f_{BGD}: A Unified Batch Gradient Method for Embedding Learning from Positive Unlabeled Data

Fajie Yuan,[1] **Xin Xin,**[1] **Xiangnan He,**[2] **Guibing Guo,**[3] **Weinan Zhang,**[4]
Chua Tat-Seng[2] and **Joemon M. Jose**[1]

University of Glasgow, UK[1], National University of Singapore, Singapore[2]
Northeastern University, China[3], Shanghai Jiao Tong University, China[4],

{f.yuan.1,x.xin.1,Joemon.Jose}@research.gla.ac.uk , xiangnanhe@gmail.com
guogb@swc.neu.edu.cn, wnzhang@sjtu.edu.cn, chuats@comp.nus.edu.sg

Abstract

Learning sparse features from only positive and unlabeled (PU) data is a fundamental task for problems of several domains, such as natural language processing (NLP), computer vision (CV), information retrieval (IR). Considering the numerous amount of unlabeled data, most prevalent methods rely on negative sampling (NS) to increase computational efficiency. However, sampling a fraction of unlabeled data as negative for training may ignore other important examples, and thus lead to non-optimal prediction performance. To address this, we present a fast and generic batch gradient descent optimizer (f_{BGD}) to learn from *all* training examples without sampling. By leveraging sparsity in PU data, we accelerate f_{BGD} by several magnitudes, making its time complexity the same level as the NS-based stochastic gradient descent method. Meanwhile, we observe that the standard batch gradient method suffers from gradient instability issues due to the sparsity property. Driven by a theoretical analysis for this potential cause, an intuitive solution arises naturally. To verify its efficacy, we perform experiments on multiple tasks with PU data across domains, and show that f_{BGD} consistently outperforms NS-based models on all tasks with comparable efficiency.

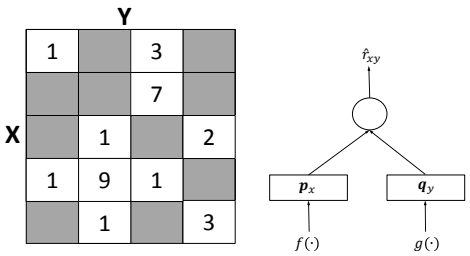
1 Introduction

Learning from only positive and unlabeled (or non-observed) data, aka PU learning, occurs in numerous domains such as NLP, CV, IR. In these scenarios, the observed training data usually consists of positive data only. Moreover, the overall training data is typically very sparse, since only a small fraction of positive examples are observed, and the non-observed negative examples are of a much larger scale.

To generalize well on such sparse data, embedding learning, such as word embedding in NLP (Mikolov et al., 2013b), image (category) embedding in CV (Weston et al., 2011), user (item) embedding in IR (Yuan et al., 2016a), and DNA k-mer embedding in genetic engineering (Ng, 2017), has become a common practice. However, learning embeddings from PU (or positive-only) data is computationally expensive, since each observed positive example needs to be paired with *all* non-observed negatives.

To learn from large-scale non-observed data, most recent embedding methods employ negative sampling (NS) and stochastic gradient descent (SGD) for efficient optimization (Mikolov et al., 2013b; Weston et al., 2012; Guo et al., 2018a,b; Yuan et al., 2016a, 2017). However, the training time and prediction accuracy are largely determined by the sampling distribution and size of negative samples. Sampling a fraction of non-observed data as negative for training may ignore other useful examples, or lead to insufficient training of them. This is our main motivation in this work. Moreover, it is known that SGD performs frequent gradient updates with a high variance, which can cause the objective function to fluctuate heavily near the optimum (Ruder, 2016). By contrast, batch gradient descent (BGD) computes the gradient on all training data for updating a model parameter. As such, the learning process has the potential to converge to a better optimum. Unfortunately, the low efficiency caused by the full-batch gradient computation makes it less applicable to large-scale datasets.

To deal with these issues, we introduce a fast and generic batch gradient descent algorithm (called f_{BGD}) for learning various embedding models from positive-only data. f_{BGD} optimizes a commonly used square loss function that accounts for all non-observed examples without any sampling. To ensure the learning efficiency, we acceler-



(a) Co-occurrence matrix (b) Embedding function

Figure 1: (a): PU data with (x, y) co-occurrence matrix $\mathbf{H}$. The grey cells denote no explicitly observed (x, y) examples. (b): Embedding function. $f(\cdot)$ and $g(\cdot)$ are functions to construct the embedding vectors $\mathbf{p}_x$ and $\mathbf{q}_y$ respectively.

ate f_{BGD} with rigorous mathematical reasoning. Notably, despite that f_{BGD} computes loss and gradients over all examples, its actual complexity is comparable with NS-based SGD methods that utilize only partial examples. Furthermore, we show that standard batch learning are prone to the gradient exploding and vanishing problem and solve it by an intuitive way. We provide an intuitive solution after understanding its potential cause. To summarize, the main work of this paper is as follows:

- We propose a unified BGD implementation to solve the sparse feature learning problem from PU data. For efficiency optimization, we accelerate it by a natural reformulation of loss and rearrangement for the dot product. For generality, we identify the dot product structure for a class of complex embedding models.
- We provide theoretical explanations that the standard batch gradient learning suffers from gradient instability issues when learning embedding models due to large batched summation of sparse features.
- We employ a general weighting scheme that suits well for unlabeled examples in various domains, which not only largely improves the prediction accuracy of f_{BGD}, but also make efficiency optimization possible.
- f_{BGD} achieves state-of-the-art performance in multiple research fields with comparable costs to NS-based SGD methods. Insightful comparisons for sampling based methods have been thoroughly studied. Another insightful observation is that many specific models used in one of these fields are promising to benefit others by minor (or no) changes. This opens a new direction of research to bridge these fields.
- We release the source code of f_{BGD} at: `https://github.com/fajieyuan/fBGD`.

2 Problem Formulation

2.1 Learning from Positive-Only Data

Assuming we have two sets of examples that are available for training: the positive set P and an unlabeled set U, which is typically non-observed and contains both positive and negative samples. Each sample in P is an observed (x, y) pair, where $x \in X$ and $y \in Y$. X and Y are the set of distinct x and y respectively. For a given x, we have a set of relevant y labelled, denoted by Y_x^+, the size of which is much smaller than that of the non-observed set Y_x^-. As shown in Figure 1 (a), we can use a matrix $\mathbf{H} \in \mathbb{R}^{|X|\times|Y|}$ to denote the historical interactions between x and y. The goal of PU learning is to find a function $\hat{r}_{xy}$ (parameterized by Θ) that explains a set of observed pairs (x, y), such as "relevant-or-not" and "like-or-not".

2.2 Embedding Models

Embedding models have been widely adopted in many specific PU learning tasks. In this work, we focus on optimizing the embedding functions that can be *explicitly* or *implicitly* expressed by a dot product structure, given below.

$$\hat{r}_{xy} = \langle \mathbf{p}_x, \mathbf{q}_y \rangle = \sum_{d=1}^{g} p_{x,d} q_{y,d} \tag{1}$$

where $\mathbf{p}_x$ and $\mathbf{q}_y$ are *compressed* embedding vectors with embedding dimension g. They can be obtained by directly projecting the ID of row/column into the embedding space (i.e., explicit structure as in Xin et al. (2018) and He et al. (2016b)), or projecting with other features of row/column (i.e., implicit stricture as in Rendle and Freudenthaler (2014); Bayer et al. (2017)). The time complexity of evaluating this equation is $O(g)$. Note that the implicit dot product structure can describe a variety of multi-linear models, such as SVDFeature (Chen et al., 2012) and tensor models (Bailey and Aeron, 2017; Rendle and Schmidt-Thieme, 2010). Later, we will show how to construct this dot product structure for some state-of-the-art embedding models.

