Iterative Splits of Quadratic Bounds for Scalable Binary Tensor Factorization

Beyza Ermis
Department of Computer Engineering
Bogazici University
34342 Bebek, Turkey
beyza.ermis@boun.edu.tr

Guillaume Bouchard
Xerox Research Centre Europe
6, Chemin de Maupertuis,
38240 Meylan, France
guillaume.bouchard@xrce.xerox.com

Abstract

Binary matrices and tensors are popular data structures that need to be efficiently approximated by low-rank representations. A standard approach is to minimize the logistic loss, well suited for binary data. In many cases, the number n^+ of non-zero elements in the tensor is much smaller than the total number N of possible entries in the tensor. This create a problem for large tensors because the computation of the logistic loss has a linear time complexity with N. In this paper, we show that an alternative approach is to minimize the quadratic loss (root mean square error) which leads to algorithms with a training time complexity that is reduced from $O(N)$ to $O(n^+)$, as proposed earlier in the specific case of alternating least-square algorithms. In addition, we propose and study a greedy algorithm that partitions the tensor into smaller tensors, each approximated by a quadratic upper bound. This technique provides a time-accuracy tradeoff between a fast but approximate algorithm and an accurate but slow algorithm. We show that this technique leads to a considerable speedup in learning of real world tensors.

1 INTRODUCTION

In multi-relational data factorization problems [20, 25], many negative examples are implicitly created for relations that are *not* true. For example, in a knowledge base of family relationships, the fact `isFather(x,y)` can be considered as a positive example and automatically induces $m - 1$ negative examples of the form `not(isFather(x,z))`, for all m individuals z different from y. In other words, for some relations, one positive example is always associated with several thousands of negative examples. The focus of our work is to consider algorithms that are independent on this number of negative examples. In a different domain, state-of-the-art detection systems in computer vision are based on a binary classifier applied many times on a dense multi-resolution scan of an image [7]. Here, most of the examples do not contain the object to be detected and negative patches often overwhelm the number of positive examples. Finally, another classical example of such problems with unbalanced categories corresponds to recommender systems taking into account implicit feedback: in this domain, it corresponds to the signal that if a user did not do some action, such as buying an object in an online shopping web site or did not click on an online advertisement, then a *negative* training example is created to take into account the fact that the proposed item or advert might not be appropriate. While these negative examples are sometimes subject to controversy since one does not know whether the recommendation was correct or not, they are nevertheless considered as very useful by practitioners and are key components of most of online recommendation engines [9].

For a binary classification problem where the total number n^+ of positive examples is largely inferior to the total number n^- of negative examples, the complexity of most of the existing learning algorithms is *at least* linear in the number $n = n^+ + n^-$ of training samples, since it is a general belief that every training point needs to be loaded in memory at least once. In fact, the sparsity of the data can be used to drastically reduce the computation time of square-norm minimization problems, as proposed by [9] using a alternating least square algorithm, where each least square problem has a complexity linear in the number of positive data only. We will give an alternative derivation of this result and show that it is also valid for gradient-based algorithms.

However, the squared loss is not always satisfactory. For example, binary tensor decomposition with logistic loss gives much better predictive performance than minimizing the squared loss. The downside of it is that the computational cost increases significantly [14, 19], and one usually relies on heuristic rules to subsample the negative examples [11]. The time to minimize the logistic loss (or

other non-quadratic loss) scales linearly in the total number $n = n^+ + n^-$ of observations, which is a real issue in these heavily unbalanced datasets for which $n^+ << n^-$.

In this work, the key contributions are:

- For matrix and tensor factorization models learned by minimizing the squared loss, we show that all the algorithms can benefit from this speedup, i.e. it is not restricted to the alternating least square algorithm of [8],
- We propose a new algorithm to minimize non-quadratic losses. It is based on the partitioning of the tensor into blocks and the use of Jaakkola's quadratic upper bound to the logistic loss [10]. While Jaakkola's bound has already been used to factorize large matrices, the case of unbalanced datasets was not addressed [24]. Our work can be viewed as a novel application of these upper-bounding techniques, where the incremental refinement of the approximation provides a natural way to correct the optimality gap introduced by the bound.

