessary because all the valid parameters has been stored in each worker. Second, the pre-computation mechanism requests to update the pre-computation terms with the latest block at the end of block updating but the above system doesn't.

Obviously, to incorporate the pre-computation into our distributed BCD framework, we must optimize the above synchronous PS architecture. Follow the principle of pre-computation mechanism, we give the detail training process in Algorithm 1. Before the model learning, server node does the same initialization as the algorithm described in Sect. 3.1 (Line 2). And then, each worker pulls all valid parameters, and pre-compute e and q for all instances (Line 4). Specific to the update of each block, the workers first calculate the intermediate results for the current block and then push them back to the server (Line 7&8). After collecting all the intermediate results, server node updates the parameters of current block and then push the latest version back to all workers (Line 14&15). Once each worker receives the latest block, it updates its own parameters and pre-computation terms (Line 9&10). To better understand the algorithm, we

illustrate the training architecture of our optimized distributed framework in Fig. 1(b). The notations $e(\mathbf{x}, y)$ and $q(\mathbf{x}, y)$ are two pre-computation terms, and B_j^* denotes the latest parameters in block j. From the perspective of architecture, we can see that the purposes, occasions and triggers of operations are very different in these two frameworks. In terms of framework efficiency, the optimized PS framework has two advantages. First, the massive repetitive calculations can be avoided by the pre-computation. Second, the communication cost between the server and the workers is further reduced because only the parameters of the current block need to transfer in the optimized framework.

Algorithm 1. Distributed BCD for FM under optimized PS

Input: Dataset D, B blocks, M workers, maximal iterations T
Output: Model parameters $\mathbf{w}$ and $\mathbf{V}$

Servers:
Initialize parameters and assign D to workers
Workers:
Pull valid parameters and conduct pre-computing $e(\mathbf{x}, y)$ and $q(\mathbf{x}, f)$
for $t = 0$ to T **do**
    for $b = 0$ to B **do**
        Compute the inter results with Eq.(4) for b-th block
        Push all intermediate results to the server
        Receive pushed latest parameters from the server
        Update the pre-computation terms with Eq.(6) and Eq.(8)
    end for
end for
Servers:
Aggregate all intermediate results from workers and update parameters
Push the latest parameters to each worker

5 Experiments

In this section, we conduct various experiments to evaluate our algorithm.

5.1 Experimental Setup

Datasets. We perform our experiments on three datasets: MovieLens10M[1], Movielens20M(See footnote 1) and Yahoo music[2]. The details are shown in Table 1.

[1] https://grouplens.org/datasets/movielens.
[2] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r.

Table 1. Datasets

Dataset	Ratings	Users	Items
Movielens10M	10 million	71,567	10,681
Movielens20M	20 million	138,000	27,000
Yahoo music	717 million	1.8 million	136,000

Comparison Methods. The comparison methods are as follows:

FM-SCD-PS: Implemented by SCD and implemented on PS [20].

FM-Asyn-SGD-PS: A simple version of distributed FM proposed in [9] which is implemented with standard SGD and asynchronous PS.

FM-Ada-SGD-PS: Implemented with AdaGrad and asynchronous PS.

FM-Syn-SGD-PS: Implemented with standard SGD and synchronous PS.

FM-SGD-Spark: Implemented with standard SGD and Spark. For the sake of fairness, we simulate Spark with standard PS system.

FM-BCD-NPC-PS: Implemented with BCD and PS but no pre-computation.

FM-BCD-PS (Our final approach): Implemented with BCD and the distributed pre-computation mechanism under our optimized PS.

Evaluation Measures. We consider the following performance measurements:

- **Accuracy & Efficiency Performance.** We use RMSE and the training time to compare the performance of different methods respectively.
- **Parameter Analysis.** We analyze the effect of different factors in our methods: different blocking schemes and using pre-computation or not.