2.3 Loss Function and BGD Optimization

We propose optimizing the standard regression loss, which can also be used for classification and ranking tasks. Unlike Pennington et al. (2014), the optimized loss function should explicitly account for all unlabeled samples.

$$J(\Theta) = \sum_{(x,y)\in P} \alpha_{xy}^{+}(r^{+} - \hat{r}_{xy})^2 + \underbrace{\sum_{(x,y)\in U} \alpha_{xy}^{-}(r^{-} - \hat{r}_{xy})^2}_{JM(\Theta)} \tag{2}$$

where $JM(\Theta)$ denotes the errors of all unlabeled examples, α_{xy}^{+} and α_{xy}^{-} are the weight functions. Eq. (2) can be minimized by BGD, which computes the gradient of loss function w.r.t. $\theta \in \Theta$ on the entire (positive and unlabeled) samples:

$$\theta \leftarrow \theta - \gamma \nabla_\theta J(\theta) \tag{3}$$

where γ is the learning rate, and $\nabla_\theta J(\theta)$ is the gradient of $J(\Theta)$ w.r.t. θ, given below:

$$\nabla_\theta J(\theta) = 2\Big(\underbrace{\sum_{(x,y)\in P} \alpha^+_{xy}(r^+ - \hat{r}_{xy})\nabla_\theta \hat{r}_{xy}}_{O(|Y^+_x|g)} + \underbrace{\sum_{(x,y)\in U} \alpha^-_{xy}(r^- - \hat{r}_{xy})\nabla_\theta \hat{r}_{xy}}_{O(|Y^-_x|g)} \Big) \tag{4}$$

where $O(|Y^+_x|g)$ and $O(|Y^-_x|g)$ are the complexity of gradient computation on positive and unlabeled data.

2.4 Efficiency Issues

As can be seen, the second term $J_M(\Theta)$ in Eq.(2) dominates the computational complexity. This is because computing $J_M(\Theta)$ in Eq.(2) has almost $O(|X||Y|g)$ time because $|P| \ll |U|$. Similarly, updating a parameter (under the explicit dot product setting), e.g., $p_{x,d}$, by Eq.(4) is $O((|Y^-_x|)g)$, or $O(|Y|g)$, because $|Y^+_x| \ll |Y^-_x|$. The total cost by iterating over all $p_{x,d}$ in Θ in each iteration becomes O($|X||Y|g$). Clearly, the straight-forward way to calculate gradients by BGD is generally infeasible, because $|X||Y|$ can easily reach billion level or even higher.

3 Fast & Generic BGD for PU Learning

In this section, we first derive an efficient loss function of the generic f_{BGD} for PU learning, and show how to extend it to various embedding models. Then, we design a general weighting scheme for the unlabeled examples in f_{BGD}.

3.1 Efficient f_{BGD} Loss

In the above learning setting, the dominant computation is the minimization of $J_M(\Theta)$ in Eq.(2) since each x has its standalone unlabeled set of y, i.e., Y^-_x. As such, the BGD algorithm basically needs to iterate through all elements in Y^-_x, and repeat the operation for all $x \in X$, which produces the main cost. To solve the problem, we reformulate the standard BGD loss according to the set theory[1]. Naturally, for any PU learning problem the loss of U (unlabeled) data can be expressed by the *residual* between the loss of *all* data and that of P (positive) data.

$$J_M(\Theta) = \sum_{x\in X}\Big(\sum_{y\in Y} \alpha^-_{xy}(r^- - \hat{r}_{xy})^2 - \sum_{y\in Y^+_x} \alpha^-_{xy}(r^- - \hat{r}_{xy})^2\Big) \tag{5}$$

A new objective function can be achieved by substituting Eq.(5) in Eq.(2). We combine the two terms that associates with P data together into a single term (Note that r^+, r^-, α^+_{xy}, α^-_{xy} are independent of $\theta \in \Theta$). $J(\Theta)$ is rewritten as

$$J(\Theta) = const + J_A(\Theta) + J_P(\Theta) \tag{6}$$

where

$$\begin{aligned} J_A(\Theta) &= \sum_{x\in X}\sum_{y\in Y} \alpha^-_{xy}(\hat{r}_{xy} - r^-)^2 \\ J_P(\Theta) &= \sum_{x\in X}\sum_{y\in Y^+_x} (\alpha^+_{xy} - \alpha^-_{xy})\Big(\hat{r}_{xy} - \frac{\alpha^+_{xy}r^+ - \alpha^-_{xy}r^-}{\alpha^+_{xy} - \alpha^-_{xy}}\Big)^2 \end{aligned} \tag{7}$$

where $J_A(\Theta)$ and $J_P(\Theta)$ denote the loss for *all* and P data respectively; $const$ denotes a Θ-invariant constant value. Clearly, the loss of U data has been eliminated. The new computation complexity is now in $\widetilde{J}_A(\Theta)$, which is part of $J_A(\Theta)$, defined as:

$$\widetilde{J}_A(\Theta) = \sum_{x\in X}\sum_{y\in Y} \alpha^-_{xy}\hat{r}^2_{xy} - 2r^- \sum_{x\in X}\sum_{y\in Y} \alpha^-_{xy}\hat{r}_{xy} \tag{8}$$

So far, we have focused on the loss without considering the specific formulation of model prediction $\hat{r}_{xy}$. As described in Section 2.2, we focus on $\hat{r}_{xy}$ that can be either *explicitly* or *implicitly* formalized as a dot product (i.e., Eq.(1)) structure based on embedding vectors of x and y. In the following, we first show the generalized transformation for a *compressed* dot product structure. Then, we show how to apply f_{BGD} to various complex embedding functions with more input features by constructing the similar structure.

$$\begin{aligned} \alpha^-_{xy}\hat{r}^2_{xy} &= \alpha^-_{xy}\sum_{d=1}^{g} p_{x,d}q_{y,d}\sum_{d'=1}^{g} p_{x,d'}q_{y,d'} \\ &= \sum_{d=1}^{g}\sum_{d'=1}^{g} \alpha^-_{xy}\Big(p_{x,d}p_{x,d'}\Big)\Big(q_{y,d}q_{y,d'}\Big) \end{aligned} \tag{9}$$

where we observe that there exists a very nice structure in above equation — if α^-_{xy} is a constant value or a value only associates with x or y but not (x, y) pair. Considering that there is no observed (x, y) interaction in unlabeled examples, it is reasonable to set α^-_{xy} as α^-_y or α^-_x. The simplified weight design is a necessary condition for efficient optimization in the following. Here we continue to discuss the algorithm, assuming $\alpha^-_{xy} = \alpha^-_y$, and later show how to design a good weighting scheme. With this setting, the interaction between $p_{x,d}$ and $q_{y,d}$ can be safely separated. Thereby, $\sum_{y\in Y} \alpha^-_y q_{y,d}q_{y,d'}$ can be independent of the optimization of x-related parameters. That is, we could achieve a significant speed-up by precomputing this term. Let caches $S^q_{dd'} = \sum_{y\in Y} \alpha^-_y q_{y,d}q_{y,d'}$, and $S^q_d = \sum_{y\in Y} \alpha^-_y q_{y,d}$, $\widetilde{J}_A(\Theta)$ is derived as follows

$$\widetilde{J}_A(\Theta) = \sum_{d=1}^{g}\sum_{d'=1}^{g} S^q_{dd'}\sum_{x\in X} p_{x,d}p_{x,d'} - 2r^-\sum_{d=1}^{g} S^q_d \sum_{x\in X} p_{x,d} \tag{10}$$

The rearrangement of nested sums in Eq.(10) is the key transformation that allows the fast optimization of f_{BGD}. The computation complexity has reduced from $O(|X||Y|g)$ in Eq.(8) to $O((|X| + |Y|)g^2)$ in Eq.(10). Optimization details regarding the gradient computation are given in Section 3.3.