2 PROBLEM FORMULATION

Let $\Omega := \{(i_1, \cdots, i_D) \; ; \; i_d \in \{1, \cdots, n_d\} \; \forall d = 1, \cdots, D\}$ denotes the set of D-uplets for the dimensions $n_1, n_2, \cdots, n_D$. For each of these D-uplets, we observe a noisy binary values $y_t \in \{0, 1\}$ indexed by $t \in \Omega$. These observations can also be represented as a noisy binary tensor $Y \in \{0, 1\}^{n_1 \times \cdots \times n_D}$ where $n_1, n_2, \cdots, n_D$ correspond to the tensor dimensions. Typically, $D = 2$ will correspond to binary matrices, and $D = 3$ to third-order tensors as used in database factorization models such as RESCAL [20]. Our objective is to predict the value of some specific entries in the tensor, which can be understood as detecting which entries in the tensor are outliers y_t. To do this, we estimate a tensor $Z(\theta) \in \mathbb{R}^{n_1 \times \cdots \times n_D}$ of log-odds parameterized by $\theta \in \Theta$.

We formulate the problem as an empirical loss minimization. In the training phase, the empirical loss $\mathcal{L}$ is minimized with respect to the parameter vector θ:

$$\min_{\theta \in \Theta} \mathcal{L}(\theta) \qquad \mathcal{L}(\theta) := \sum_{t \in \Omega} \ell(y_t, z_t(\theta)) \tag{1}$$

where $\ell(y, z) = -y\sigma \log(z) - (1 - y) \log(1 - \sigma(z))$ with σ representing the sigmoid function: $\sigma(z) := \frac{1}{1+e^{-z}}$. The predicted tensor $Z(\theta) := \{z_t(\theta)\}_{t \in \Omega}$ is assumed to be a factored representation, i.e. it has a low rank structure. For clarity, we consider only multi-linear models based on the PARAFAC [6] tensor parametrization.[1] The predictions z_t are obtained by the multilinear product of rank-K factors represented in the rows of matrices $\Theta_d \in \mathbb{R}^{n_d \times K}$, $d \in \{1, \cdots, D\}$:

$$\mathbf{z}_t(\theta) \quad := \quad \sum_{k=1}^{K} \prod_{d=1}^{D} \theta^{(d)}_{t_d k} := \left\langle \boldsymbol{\theta}^{(1)}_{t_1}, \cdots, \boldsymbol{\theta}^{(D)}_{t_D} \right\rangle \quad .$$

For matrices ($D = 2$), this model is a special case of exponential-family PCA [5] where the link function is logistic. For $D = 3$, this model has been studied in the context of multi-relational knowledge bases factorization [14, 18]. To solve Equation (1), several optimization methods have been proposed in the literature. Alternating optimization, gradient descent and stochastic gradient descent:

- The gradient descent algorithms are only based on the minimization of $\mathcal{L}$ by doing small steps in the direction of the gradients. Since the loss is differentiable, we derive its gradient in closed form and use a generic software to choose the optimal descent direction. In the experiments below, we use Marc Schmidt's `minFunc` Matlab function.[2]
- The alternating optimization procedure, also called block coordinate descent, consists in minimizing the loss $\mathcal{L}$ with respect to the i-th component Θ_i, keeping all the other components Θ_j, $j \neq i$ fixed. This optimization makes use of existing optimized linear logistic regression algorithms. The optimization procedure is obtained through a round-robin schedule. This approach is simple to implement, but it involves an inner loop since linear logistic regression algorithms are also based on gradient minimization.
- For large scale optimization, there is a growing interest in stochastic gradient descent algorithms since every gradient computation can potentially be too expensive. It consists in computing the gradients for a subset of the observations.

For highly sparse matrices where the number of zeros is much larger than the number of ones, these approaches do not scale well with the dimension of the tensor. For each of these algorithms, the time to make one function evaluation is the key bottleneck. The gradient descent algorithm requires to sum $|\Omega|$ elements (one per possible prediction). One step of the alternating optimization procedure is computationally costly because it requires to solve a linear logistic regression with $|\Omega|$ observations. The stochastic gradient descent algorithm seems to be better as each iteration is very fast, but it still requires $|\Omega|$ iteration to do one pass through the data. For some problems, good predictive performances are obtained even before the first pass

[1] We can easily adapt this work to more general decompositions such as Tucker or RESCAL, or even to convex loss functions using trace-norm regularization.