Platform and Implementation. All methods are implemented using Scala and performed on a cluster containing 16 machines. Among them, 15 machines are used as workers, of which each one contains 2 CPU cores and 16 G memory, and the other one is served as the server, which contains 8 CPU cores and 64 G memory. For the BCD-based algorithms, we default to set the block size as 5000 in Movielens[3] datasets and 10000 in Yahoo music dataset. Due to the different convergence rates of SGD and CD algorithms, we set T as 10, 20 and 3000 for BCD-based methods, CD baseline and gradient based algorithms, respectively.

5.2 Accuracy and Efficiency Performance

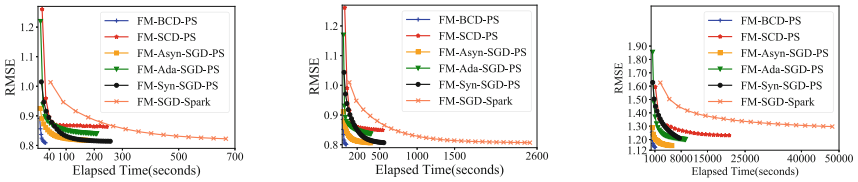
We compare RMSE performance and the corresponding elapsed time of all the comparison methods on the three datasets. To present the learning details of methods, we record a pair of the metric RMSE evaluated on the test set and the corresponding elapsed time. Then we plot all the (RMSE, time) pairs for each algorithm in Fig. 2.

From the results, we can see that the resultant (RMSE, time) points of the proposed FM-BCD-PS are more concentrated in the bottom-left corner than

[3] We use Movielens to represent two datasets: Movielens10M and Movielens20M.

those of other methods, which indicates that FM-BCD-PS can achieve smaller RMSE with less training time than the other methods. The reason for this is that our distributed BCD framework inherits the fast convergence rate of CD method and further improves the efficiency with pre-computation mechanism.

Specifically, when comparing with FM-SCD-PS, FM-BCD-PS achieves better RMSE performance (6.0%–7.8%) within less time (4.6–9.7× speedup for the same 10 iterations). The reason for this lies in that FM-SCD-PS tries to learn parameters on partial related instances and the corresponding imbalance workload in each worker. When comparing with the SGD-based baselines, FM-BCD-PS improves RMSE performance by 3.8%, 4.4% and 6.0%[4], respectively, on Movielens10M, Movielens20M and Yahoo music. Meanwhile, it achieves 7.6–11.1(See footnote 4)×, 8.3–12.3(See footnote 4)× and 5.4–7.4(See footnote 4)× speedup on Movielens10M, Movielens20M and Yahoo music, respectively. The main reason is that BCD converges faster than SGD.

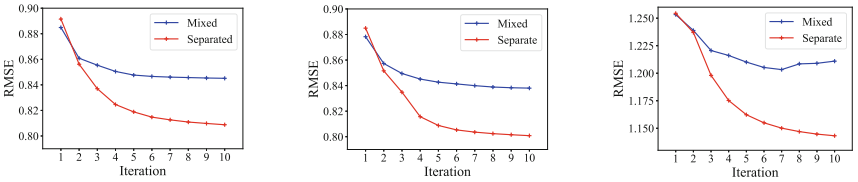


(a) Movielens10M (b) Movielens20M (c) Yahoo music

Fig. 2. Performance of all the comparison methods.

5.3 Parameter Analysis

We now discuss how different factors affect the model performance.



(a) Movielens10M (b) Movielens20M (c) Yahoo music

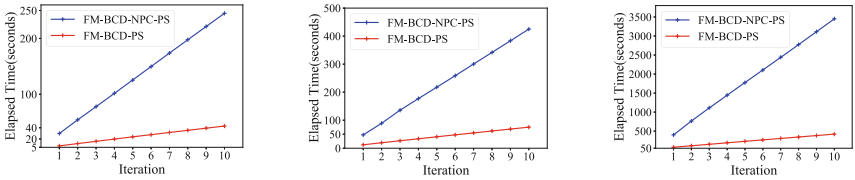
Fig. 3. Performance of FM-BCD-PS over different blocking schemes.