[1] The set relation was also applied in He et al. (2016b); Xin et al. (2018), which can be regarded as a special case of f_{BGD} as $\hat{r}_{xy}$ is only limited to an explicit dot product function that deals with two features in a specific domain.

3.2 Identifying the Dot Product Structure

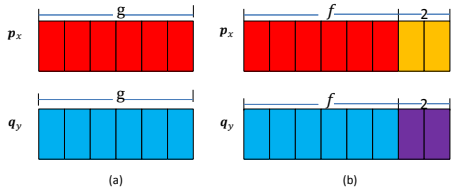


Figure 2: (a) denotes the explicit dot product structure, such as in AllVec (Xin et al., 2018), while (b) is the dot product that implicitly exists in SVDFeature. Each cell denotes a real value.

We notice that the dot product structure implicitly exists in a variety of embedding modes. Here we show the structure for a complex embedding model, aka SVDFeature (Chen et al., 2012), which can be used in context-aware recommender systems (CARS), content-based image retrieval system (CBIR) and prior knowledge based word representation. We also identify the dot product structure for two tensor-based embedding models (Bailey and Aeron, 2017; Rendle and Schmidt-Thieme, 2010) in Appendix A. The model equation of SVDFeature is defined as

$$\hat{r}_{xy} = w_0 + \mathbf{w}\mathbf{z}^T + \sum_{j=1}^{pX}\sum_{j'=1}^{pY}\langle \mathbf{v}_j^X, \mathbf{v}_{j'}^Y\rangle z_{x,j}^X z_{y,j'}^Y \tag{11}$$

where $\mathbf{z}$ is the feature vector. E.g., in a context-aware music recommender system, it is defined as

$$\mathbf{z}=(\underbrace{\overbrace{0,...,1,...,0}^{\text{user IDs}},\overbrace{0,0.1,...,0.1,0}^{\text{10 previous tracks IDs}},\overbrace{0,1}^{\text{time}},\overbrace{0,...,1,...,0}^{\text{current track ID}},\overbrace{0,1,...,0}^{\text{artists}}}_{\mathbf{z}_x^X\qquad\qquad\qquad\qquad\qquad\mathbf{z}_y^Y})$$

x and y are described by $\mathbf{z}_x^X$ and $\mathbf{z}_y^Y$ respectively. $z_{x,j}^X$ is j-th element in $\mathbf{z}_x^X$, which is x-th row in $\mathbf{Z}^X \in \mathbb{R}^{|X|\times pX}$. pX is the number of features in $\mathbf{z}_x^X$. $\mathbf{v}_j^X$ is the j-th row in $\mathbf{V}^X \in \mathbb{R}^{pX\times f}$, where f is the original embedding size. Inspired by Rendle and Freudenthaler (2014), we rewrite Eq.(11) as an implicit dot product structure (see Figure 2).

$$\hat{r}_{xy} = \sum_{d=1}^{g} p_{x,d}q_{y,d} \tag{12}$$

where $g = f + 2$ and

$$\begin{aligned} p_{x,d} &= \sum_{j=1}^{pX} z_{x,j}^X v_{j,d}^X\,, & q_{y,d} &= \sum_{j=1}^{pY} z_{y,j}^Y v_{j,d}^Y \\ p_{x,f+1} &= w_0 + \sum_{j=1}^{pX} w_j z_{x,j}^X\,, & q_{y,f+1} &= 1 \\ p_{x,f+2} &= 1\,, & q_{y,f+2} &= \sum_{j=1}^{pY} w_{(j+pX)} z_{y,j}^Y \end{aligned} \tag{13}$$

where $v_{j,d}^X$ is the d-th element in $\mathbf{v}_j^X$. Next, we show the gradient computation for both explicit (i.e., Eq. (1)) and implicit dot product (e.g., Eq. (11)) structure.

3.3 Efficient Gradient

Following Section 3.1, the gradients of $\widetilde{J}_A(\theta)$ w.r.t. $\theta^X \in \Theta^X$ is given by

$$\begin{aligned} \nabla_{\theta^X}\widetilde{J}_A(\theta) = 2\sum_{d=1}^{g}\sum_{d'=1}^{g} S_{dd'}^q \sum_{x\in X} p_{x,d'}\nabla_\theta p_{x,d} \\ - 2r^- \sum_{d=1}^{g} S_d^q \sum_{x\in X}\nabla_\theta p_{x,d} \end{aligned} \tag{14}$$

The optimization process of $\theta^Y \in \Theta^Y$ is almost symmetric to θ^X, except that the weighting scheme α_y^- is inside the sum of $y \in Y$. In what follows, we present the gradient computation for Eq.(14) with both explicit and implicit dot products.

3.3.1 Gradient Computation with Eq.(1)

Assume Eq.(1) is a basic dot product, the gradient of $p_{x,d}$ with respect to p_{x^*,d^*} is given by

$$\nabla_{p_{x^*,d^*}} p_{x,d} = \begin{cases} 1 & x = x^* \wedge d = d^* \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

Thus, Eq.(14) simplifies to

$$\nabla_{p_{x^*,d^*}}\widetilde{J}_A(\theta) = 2\sum_{d=1}^{g} S_{d^*d}^q p_{x^*,d} - 2r^- S_{d^*}^q \tag{16}$$

The cost of Eq.(16) is in $O(g)$, and correspondingly, updating all $\theta^X \in \Theta^X$ is $O(|X|g^2)$. Overall, gradient computation for all $\theta \in \Theta$ is $O((|X| + |Y|)g^2 + |P|g)$, where $O(|P|g)$ is the complexity for the gradients of the positive loss. In contrast, the cost of NS-SGD is $O((n + 1)|P|g)$, where $n + 1$ means n negative y and 1 positive y.

3.3.2 Gradient Computation with Eq.(11)

The gradients of $p_{x,d}$ with respect to w_{j^*} and v_{j^*,d^*}^X are given by

$$\nabla_{w_{j^*}} p_{x,d} = \begin{cases} z_{x,j^*}^X & d = f+1 \\ 0 & \text{otherwise} \end{cases}, \nabla_{v_{j^*,d^*}^X} p_{x,d} = \begin{cases} z_{x,j^*}^X & d \le f \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

Thus, Eq.(14) w.r.t. to w_{j^*} and v_{j^*,d^*}^X simplifies to

$$\nabla_{w_{j^*}}\widetilde{J}_A(\theta) = 2\sum_{d=1}^{g} S_{(f+1)d}^q \sum_{x\in X} p_{x,d} z_{x,j^*}^X - 2r^- S_{(f+1)}^q \sum_{x\in X} z_{x,j^*}^X \tag{18}$$

$$\nabla_{v_{j^*,d^*}^X}\widetilde{J}_A(\theta) = 2\sum_{d=1}^{g} S_{d^*d}^q \sum_{x\in X} p_{x,d} z_{x,j^*}^X - 2r^- S_{d^*}^q \sum_{x\in X} z_{x,j^*}^X \tag{19}$$

Note that the computation of sums over $x \in X$ can be accelerated by only iterating over x where $z_{x,j^*}^X \neq 0$. Moreover, $p_{x,d}$ is able to be precomputed to reduce the cost. Although $p_{x,d}$ changes when updating θ^X, it can be updated in synchronization with the changes in θ^X, denoted by $\triangle\theta^X$.

$$p_{x,d} \leftarrow p_{x,d} + z_{x,j}^X \triangle\theta^X = p_{x,d} - z_{x,j}^X \gamma \nabla_\theta J(\theta) \tag{20}$$

Analogously with Section 3.3.1, the total time complexity of $\nabla_\theta J_A(\theta)$ (or $\nabla_\theta \widetilde{J}_A(\theta)$) in one iteration for all parameters is $O(g^2(N(X)+N(Y)))$, where $N(X)$ and $N(Y)$ are the number of non-zero elements in $\mathbf{Z}^X$ and $\mathbf{Z}^Y$. Finally, the efficient computation for θ is reasonably given as follows[2]

$$\theta \leftarrow \theta - \gamma(\nabla_\theta J_A(\theta) + \nabla_\theta J_p(\theta)) \tag{21}$$

The detailed implementation of f_{BGD} is in Appendix B.