[2] Can be found at `http://www.di.ens.fr/~mschmidt/Software/minFunc.html`.

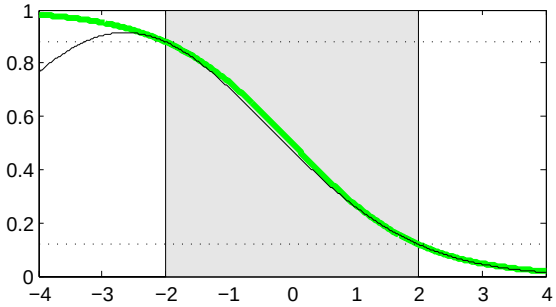

Figure 1: Comparison of the logistic function and the Gaussian distribution. The Gaussian distribution (the mean is -2.61 and the variance is 5.25) approximates well the logistic function in a range of values (gray area in the range [-2,2]) that corresponds to probabilities between 0.12 and 0.88 (dotted lines). In most of recommender systems applications, the probability of correctly predicting user choice is in this range, which explains that RMSE loss is reasonable.

through the data has been completed, but this case is relatively rare in practice. Our objective is to study methods that have a sub-linear complexity in the number of training sample, i.e. we can take into account *all* the $|\Omega|$ training samples while having a complexity that scales only in the number $n^+ = |\Omega^+|$ of non-zero elements. In the following we show that this complexity can be obtained by using a quadratic approximation to the logistic loss, leading to considerable speedup in our experiments.

3 QUADRATIC LOSS: FAST BUT OFTEN INACCURATE

As illustrated in Figure 1, the logistic loss can be reasonably approximated by a quadratic function, so the Root Mean Square Error (RMSE) should be a a good surrogate function to minimize. A naive computation of the square loss $\mathcal{L}(\theta) = \sum_{t\in\Omega}(y_t - z_t(\theta))^2$ would require $O(KD\prod_d n_d)$ operations, since there are $\prod_d n_d$ possible predictions, but simple algebra shows that it is equal to:

$$\mathcal{L}(\theta) = \sum_{t\in\Omega}(y_t - z_t)^2 \quad (2)$$

$$= n^+ - 2\sum_{t\in\Omega^+} z_t(\theta) + \sum_{k=1}^{K}\sum_{k'=1}^{K}\prod_{d=1}^{D} M^{(d)}_{kk'} \quad (3)$$

where the $K \times K$ matrices $M^{(d)}$ are defined by $M^{(d)}_{kk'} := \sum_{j=1}^{d} \theta^{(d)}_{jk}\theta^{(d)}_{jk'}$. Hence, for tensors of high dimension, we get a significant speedup as Equation (3) can be computed in $O(Kn^+ + K^2\sum_d n_d)$ operations.As an example, assume one wishes to compute the loss of a $1000 \times 1000 \times 1000$ tensor containing 10^5 entries and the low-rank approximation has rank $K = 100$. Then, we can see that there are 3.10^{11} basic operations in the formula of Equation (2), while the formula of Equation (3) contains 6.10^7 basic operations. This means that it will be 5000 times faster to compute exactly the same quantity!

If we minimize the loss $\mathcal{L}(\theta)$ with respect to $\boldsymbol{\theta}$ using block-coordinate descent, the iterations end up being least squares problem with a per-iteration complexity that scales linearly with the number of positive examples only. This corresponds exactly to the iTALS algorithm [22], which is the tensor generalization of the alternating least squares algorithm of [9]. In our experiments, we used gradient descent to minimize the objective function, using Equation (3) to compute the gradient efficiently (the complexity is the same as the function evaluation).