Effect of Different Blocking Schemes. As mentioned in Sect. 3.1, our proposed algorithm can apply two types of blocking schemes. We compare them and

[4] Here we abandon the results of FM-SGD-Spark for its bad performance.

study how different blocking schemes affect the RMSE performance of FM-BCD-PS in Fig. 3. From Fig. 3, we can see that the Separate scheme achieves better RMSE performance (4.6%-5.9%) with similar runtime in all datasets than the Mixed scheme. This is due to the fact that Mixed scheme update the coordinates $(\mathbf{w}_i, v_i)$ corresponding to the same feature i in the same block which will affect each other. Different from Mixed scheme, Separate scheme not only ensures that the coordinates corresponding to the same features locate in different blocks, but also guarantee that the most coordinates existing in the same instance are put in the different blocks. Thus, we adopt the Separate scheme on our proposed FM-BCD-PS algorithm in all other experiments.

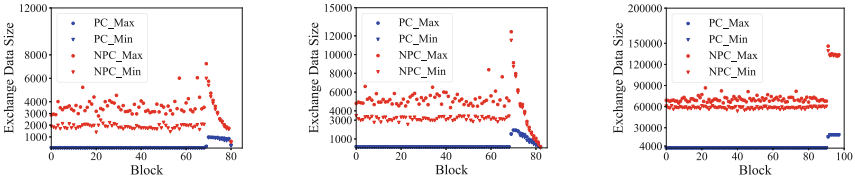
Effect of the Pre-computation Mechanism. We compare FM-BCD-NPC-PS with FM-BCD-PS, and study how the proposed pre-computation mechanism affects the efficiency performance of our algorithm in Fig. 4 and 5.



(a) Movielens10M (b) Movielens20M (c) Yahoo music

Fig. 4. Runtime of FM-BCD-PS and FM-BCD-NPC-PS.

In Fig. 4, we record the runtime for each iteration of two algorithms. From Fig. 4, we can see that FM-BCD-PS achieves up to 4.6× speedup on both Movielens datasets and 7.2× speedup on Yahoo music dataset when obtaining a comparable RMSE performance. The reason for this is that FM-BCD-PS not only reduces the size of data exchanged between the server and the workers but also avoids the massive repetitive calculations for updating coordinates.

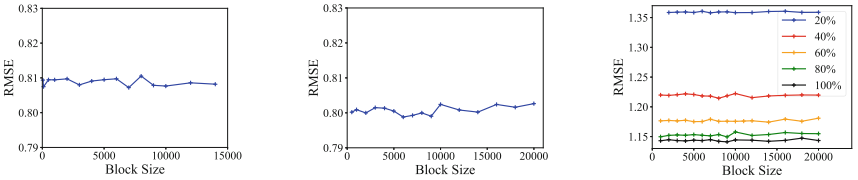


(a) Movielens10M (b) Movielens20M (c) Yahoo music

Fig. 5. Exchange Data Size of FM-BCD-PS and FM-BCD-NPC-PS.

In Fig. 5, the blue scatter plots and the red scatter plots illustrate the size of exchange data in each block of FM-BCD-PS and FM-BCD-NPC-PS respectively.

Furthermore, in the same color of scatter plots, the dot scatter plots and triangle scatter plots are used to respectively represent the maximum and minimum size of exchange data of the block. From Fig. 5, we can see that FM-BCD-PS achieves fewer exchange data size in each block than FM-BCD-NPC-PS. That is to say, the proposed distributed pre-computation mechanism can also reduce the communication cost between the server and the workers.

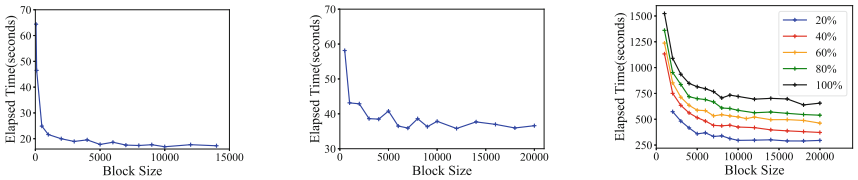


(a) Movielens10M (b) Movielens20M (c) Yahoo music

Fig. 6. RMSE of FM-BCD-PS over different block sizes.