3.4 Effective Weighting on Unlabeled Data

Now that the basic description of the speed-up process for f_{BGD} is completed, we proceed to discuss the weighting scheme in this section. First, in terms of α^+_{xy}, any reasonable weighting scheme could be adopted and will not affect the analysed computation. For example, on the word embedding task (see Section 5) we set α^+_{xy} the same as in GloVe (Pennington et al., 2014), while we set it as 1 for the other tasks considering that there is no available frequency information for positive (x, y) pairs.

As for α^-_y, we design a non-uniform weighting scheme based on the property of y. Our weighting scheme is originally motivated by the frequency-based oversampling idea such as Skip-gram model (Mikolov et al., 2013b) and (Yuan et al., 2016a). However, both methods are tailored for the SGD or the mini-batch gradient descent (MGD) (He and Chua, 2017) optimization. Clearly, sampling techniques do not suit our model, because the focus of f_{BGD} is an all-sample based optimization method. Hence, a frequency-based weighting scheme is more suitable for our optimization setting. To effectively differentiate true negative and unknown examples, we assign a larger weight for the unlabeled data with high y frequency, and a smaller weight for the low-frequency y.

$$\alpha^-_y = \alpha_0 \frac{(e^{z_y}-1)^\rho}{\sum_{y=1}^{|Y|}(e^{z_y}-1)^\rho} \text{ where } z_y = \frac{p_y}{|P|} \tag{22}$$

where p_y denotes the frequency of y, which is the number of observations in P, and α_0 determinates the overall weight of unlabeled examples to solve the imbalanced-class problem. The exponent ρ controls weight distribution, which should be tuned based on the dataset.

4 Improved f_{BGD}

So far, we have discussed the efficiency optimization of f_{BGD}, However, we observe unreliable results during evaluation especially for complex embedding models with more input features, as shown in Figure 3. A novel contribution here is to reveal why unstable gradient issues will occur for the standard BGD.

[2] Again, $\nabla_\theta J_p(\Theta)$ can be calculated by the standard way, which has the same time complexity with NS-SGD with the same ratio of negative and positive samples.

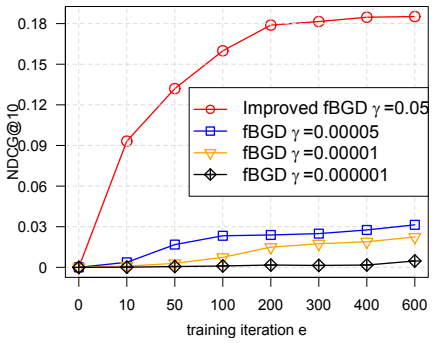


Figure 3: Performance of the improved f_{BGD} (Section 4.2) and standard f_{BGD} on Last.fm with four features. Note for the standard f_{BGD}, some gradients will be evaluated as infinite (NaN) when $\gamma > 5 \times 10^{-5}$. Clearly, f_{BGD} with vanishing gradient performs poorly on Last.fm even by fine tuning the learning rate.

4.1 Gradient Instability of f_{BGD}

While the unstable gradient problem, such as the gradient exploding and vanishing, has been observed when training deep neural networks (He et al., 2016a), the optimized models of f_{BGD} in this paper are mostly shallow embeddings. Therefore, the cause of the unstable gradient issue in our case is fundamentally different from that in the existing deep layer models, in the sense that in deep models unstable gradients occur mainly due to cumulative multiplying of small/big numbers from previous layers, whereas in f_{BGD} it is caused by the large batched summation of sparse features. We expect the following theoretical analysis and solution could provide practical guidelines for the future development of batch gradient optimization.

To understand the weird behavior of gradient instability, we need to revisit the form of gradients. We take the derivation of $v^X_{j,d}$ $(d \le f)$ in Eq. (11) w.r.t. the loss of positive data as an example.

$$\nabla_{v^X_{j^*,d^*}} J_P(\theta) = 2 \sum_{x \in X} \sum_{y \in Y^+_x} z^X_{x,j^*}(\alpha^+_{xy} - \alpha^-_y) \Big(\hat{r}_{xy} - \frac{\alpha^+_{xy} r^+_{xy} - \alpha^-_y r^-_{xy}}{\alpha^+_{xy} - \alpha^-_y}\Big) q_{y,d^*} \tag{23}$$

Due to the data sparsity, to compute $\sum_{x \in X} \sum_{y \in Y^+_x} z^X_{x,j^*}$ we only need to consider $x \in X$ that has a non-zero z^X_{x,j^*} (note that for a feature j, most x have z^X_{x,j^*} equal to zero which can be safely ignored). Let l_{j^*} be the number of non-zero elements in the j^*-th column of $\mathbf{Z}^X$.

In real-world data sets, the number of rows in $\mathbf{Z}^X$, i.e., $|X|$, can easily scale to many millions or even billion level and, therefore, it is very likely that l_{j^*} has distinct magnitudes for a different column j^*. Moreover, in Eq.(23) there is another summation $\sum_{y \in Y^+_x}$, which represents the size of observed y for x. The component value of $\sum_{x \in X} \sum_{y \in Y^+_x} x^X_{x,j^*}$ in Eq.(23) varies from 1 to $|X| \cdot |Y|$, assuming $\mathbf{Z}^X$ is a binary matrix. This indicates

the value of Eq.(23) may be very unstable: $\nabla_{v^X_{j^*,d}} J_P(\theta)$ can be too large for a denser feature j^* that is accompanied by a large $\sum_{x\in X}\sum_{y\in Y_x^+} z^X_{x,j^*}$ (e.g., $= 10^6$), while it may be too small for a sparser feature with a small $\sum_{x\in X}\sum_{y\in Y_x^+} z^X_{x,j^*}$(e.g., $= 1$). Accordingly, the overall gradient $\nabla_\theta J(\theta)$ in Eq.(21) has the same unstable problem. In this case, a uniform learning rate γ is no longer suitable because $\nabla_\theta J(\theta)$ with a larger $\sum_{x\in X}\sum_{y\in Y_x^+} z^X_{x,j^*}$ is likely to explode (i.e. $\nabla_\theta J(\theta) =$ NaN) if using a large γ, while $\nabla_\theta J(\theta)$ with a smaller $\sum_{x\in X}\sum_{y\in Y_x^+} z^X_{x,j^*}$ may vanish (i.e. $\nabla_\theta J(\theta) \approx 0$) if using a small γ. Generally, it is hard or even impossible to find a medium learning rate that balances reasonably well in both conditions. To gain more insight into the performance of f_{BGD} with unstable gradients, we show results with different learning rates in Figure 3.

Interestingly, we empirically find that on many datasets with only two input features (or an explicit dot product structure), the gradient instability problem may be alleviated by carefully tuning γ. In other words, by many trials with different learning rates, f_{BGD} sometimes is able to offer reasonable results. However, on data sets with more feature variables (e.g., Last.fm), the outputs of f_{BGD} are prone to the NaN error. This is because in the pure dot product setting, the nested summation $\sum_{x\in X}$ can be dropped. As such, although the gradient instability issue may still happen because of $\sum_{y\in Y_x^+}$, it is less severe as the value of $|Y_x^+|$ is much smaller than that of $l_j \cdot |Y_x^+|$.

4.2 Solving the Unstable Gradient Problem

The above theoretical analysis for the gradient estimation over all data suggests that the same learning rate does not hold for all model parameters due to the large batched summation of sparse features. Analytically, by assigning a specific learning rate for each parameter update, we can control the unstable gradient to a certain extent. In other words, f_{BGD} should perform larger updates for small $\nabla_\theta J(\theta)$, and vice verse.