4 SPEED-ACCURACY TRADE-OFF BY BOUNDING SPLITS

4.1 UPPER BOUNDING THE LOSS

To speed-up computation, we minimize a quadratic upper bound to the logistic loss. We use Jaakkola's bound to the logistic loss [10]:

$$\log(1+e^z) \leq \lambda(\xi)(z^2 - \xi^2) + \frac{1}{2}(z-\xi) + \log(1+e^\xi) \ ,$$

where $\lambda(\xi) := \frac{1}{2\xi}(\frac{1}{1+e^\xi} - \frac{1}{2})$. We keep the same value for ξ for all the elements of the tensor Z, so that the upper bound has exactly the form required to apply the computational speedup described in the previous section.

$$\bar{\mathcal{L}}(\theta,\xi) = \lambda(\xi)\sum_{t\in\Omega}\left(z_t - \frac{2y_t}{4\lambda(\xi)}\right)^2 + c(\xi)$$

where $c(\xi)$ is a constant function that does not depend on θ:

$$c(\xi) = |\Omega|\left(\log(1+e^\xi) - \lambda(\xi)\xi^2 - \frac{1}{2}\xi - \frac{1}{16\lambda(\xi)}\right)$$

The optimization of this bound with respect to ξ gives exactly the Frobenius norm of the tensor:

$$\sum_{t\in\Omega} z(\theta)^2 = \arg\min_{\xi} \bar{\mathcal{L}}(\theta,\xi) \ . \quad (4)$$

Note that $Z(\theta)$ is low rank in general. This means that which can also be efficiently computed using the third term of Equation (3). We have now an upper bound to the original loss $\mathcal{L}$ that needs to be minimized:

$$\mathcal{L}(\theta) \leq \bar{\mathcal{L}}^{\Omega}(\theta,\xi) \ . \quad (5)$$

As usual with bound optimization, we alternate between two steps: 1) minimizing the bound with respect to the variational parameter ξ using closed form updates (e.g. when using Jaakkola's bound) or dichotomic search where no closed form solution exists; and 2) minimizing the

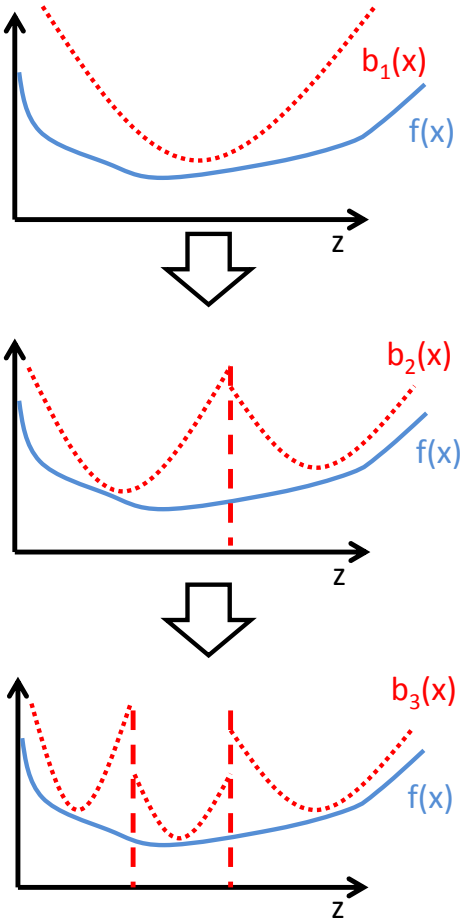

Figure 2: Illustration of the idea of bound refinement. The main idea is that computing the integral of the quadratic upper bound (such as b_1 in the top graph) is much faster than computing the integral of f directly. To improve the accuracy, we use piecewise bounds. To choose the domain of the pieces, we use a greedy algorithm that identifies the partition that leads to the diminished upper bound (leading to the upper bounds b_2 and b_3.

bound with respect to $\boldsymbol{\theta}$ using a standard gradient descent algorithm. This algorithm is sometimes referred as Majorization-Minimization algorithm in the literature [16]. The algorithm minimizes the loss with a complexity per iteration equal to $O\left(Kn^+ + K^2 \sum_d n_d\right)$ In the experiments, this algorithm is called *Quad-App*. It is detailed in Algorithm 2.