Effect of Block Size. We study how the block size affects RMSE and efficiency performance of the proposed FM-BCD-PS in Fig. 6 and 7 respectively. To further illustrate the reliability of the experimental results, we sample 20%, 40%, 60%, 80% and 100% of Yahoo music to do experiments, respectively.

From Fig. 6, we can see that there is no big difference of RMSE over different block sizes on all datasets, which indicates that the block size has little or nothing effect on the model performance. The reason is that most coordinates existing in the same instance are still located in the different blocks. That is to say, the update of coordinates in the same block do not affect each other.



(a) Movielens10M (b) Movielens20M (c) Yahoo music

Fig. 7. Efficiency of FM-BCD-PS over different block sizes.

Figure 7 shows the training time can be reduced by increasing the block size. However, the elapsed time becomes stale when the block size reaches about 6000 on Movielens datasets and 12000 on Yahoo music dataset. This is due to the fact that there is a balance between the communication cost of each block and the number of blocks. In summary, we can speed up the model training by increasing the block size to an appropriate value while keeping the model performance.

6 Related Work

6.1 Factorization Machine

To achieve better performance, researchers mainly focus on three directions to extend FM. First, how to learn the FM with high-order interaction? Although [11] gives the general form of the FM model, the papers only propose the learning methods for 2-order FM [5,11,13]. To optimize the high-order FM efficiently, [2,3,10] give their solutions. Secondly, researchers consider combining FM with neural network algorithms in different ways [6,15,17,19]. Thirdly, to address the non-convex problem of the FM model, researchers try to reconstruct the FM and make it be convex [1,16]. To improve the training efficiency, some research efforts have been made to scale up FM [4,9,18,20]. These works focus on building FM on distributed frameworks. DiFacto [9] is a distributed FM and can perform fine-grained capacity control based on both data and model statistics. Zhong et al. proposed another version of distributed FM which can take advantages of both data parallelism and model parallelism [20]. To address the heavy communication cost problem, [18] proposes a client-to-client architecture to learn FM model.

6.2 Coordinate Descent on Big Data

To adapt the CD to large scale datasets, researchers tried to extend it to distributed platform [14,20]. [14] designed the first distributed CD system: Hydra. It divides the coordinates to disjoint subset and distributes them to all workers. If we adapt FM to Hydra, the instances stored in each worker are in large-scale and the calculation of gradients for coordinates are imbalanced. In such cases, the training efficiency will be greatly affected. Similar to Hydra, [20] proposed a stochastic CD (SCD) under the hybrid distributed framework. The distributed SCD-based FM has two drawbacks. First, its model performance is not good since the update of parameters may be based on the partial related instances. Second, the model training is not efficient because it does not ensure a balanced amount of calculation among the workers.

7 Discussion and Future Work

We discuss how to distribute logistic regression (LR), matrix factorization (MF) [8] and other factorization models to our proposed framework.

Logistic Regression. LR only considers the effect of independent variables and equals to the factorization machine that ignores the interaction part, i.e.,

$$\widehat{y}(\mathbf{x}) = w_0 + \sum\nolimits_{i=1}^{p} \mathbf{w}_i \mathbf{x}_i. \tag{9}$$

Compared with FM, LR has fewer features but with similar update rule. Thus, LR can directly apply the distributed BCD framework.

Matrix Factorization. MF only incorporates user and item identifications as features and equals to the FM that ignores the other heuristic features, i.e.,

$$\widehat{y}(\mathbf{x}) = w_0 + \sum_{u=1}^{m} \mathbf{w}_u + \sum_{i=1}^{n} \mathbf{w}_i + \sum_{u=1}^{m} \sum_{i=1}^{n} \langle \mathbf{V}_u, \mathbf{V}_i \rangle = w_0 + \mathbf{w}_u + \mathbf{w}_i + \langle \mathbf{V}_u, \mathbf{V}_i \rangle \quad (10)$$

where m and n denotes the number of users and items, respectively. In this case, w_u represents the bias for user u, and w_i is the bias for item i. Accordingly, there is only two interaction latent factors v_u and v_i for user u and item i, respectively.

Compared with FM, MF has fewer variables but with the same update rule. Thus, we can implement the distributed BCD on MF.