Based on the above analysis, an intuitive solution is to adapt the learning rate for each parameter, such as having done in Adagrad Duchi et al. (2011). While Adagrad is originally proposed for stochastic gradient method to accelerate convergence, here we show how to apply it on the full gradient method to address the gradient instability issue. Denoting γ_t as the learning rate for the t-th update, we then assign a personalized learning rate for each parameter θ:

$$\gamma_t(\theta) = \frac{\gamma}{G_t(\theta)}, \quad G_t(\theta) = \begin{cases} \nabla_\theta J(\theta)_t + \epsilon & G_t(\theta) = 0 \\ \sqrt{\sum_{t=1}(\nabla_\theta J(\theta)_t)^2} & G_t(\theta) \neq 0 \end{cases} \tag{24}$$

where $\nabla_\theta J(\theta)_t$ is the gradient w.r.t. θ for the $t-$th update, $G_T(\theta)$ is the accumulation of the squared gradients, and ϵ is a smoothing term to avoid division by zero, set as 10^{-4}. The overall algorithm of improved f_{BGD} can be implemented by replacing γ in Eq.(21) and Eq.(20) with the new $\gamma_t(\theta)$.

Table 1: Dataset statistics ($|U| = |X \times Y| - |P|$). "Open" is the OpenImages dataset. "K", "M" and "B" are short for thousand, million and billion. (x, y) denotes (word, context), (user, item) and (image, label) in WE, OCCF and IC respectively. Note that user and item in Lastfm contain user- and item-related variables.

Data	$\lvert X\rvert$	$\lvert Y\rvert$	pX	pY	$\lvert P\rvert$	$\lvert X\times Y\rvert$
NewsIR	83K	83K	83K	83K	150M	6.9B
Text8	71K	71K	71K	71K	47M	5.0B
Yahoo	200K	136K	200K	136K	76M	27.2B
Lastfm	63K	58K	65K	75K	1.3M	3.7B
Open	1.4M	7.5K	1.4M	7.5K	11.4M	10.5B

5 Experiments

f_{BGD} is a generic PU learning model and can be applied in a wide range of tasks with PU data and sparse features. For evaluation purpose, we verify its performance in three fields — word embedding (WC) of NLP, collaborative filtering (CF) of IR, and image classification (IC) of CV.

5.1 Experimental Setup

5.1.1 Datasets

We use five benchmark datasets for evaluation: NewsIR[3] and Text8[4] for WE, Yahoo music[5] and Lastfm[6] for CF, and OpenImages Krasin et al. (2017) for IC. For NewsIR, we preprocess them by a standard pipeline, i.e., removing non-textual elements, lowercasing and tokenization. For Yahoo, we use the "train_0" file. For Lastfm, we follow Weston et al. (2012) by extracting the latest one-week actions per user via the timestamp, and consider two tracks played by the same user as "consecutive" if they are played within 90 minutes. It is used as a context-aware (or next-item) recommendation dataset, where each x contains a user and his previously played music tracks and each y contains a music track and its artist. For OpenImages, we randomly sample a number of (image, label) pairs from the original dataset. The statistics of datasets are summarized in Table 1.

5.1.2 Baselines and Evaluation

For WE, we compare f_{BGD} with Skip-gram (Mikolov et al., 2013b) and GloVe (Pennington et al., 2014). For CF, we compare it with SVDFeature (Chen et al., 2012), BPRFM (Rendle, 2012; Rendle et al., 2009), and λFM

[3] http://research.signalmedia.co/newsir16/signal-dataset.html

[4] http://mattmahoney.net/dc/text8.zip

[5] http://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=2

[6] http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/

Table 2: Model Comparison. "SG" and "SVDF" is short for Skip-gram and SVDFeature respectively.

Model	Sampler	Ratio	Optimizer	Loss
SG×10	Static	1:10	SGD	LOG
GloVe	-	-	SGD	LS
SVDF×8	Uniform	1:8	SGD	LS
BPRFM	Uniform	1:1	SGD	L2R
λFM	Static	1:1	SGD	L2R
WARP	Dynamic	1:1	SGD	L2R
VSE-ens	Dynamic	1:1	SGD	L2R
f_{BGD}	-	-	BGD	LS

"Uniform","Static" and "Dynamic" are short for a uniform, static and dynamic sampler respectively. Static sampler means the sampling distribution of negative examples is defined before training and keeps unchanged during the whole optimization process. Dynamic sampler changes the sampling distribution of negative examples according to the current state of the learning algorithm. "Ratio" represents the positive-to-negative example ratio. "LS", "LOG", and "L2R" are short for the least square, logistic, and learning-to-rank loss function respectively. Note that we only evaluate λFM with the static sampler in this paper, considering the efficiency issues of the dynamic samplers.

Table 3: Results on the word analogy task. "Sem", "Syn" and "Tot" denote the semantic, syntactic and total accuracy [%]. The positive-to-negative example ratio in SG is $1:10$ and $1:25$ in NewsIR and Text8 respectively suggested by Mikolov et al. (2013b).

Model	NewsIR			Text8		
	Sem	Syn	Tot	Sem	Syn	Tot
SG	70.8	47.5	58.1	47.5	32.3	38.6
GloVe	78.8	41.6	58.5	45.1	26.9	34.5
f_{BGD}	77.0	46.1	**59.7**	56.5	30.4	**41.3**

(Yuan et al., 2016a). For IC, we compare it with SVDFeature, WARP (Weston et al., 2011) and VSE-ens (Guo et al., 2018b). For SVDFeature, we optimize it with the least square loss, and use the negative sampling strategy. The negative examples used for training are uniformly sampled from U. To show the impact of negative sampling, we vary the size of negative examples for each positive one. E.g., SVDFeature×8 means the positive-to-negative ratio is 1:8. Table 2 summarizes the characteristics of these baselines.

To assess the performance of f_{BGD} on the WE task, we use the analogical reasoning task introduced by Mikolov et al. (2013a). While to evaluate the CF and IC tasks, we regard them as a ranking or classification task. We report NDCG@10 (Normalized Discounted Cumulative Gain) and MRR@10 (Mean Reciprocal Rank) for CF and AUC (Area Under ROC Curve) for IC.

On the WE task, we evaluate the quality of the word vectors learned from the training datasets. For CF and IC, we adopt the leave-one-out evaluation protocol (Rendle et al., 2009).

5.1.3 Experimental Settings

All reported results on each task use a fixed-size embedding dimension without special mention. Specifically, we set embedding dimension as 200, 20 and 100 for the WE, CF and IC tasks respectively. For f_{BGD}, we set the learning rate γ as 0.05 with Adagrad on all three tasks. Regarding r^+, we apply the PPMI (positive pointwise mutual information) on the WE task inspired by Levy and Goldberg (2014). For the other two tasks, we simply set it as 1. r^- can be set as 0 or -0.5. Empirically, we report results of baseline models with optimal hyperparameters whereas for f_{BGD}, we only report results with above default settings.

Table 4: Results on the CF task. NDCG and MRR denote NDCG@10 and MRR@10 respectively. For each measure, the best results for SVDFeature (SVDF) and all models are indicated in bold.

Model	Yahoo		Lastfm	
	NDCG	MRR	NDCG	MRR
SVDF×1	0.0067	0.0044	0.0436	0.0285
SVDF×4	0.0133	0.009	**0.0565**	**0.0391**
SVDF×16	0.0186	0.0132	0.0535	0.0390
SVDF×64	**0.0197**	**0.0139**	0.0360	0.0263
SVDF×256	0.0193	0.0139	-	-
BPRFM	0.0178	0.0124	0.1056	0.0740
λFM	0.0200	0.0140	0.1312	0.0950
f_{BGD}	**0.0224**	**0.0161**	**0.1800**	**0.1371**

Table 5: Results on the IC task.