4.2 SPLIT THE DATA TO IMPROVE ACCURACY

The drawback of the previous approach, even with one or two orders of magnitude speedups, the resulting quadratic approximation can be quite loose for some data, and the accuracy of the method can be too low. We give here a family of approximation that interpolates between this fast but inaccurate quadratic approximation and the slow but

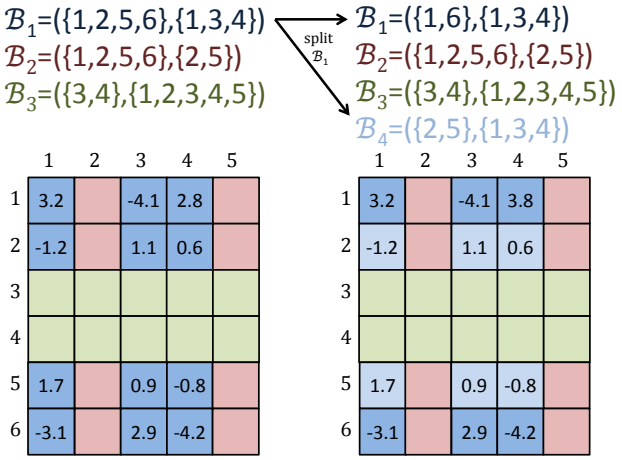

Figure 3: Example of matrix split to improve upper bound accuracy. Colors identify blocks. Numbers in the matrix represent predictions. The block 1 is decomposed into blocks 1 and 4, both having a small variance on the absolute value of the predictions.

exact minimization of the non-quadratic loss.

We propose to take advantage of the speedup due to the quadratic upper bound by applying it on a partition of the the original tensor: we select a set $\mathcal{B} = \{B_1, \cdot, B_{|\mathcal{B}|}\}$ of disjoint *blocks* that partitions the space of possible observation indices Ω. On each of these blocks, the bounding technique described in the previous section is applied, the main difference being that the minimization with respect to θ is done jointly on all the blocks. This process is illustrated in Figure 2. Formally, each block B_b, $b \in \{1, \cdots, |\mathcal{B}|\}$ is identified by D sets of indices which represent the dimensions that are selected in the given block b. The split of these indices are illustrated in a toy matrix example in Figure 3. Blocks for tensors are computed the same way as matrices, but the depth indices are also splitted: At each refinement step, we choose to partition the rows indices, column indices or depth indices.

To select the blocks, we use a greedy construction of the blocks: starting with a single block containing all the indices, i.e. $\mathcal{B} = \{\Omega\}$, we iteratively refine the blocks using the following two-step procedure, called `RefineBlocks`:

1. select the block b to split that has the maximal variance in the absolute values of the predictions $|z_t|$;
2. split the block b into two block so that the variance of the absolute values of the predictions $|z_t|$ is minimized.

An example of such a split is shown in Figure 3. This method is fully described in Algorithm 1, which uses Algorithm 2 as a sub-program to learn the parameters for a

Algorithm 1 Iterative block splitting: $\min_{\theta,\mathcal{B}} \bar{\mathcal{L}}(\theta, \mathcal{B}, Y)$

$\hat{\boldsymbol{\theta}} =$ `UBAdaptiveMinimization`$(Y, \boldsymbol{\theta}^{(0)}, \varepsilon)$
Inputs: tensor Y, initial $\boldsymbol{\theta}^{(0)}$, tolerance ε
Outputs: latent factors $\hat{\boldsymbol{\theta}} = \{\hat{\Theta}_1, \cdots, \hat{\Theta}_D\}$
Initialize blocks $\mathcal{B}^{(0)} = \{\Omega\}$,
for $i = 1, 2, \cdots$ until improvement less than ε **do**
  $\boldsymbol{\theta}^{(i)} =$ `UBMinimization`$(Y, \boldsymbol{\theta}^{(i-1)}, \varepsilon/2, \mathcal{B}^{(i-1)})$
  $\mathcal{B}^{(i)} \leftarrow$ `RefineBlocks`$(\mathcal{B}^{(i-1)}, \boldsymbol{\theta}^{(i)})$
end for
$\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(i)}$