Other Factorization Models. FM is a general factorization method, which can mimic other state-of-the-art models like SVD++ [7], PITF [12] and so on. Therefore, our framework can optimize these models in a similar way.

8 Conclusions

This paper proposes a new distributed BCD framework to learn the FM. Through conducting a pre-computation mechanism incorporated with our optimized PS framework, we can avoid massive repetitive calculations and further reduce the communication cost. In addition, it is worth mentioning that our proposed distributed BCD framework can also be applied to many other factorization models, such as LR, MF, SVD++ and so on. We compare the proposed algorithm with the state-of-the-art baselines, and find that our proposed FM-BCD-PS can achieve better performance (3.8%–6.0% RMSE) within shorter time (4.6–12.3× speedup). For future work, we aim to generalize the distributed BCD framework and apply more other machine learning algorithms on it.

Acknowledgments. This work is supported by National Key R&D Program of China (No.2018YFB1004401), and NSFC under the grant No. (61772537, 61772536, 61702522, 61532021).

References

1. Blondel, M., Fujino, A., Ueda, N.: Convex factorization machines. In: Appice, A., Rodrigues, P.P., Santos Costa, V., Gama, J., Jorge, A., Soares, C. (eds.) ECML PKDD 2015. LNCS (LNAI), vol. 9285, pp. 19–35. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23525-7_2
2. Blondel, M., et al.: Higher-order factorization machines. In: NIPS 2016, pp. 3351–3359 (2016)
3. Blondel, M., et al.: Polynomial networks and factorization machines: new insights and efficient training algorithms. In: ICML 2016, pp. 850–858 (2016)
4. Cao, B., et al.: Multi-view machines. In: WSDM 2016, pp. 427–436 (2016)
5. Freudenthaler, C., et al.: Bayesian factorization machines (2011)
6. Guo, H., et al.: DeepFM: a factorization-machine based neural network for CTR prediction. In: IJCAI 2017, pp. 1725–1731 (2017)

7. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: SIGKDD 2008, pp. 426–434 (2008)
8. Koren, Y., et al.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
9. Li, M., et al.: DiFacto: distributed factorization machines. In: WSDM 2016, pp. 377–386 (2016)
10. Lu, C.T., et al.: Multilinear factorization machines for multi-task multi-view learning. In: WSDM 2017, pp. 701–709 (2017)
11. Rendle, S.: Factorization machines. In: ICDM 2010, pp. 995–1000. IEEE (2010)
12. Rendle, S., et al.: Pairwise interaction tensor factorization for personalized tag recommendation. In: WSDM 2010, pp. 81–90 (2010)
13. Rendle, S., et al.: Fast context-aware recommendations with factorization machines. In: SIGIR 2011, pp. 635–644 (2011)
14. Richtárik, P., et al.: Distributed coordinate descent method for learning with big data. J. Mach. Learn. Res. **17**(1), 2657–2681 (2016)
15. Xiao, J., et al.: Attentional factorization machines: learning the weight of feature interactions via attention networks. In: IJCAI 2017, pp. 3119–3125 (2017)
16. Yamada, M., et al.: Convex factorization machine for toxicogenomics prediction. In: SIGKDD 2017, pp. 1215–1224 (2017)
17. Zhang, W., Du, T., Wang, J.: Deep learning over multi-field categorical data. In: Ferro, N., et al. (eds.) ECIR 2016. LNCS, vol. 9626, pp. 45–57. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30671-1_4
18. Zhao, K., Zhang, J., Zhang, L., Li, C., Chen, H.: CDSFM: a circular distributed SGLD-based factorization machines. In: Pei, J., Manolopoulos, Y., Sadiq, S., Li, J. (eds.) DASFAA 2018. LNCS, vol. 10828, pp. 701–709. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91458-9_43
19. Zheng, L., et al.: Joint deep modeling of users and items using reviews for recommendation. In: WSDM 2017, pp. 425–434 (2017)
20. Zhong, E., et al.: Scaling factorization machines with parameter server. In: CIKM 2016, pp. 1583–1592 (2016)