Metric	SVDF×1	SVDF×4	SVDF×16
AUC	0.681	0.724	**0.747**
SVDF×64	WARP	VSE-ens	f_{BGD}
0.663	0.696	0.723	**0.772**

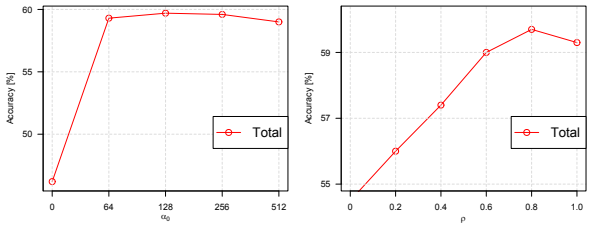
5.2 Accuracy and Discussion

5.2.1 Overall Results and Sampling Bias

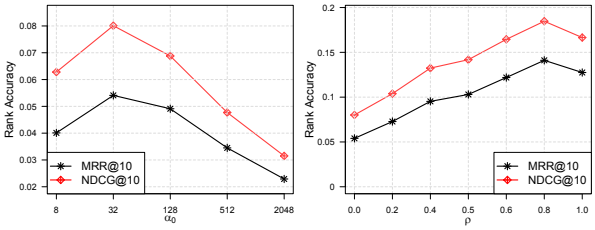
We report results of all models in Tables 3, 4 and 5 for the three tasks. Our first observation is that f_{BGD} achieves the best performance across all the evaluation metrics and all the datasets. For example, f_{BGD} outperforms Skip-gram and GloVe in the two text corpora w.r.t. the total accuracy.

Remarkably, f_{BGD} can easily outperform the strong baselines (e.g., λFM, WARP and VSE-ens) in the ranking and classification tasks, although it optimizes a regression loss which is typically suboptimal for ranking and classification. We attribute the advantage of f_{BGD} to two aspects: (1) the optimization of each model parameter in f_{BGD} makes use of all unlabeled data, whereas the SGD models (including MGD) only use a fraction of sampled data. In other words, important negative examples may be ignored or under-trained; (2) the tailored weighting scheme can help BGD address the imbalanced-class problem in PU data (see Table 1), and assign fine-grained penalties for further improvement.

Our second key observation in the following also verifies the above analysis. As shown in Table 4 (Yahoo), SVDF×64 $>$ SVDF×16 $>$SVDF×4 $>$SVDF×1, while



(a) NewsIR: tune α_0 ($\rho = 0.8$) (b) NewsIR: tune ρ (α_0=128)



(c) Lastfm: tuneα_0 ($\rho = 0$) (d) Lastfm: tune ρ (α_0=32)

Figure 4: Impact of α_0 and ρ on f_{BGD}.

Table 6: Accuracy evaluation of f_{BGD} by adding features. u, p, i and a denote user, last item (song), next item and artist respectively.

Metrics	(u, i)	(u, p, i)	(u, p, i, a)
NDCG@10	0.0416	0.1722	0.1800
MRR@10	0.0281	0.1301	0.1371

SVDF×256 <SVDF×64. The results suggest that the performance of SGD models is sensitive to the sampling size of negative examples. To be more specific, one negative sample for a positive example is insufficient to achieve optimal performance; sampling more negative examples is beneficial but too many negative examples may also hurt the performance. In addition, although SVDF×64 > SVDF×1, the theoretical computation complexity of SVDF×64 is about 32 times higher than SVDF×1. Still in Table 4, λFM largely improves BPRFM, which demonstrates the impact of sampling distribution of negative examples (see Table 2). However, the true distribution of negative (x, y) pairs in unlabeled samples is unknown in practice. That is, regardless of what samplers are used, sampling based methods cannot converge to the same loss with all examples or true data.

Table 7: Time Complexity of various optimizers per iteration. $|X||Y|g$ is much larger than $(|X|+|Y|)g^2$ and $|P|g$. The size relation between $|P|g$ and $(|X|+|Y|)g^2$ depends on the sparsity of the data and the embedding dimension g.

Model	Time Complexity
SGD×n	$O((n+1)\|P\|g)$
BGD	$O(\|X\|\|Y\|g)$
f_{BGD}	$O((\|X\|+\|Y\|)g^2 + \|P\|g)$

Table 8: Comparison of runtime (second/minute/hour [s/m/h]). "S", "I" and "T" represents the training time for a single iteration, overall iterations and total time respectively. SGD denotes Skip-gram for NewsIR and SVDFeature for other datasets. n is set as the optimal value, i.e., 10, 4 and 16 for NewsIR, Lastfm and OpenImages respectively.

Model	NewsIR			Lastfm			OpenImages		
	S	I	T	S	I	T	S	I	T
SGD×n	715s	15	179m	156s	50	130m	26m	100	43h
f_{BGD}	388s	75	485m	26s	200	87m	576s	200	32h

5.2.2 Impact of Weighting in f_{BGD}

In this section, we show the impact of the weighting function for f_{BGD}. We take the NewsIR and Lastfm datasets as an example, and omit similar results in other datasets. Figure 4 shows the prediction quality by tuning α_0 and ρ in the weight function. We first fix the value of ρ (e.g., 0 in CF and 0.8 in WE) to study the impact of α_0. Then, we use the best value of α_0 to study ρ. As shown, the overall coefficient α_0 largely impacts the performance as the amount of positive and "negative" examples fed in f_{BGD} is highly imbalanced, the results of which are reflected in (a) and (c). We observe that a proper ρ can improve the performance, as shown in (b) and (d). The intuition behind the improvement is that high-frequent y (words or items) that are not observed in Y_x^+ have a higher likelihood to be true negatives, and thus deserve more penalties.

5.2.3 Effectiveness in Modelling features

To show the generality of f_{BGD}, we have described how to apply it to complex embedding models, e.g., SVDFeature used in CARS. For example, we gradually add features for f_{BGD} on Lastfm and report results in Table 6. As expected, f_{BGD} performs largely better with (u, p, i) than (u, i) and that performance is further enhanced with (u, p, i, a). That is, f_{BGD} yields the best prediction accuracy with all features, demonstrating its power on feature engineering.

5.2.4 Runtime

Table 7 summarizes the time complexity of the SGD, BGD and f_{BGD} algorithms in one iteration when optimizing the pure dot product function. As shown, the complexity of f_{BGD} is determined by the gradient computation of both positive and unlabeled data, rather than the unlabeled data only. In practice, the runtime is mainly affected by the data sparsity and embedding size. For example, on the WE task, O($|P|g$) is larger than O($(|X|+|Y|)g^2$), while on the IC task O($(|X|+|Y|)g^2$) is almost 10 times larger than O($|P|g$) because the NewsIR and Text8 datasets are much denser than the OpenImages dataset (see Table 1). We have

compared the overall training time[7] of f_{BGD} with the NS-based SGD methods in Table 8. It shows that f_{BGD} obtains comparable efficiency to the classic SGD-based algorithm. More detailed runtime results in shown in Appendix C.