Algorithm 2 `UBMinimization` $\min_\theta \bar{\mathcal{L}}(\theta, \mathcal{B}, Y)$

$\hat{\boldsymbol{\theta}} =$ `UBMinimization`$(Y, \boldsymbol{\theta}^{(0)}, \varepsilon, \mathcal{B})$
Inputs: tensor Y, initial $\boldsymbol{\theta}^{(0)}$, tol. ε, blocks $\mathcal{B}$
Outputs: latent factors $\hat{\boldsymbol{\theta}} = \{\hat{\Theta}_1, \cdots, \hat{\Theta}_D\}$
for $i = 1, 2, \cdots$ until improvement less than ε **do**
  for $b = 1, 2, \cdots, |\mathcal{B}|$ **do**
    $\xi_b^{(i)} \leftarrow \arg\min_{\xi_b} \bar{\mathcal{L}}_b(\boldsymbol{\theta}^{(i-1)}, \xi_b, \mathcal{B}_b)$
  end for
  $\boldsymbol{\theta}^{(i)} \leftarrow \arg\min_{\boldsymbol{\theta}} \sum_b \bar{\mathcal{L}}_b(\boldsymbol{\theta}, \xi_b^{(i)}, \mathcal{B}_b)$
end for
$\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(i)}$

fixed set of blocks, following the approach described in the previous section. There is a tradeoff between optimizing the bound using Algorithm 2 and refining the partition $\mathcal{B}$ in Algorithm 1. A simple strategy that worked well in practice was to refine the partition when the upper bound minimization did not improve more than a given tolerance level $\varepsilon/2$ in two successive iterations. This piecewise refinement strategy is called *PW Quad-App* in the experimental section.

We also introduced a slight variant of this basic algorithm, where we compute the exact logistic loss in blocks that are sufficiently dense, i.e. when computing Equation (3) requires more iterations than computing the loss in the classical way (Equation (3)). This condition is verified when $n^+(B) \geq \prod_{d=1}^{D} n_d^{(B)} - K \sum_{d=1}^{D} n_d^{(B)}$, where $n^+(B)$ and $n_d^{(B)}$ correspond to the number of non-zero elements and dimensions of the block tensor B. We call this variant *PW Quad-App + Logistic*.

5 EXPERIMENTS AND RESULTS

In this section, to evaluate the performances of our framework, we conducted experiments on both synthetic and real datasets.

Synthetic Data Experiments To explore the speed-accuracy tradeoff, we generated different binary matrices Y by randomly sampling noisy low-rank matrices $X = UV + E$ where $U \in \mathbb{R}^{n_1 \times r}$ and $V \in \mathbb{R}^{r \times n_2}$ are generated using independent standard normal variables and $E \in \mathbb{R}^{n_1 \times n_2}$ is a normally distributed Gaussian noise with standard deviation σ. To create the binary matrix $Y \in \{0, 1\}^{n_1 \times n_2}$, we round the values of X using a high percentile of X as a threshold to produce a heavy tendency towards the negative class. We learn an estimation $\hat{X}$ of the original matrix X on the data matrix Y assumed to be fully observed and compute the RMSE on the recovery of X, i.e. RMSE=$\|X - \hat{X}\|_F$. We measure the running time of each of the methods to understand their scalability to large datasets.

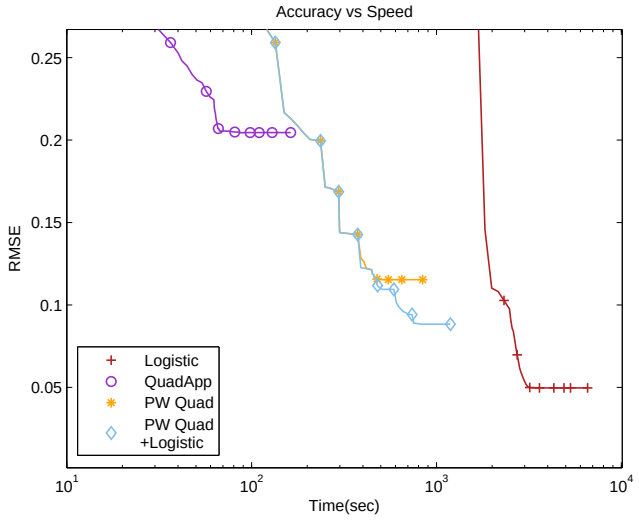

Figure 4: Matrix recovery results on simulation data with size 10000×5000, sparsity $\%0.1$ and noise $\sigma = 0.1$. Markers are plotted at iterations 10, 17, 28, 35, 52, 63 and 73 (these times correspond to the refinement of the piecewise bound).