6 Related Work

Gradient methods are one of the most popular algorithms to perform optimization in the practice of machine learning. They have also been widely used for training embedding models, and have almost dominated the optimization field. So far the most commonly used gradient optimization method is SGD (Mikolov et al., 2013a,b; Pennington et al., 2014; Rendle et al., 2009; Weston et al., 2011) or a compromise MGD (mini-batch gradient descent) (Li et al., 2014; He et al., 2017), which attempts to approximate the true gradient by a single or a mini-batch of instances with sampling techniques. However, the balance between computing the expensive true gradient based on the whole batch and the immediate gradient based on a single or a fraction of instances could easily result in suboptimal performance. More importantly, on large-scale data the sampling size and distribution for SGD/MGD also significantly affect the convergence rate and prediction accuracy (Bengio and Senécal, 2008). In particular for PU data, it is non-trivial to sample from large and highly imbalanced unlabeled data. Most works deal with this issue by proposing a certain trade-off between efficiency and accuracy. For example, various negative sampling methods have been proposed in recent literature (Mikolov et al., 2013b; Pan et al., 2008; Weston et al., 2012; Yuan et al., 2016a, 2017; Wang et al., 2017; Guo et al., 2018a,b). The basic idea behind this is to select the most informative unlabeled instances as negative examples for an SGD/MGD trainer which, however, easily leads to bias itself. Moreover, all aforementioned works either expose efficiency issues with a dynamic sampler (Weston et al., 2012; Wang et al., 2017; Yuan et al., 2016a) or result in suboptimal training instances with a uniform (Rendle et al., 2009) or static (defined before optimization) sampler (Mikolov et al., 2013a,b; Yuan et al., 2016b, 2017) in practice. Our f_{BGD} in this work departs from all above studies by adopting BGD to optimize general embedding models with the entire batch of data.

It is worth mentioning that the f_{BGD} method is inspired from our extensive empirical studies on previous works (He et al., 2016b; Bayer et al., 2017; Xin et al., 2018). The main difference is that these works are focused on a specific task, e.g., He et al. (2016b); Bayer et al. (2017) are only on recommendation and Xin et al. (2018) is on word representation. Specifically, He et al. (2016b); Xin et al. (2018) worked on the simple matrix factorization model, which cannot be used to incorporate other features, such as contextual variables associated with each observed example. While the alternating least squares (ALS) method proposed in Bayer et al. (2017) can be applied to any *k-separable* model[8], it requires to estimate the second-order derivatives to apply the Newton update and only supports a constant weight on unlabeled examples; moreover, our empirical evidence shows that training with Newton update is (1) very sensitive to initialization point and the regularization term, and (2) highly unstable due to some gradient issues, especially for embedding models (e.g., FM and SVD-Feature) with many input features or large word corpus. By contrast, this work targets at solving the generic PU learning problem with generic embedding models. It leads to a unified solution that is applicable to a wide range of tasks, including but not limited to the ones demonstrated in this paper, with just simple changes on input features.

7 Conclusion

This work has several key contributions. First, we showed how to efficiently train a variety of embedding models by batch gradient descent for positive unlabeled (PU) data. Second, we identified an unstable gradient issue in f_{BGD} due to the large batched summation of sparse features, and solve it by an intuitive way. To make the prediction accuracy of f_{BGD} comparable to the state-of-the-arts, we employed a general weighting scheme for unlabeled examples. Despite simple, the weighting scheme could address two challenges, namely imbalanced-class issue in PU data and the differentiation of true negative and unknown examples. We studied the performance of f_{BGD} in three subfields, and showed that f_{BGD} outperformed state-of-the-art baselines. Compared with the ranking or classification models, f_{BGD} is clearly a regression model, which means the real-valued scores estimated by it are more informative than those by ranking or classification algorithms. This will make our method highly attractive for practical usage. Moreover, the proposed f_{BGD} is not limited to the domains discussed in this paper. It potentially benefits many real-world applications with PU data, such as genes association studies (Asgari and Mofrad, 2015; Yang et al., 2014) and data stream mining (Li et al., 2009), etc.

References

E. Asgari and M. R. Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS one*, 2015.

E. Bailey and S. Aeron. Word embeddings via tensor factorization. *arXiv preprint arXiv:1704.02686*, 2017.

I. Bayer, X. He, B. Kanagal, and S. Rendle. A generic coordinate descent framework for learning from implicit feedback. In *WWW*. International World Wide Web Conferences Steering Committee, 2017.

[7] For fairness, efficiency tests for all training models were running on Intel(R) Xeon(R) E5620 @ 2.40GHz CPU and 49G RAM. Note that on the WE task, we implemented all models with 8 threads in parallel, while on the other two tasks, we implemented the models in a single-thread.

[8] In essence, the concept of k-separable is to describe a model with a dot product structure of Equation (1).

Y. Bengio and J.-S. Senécal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, 2008.

T. Chen, W. Zhang, Q. Lu, K. Chen, Z. Zheng, and Y. Yu. Svdfeature: a toolkit for feature-based collaborative filtering. *JMLR*, 2012.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2011.

G. Guo, S. Ouyang, F. Yuan, and X. Wang. Approximating word ranking and negative sampling for word embedding. In *IJCAI*, 2018a.

G. Guo, S. Zhai, F. Yuan, Y. Liu, and X. Wang. Vse-ens: Visual-semantic embeddings with efficient negative sampling. In *AAAI*, 2018b.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016a.

X. He and T.-S. Chua. Neural factorization machines for sparse predictive analytics. In *SIGIR*, 2017.

X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, 2016b.

X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *WWW*, 2017.

I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy. Open-images: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github.com/openimages*, 2017.

O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *NIPS*, 2014.

M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *KDD*, 2014.

X.-L. Li, P. S. Yu, B. Liu, and S.-K. Ng. Positive unlabeled learning for data stream classification. In *Proceedings of the 2009 SIAM International Conference on Data Mining*. SIAM, 2009.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013b.

P. Ng. dna2vec: Consistent vector representations of variable-length k-mers. *arXiv preprint arXiv:1701.06279*, 2017.

R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *ICDM*, 2008.

J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

S. Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 2012.

S. Rendle and C. Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *WSDM*, 2014.

S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM*, 2010.

S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, 2009.

S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

J. Wang, L. Yu, W. Zhang, Y. Gong, Y. Xu, B. Wang, P. Zhang, and D. Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR*, 2017.

J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, 2011.

J. Weston, C. Wang, R. Weiss, and A. Berenzweig. Latent collaborative retrieval. *ICML*, 2012.

X. Xin, F. Yuan, X. He, and J. M. Jose. Batch is not heavy: Learning word representations from all samples. In *ACL*, 2018.

P. Yang, X. Li, H.-N. Chua, C.-K. Kwoh, and S.-K. Ng. Ensemble positive unlabeled learning for disease gene identification. *PloS one*, 2014.

F. Yuan, G. Guo, J. M. Jose, L. Chen, H. Yu, and W. Zhang. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *CIKM*, 2016a.

F. Yuan, J. M. Jose, G. Guo, L. Chen, H. Yu, and R. S. Alkhawaldeh. Joint geo-spatial preference and pairwise ranking for point-of-interest recommendation. In *ICTAI*. IEEE, 2016b.

F. Yuan, G. Guo, J. M. Jose, L. Chen, H. Yu, and W. Zhang. Boostfm: Boosted factorization machines for top-n feature-based recommendation. In *IUI*, 2017.

A Unified Batch Gradient Approach for Positive Unlabeled Learning

A Dot Product Structures in Tensor Models

(1) For the Pairwise Interaction Tensor Factorization (Rendle and Schmidt-Thieme, 2010) (Note that here $\mathbf{V}^X$, $\mathbf{V}^Y$ and $\mathbf{V}^H$ are shared for pairwise interactions, while in the original paper they are independent.):

$$\hat{r}_{xy} = \sum_{d=1}^{f} v^X_{j,d} v^H_{j',d} v + \sum_{d=1}^{f} v^X_{j,d} v^Y_{j'',d} + \sum_{d=1}^{f} v^H_{j',d} v^Y_{j'',d} \tag{25}$$

we have $g = f + 1$ and

$$\begin{aligned} p_{xh,d} &= v^X_{j,d} + v^H_{j',d}\ , \quad q_{y,d} = v^Y_{j'',d} \\ p_{xh,f+1} &= \sum_{d=1}^{f} v^X_{j,d} v^H_{j',d}\ , \quad q_{y,f+1} = 1 \end{aligned} \tag{26}$$

(2) For a tensor factorization (TF) — rank-f Canonical Polyadic Decomposition (Bailey and Aeron, 2017):

$$\hat{r}_{xy} = \sum_{d=1}^{f} v^X_{j,d} v^H_{j',d} v^Y_{j'',d} \tag{27}$$

we have $g = f$ and

$$p_{xh,d} = v^X_{j,d} v^H_{j',d}\ , \quad q_{y,d} = v^Y_{j'',d} \tag{28}$$

where the gradient, e.g., $\nabla_{v^X_{j,d}} p_{xh,d}$ is $v^H_{j',d}$. Note we assume that $v^X_{j,d}$ and $v^H_{j',d}$ belong to x-related parameters, while $v^Y_{j'',d}$ is a parameter related to y.

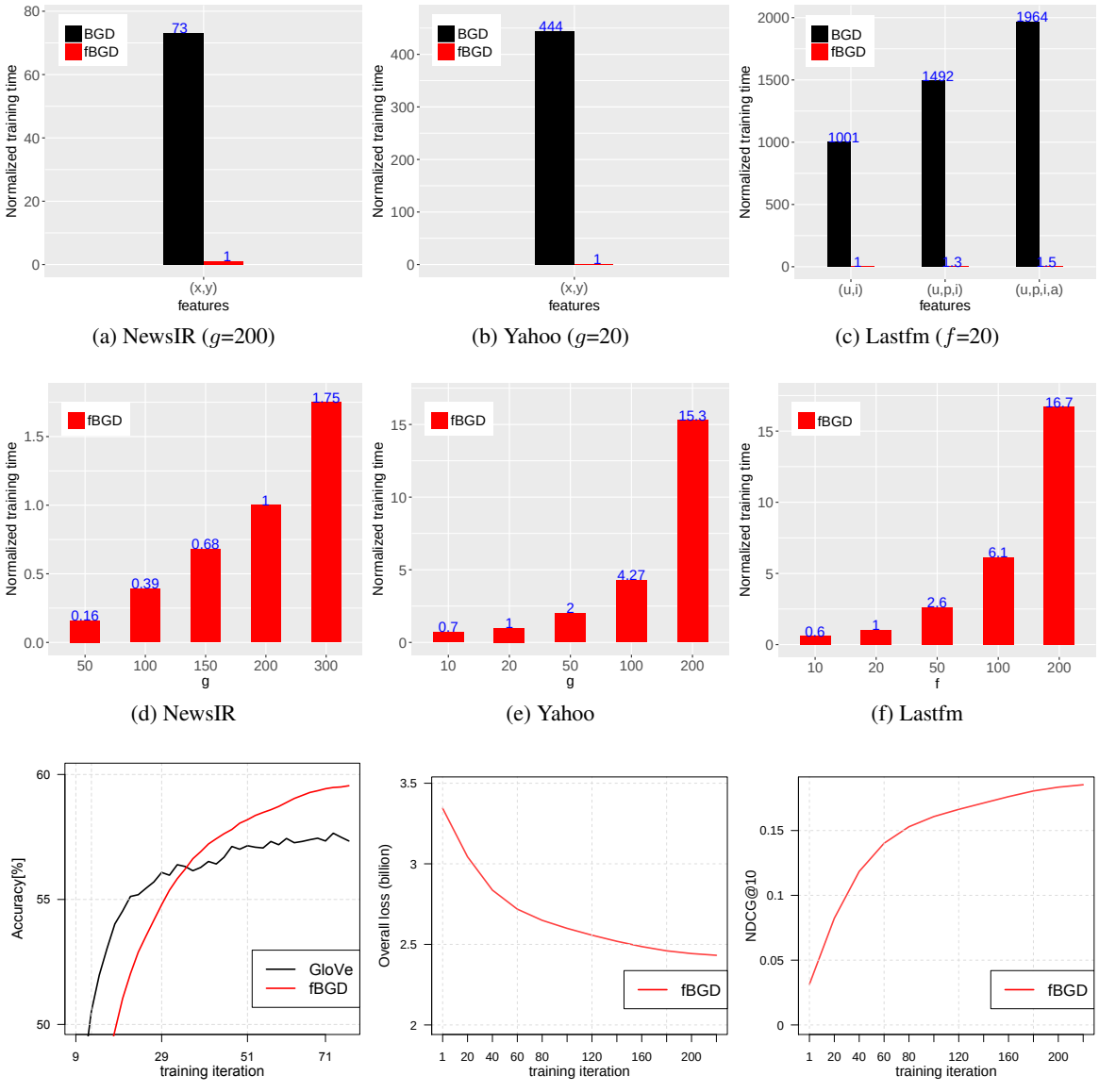
B Learning of f_{BGD}

Algorithm 1 summarizes the accelerated algorithm for f_{BGD}. We define the $S^p_{dd'}$ and S^p_d caches as $S^p_{dd'} = \sum_{x \in X} p_{x,d} p_{x,d'}$ and $S^p_d = \sum_{x \in X} p_{x,d}$ respectively. The gradients w.r.t. $\theta^Y \in \Theta^Y$ is given by

$$\nabla_{\theta^Y} \widetilde{J}_A(\theta) = 2 \sum_{d=1}^{g} \sum_{d'=1}^{g} S^p_{dd'} \sum_{y \in Y} \alpha^-_y q_{y,d'} \nabla_\theta q_{y,d} - 2r^- \sum_{d=1}^{g} S^p_d \sum_{y \in Y} \alpha^-_y \nabla_\theta q_{y,d} \tag{29}$$

Note that Line 5-10 and 21 can be omitted when optimizing the pure dot product function, i.e., Eq.(1); g' in Line 18 equals to g and $g - 2$ (i.e., f) for Eq.(1) and Eq.(11) respectively.

C Runtime Results



(a) NewsIR (g=200) (b) Yahoo (g=20) (c) Lastfm (f=20)

(d) NewsIR (e) Yahoo (f) Lastfm

(g) NewsIR (h) Yahoo (i) Lastfm

Figure 5: (a), (b) and (c) show the runtime per iteration (f_{BGD} vs BGD). One unit in (a)(d), (b)(e) and (c)(f) is 388, 165, 26 seconds respectively. (d), (e) and (f) show the runtime change (per iteration) by increasing embedding size. (g), (h) and (i) show the convergence behavior reflected by the training loss or predicted accuracy.

Algorithm 1 Generic f_{BGD} Learning Algorithm

1: **Input:** P, X, Y, Cache vectors $\mathbf{s}^q$, $\mathbf{s}^p$, Cache matrices $\mathbf{E}$, $\mathbf{Q}$, $\mathbf{S}^q$, $\mathbf{S}^p$;
2: **Output:** Θ
3: Initialize $\Theta \sim \mathcal{N}(0, 0.01)$:
4: **for** $e = 1, ...,$ maxiter **do**
5: **for** $d \in \{1, .., g\}$ **do**
6: **for** $x \in X$ **do**
7: Compute p_{xd}, store in $\mathbf{E}$ ($\mathbf{E} \in \mathbb{R}^{|X| \times g}$)
8: **end for**
9: Repeat line 6 to 8 for $y \in Y$
10: **end for**
11: **for** $d \in \{1, .., g\}$ **do**
12: Compute S^q_d, store in $\mathbf{s}^q$ ($\mathbf{s}^q \in \mathbb{R}^g$)
13: **for** $d' \in \{1, .., g\}$ **do**
14: Compute $S^q_{dd'}$, store in $\mathbf{S}^q$ ($\mathbf{S}^q \in \mathbb{R}^{g \times g}$)
15: **end for**
16: **end for**
17: **for** $j \in \{1, .., pX\}$ **do**
18: **for** $d \in \{1, .., g'\}$ **do**
19: Compute $\nabla_{\theta^X} J_A(\Theta)$,$\nabla_{\theta^X} J_p(\Theta)$
20: Update θ^X as in Eq.(21)
21: Update $p_{x,d}$ as in Eq.(20)
22: **end for**
23: **end for**
24: Repeat line 11 to 23 for updating θ^Y
25: **end for**