The timing and accuracy performances of our methods on simulation data with various dimensions, noise levels and sparsity percentages are shown in Table 1. These results are averaged over 10 runs and we choose rank $r = 5$ for every simulation. Here, the baselines are logistic loss and quadratic loss. The logistic loss gives much smaller error rates than the quadratic loss (EUC-Full and EUC-Fast) and quadratic approximation, especially when the noise level is low. However, minimizing the logistic loss requires considerably more time than the alternative approaches as the problem size grows. These experiments highlight the fact that our unified framework for quadratic loss gives a significant improvement over logistic loss in terms of runtime performance. We also observe that the piecewise quadratic bounding technique has better predictive performance than the quadratic approximation, along with a huge speedup when we compare it to the time to train the model using the logistic loss. On Figure 4, we plotted the test RMSE with respect to the CPU time (each marker corresponds to an iteration). Because the error rate of quadratic loss is

Table 1: Evaluation results of the synthetic experiments in terms of seconds for runtime and RMSE for matrix recovery.

			Methods									
			EUC-Full		EUC-Fast		Logistic		Quad-App		PW QuadApp	
Noise Level	Dimension	Sparsity	RMSE	Time	RMSE	Time	RMSE	Time	RMSE	Time	RMSE	Time
Low Noise $\sigma = 0.1$	$n_1 = 100$ $n_2 = 50$	10%	0.6970	60.45	0.6970	0.50	0.3561	103.13	0.6421	0.58	0.4377	6.16
		1%	0.6792	55.57	0.6792	0.46	0.1095	90.65	0.4568	0.56	0.1918	3.19
	$n_1 = 1000$ $n_2 = 500$	1%	0.7251	5295.7	0.7251	63.10	0.1563	7216.2	0.7054	75.49	0.3790	421.11
		0.1%	0.7251	5248.3	0.7251	42.90	0.2126	6605.6	0.7067	62.90	0.5247	301.35
	$n_1 = 10000$ $n_2 = 5000$	0.1%	0.4483	84950	0.4483	1289.2	0.0497	68199	0.2052	1353.8	0.0911	6683.9
		0.01%	0.4217	86109	0.4217	803.2	0.0329	66482	0.1814	1049.3	0.0583	4271.4
High Noise $\sigma = 2.0$	$n_1 = 100$ $n_2 = 50$	10%	2.8989	59.06.	2.8989	0.48	1.2789	94.49	1.8639	0.73	1.3212	10.18
		1%	2.8821	44.96	2.8821	0.37	0.2377	59.21	0.4589	0.54	0.2622	7.14
	$n_1 = 1000$ $n_2 = 500$	1%	2.6705	7070.3	2.6705	110.87	0.3371	6592.0	0.4052	132.55	0.3659	943.67
		0.1%	2.5116	7276.2	2.5116	109.99	0.0563	6503.1	0.1304	120.83	0.1078	783.91
	$n_1 = 10000$ $n_2 = 5000$	0.1%	1.9990	97084	1.9990	1486.1	0.2312	66124	0.2604	1523.0	0.2374	7352.8
		0.01%	1.7416	83846	1.7416	1183.0	0.1332	65664	0.1661	1589.6	0.1414	6613.1

considerably bigger than the other methods, we show the results of logistic loss, quadratic approximation and piecewise methods in this figure. It is interesting to see that the piecewise quadratic bound reaches its goal of interpolating the performances between the fast but inaccurate quadratic approximation, and the slow but accurate logistic loss minimization: On a wide range of times (from 2 minutes to 1 hour), the piecewise quadratic bound gives the best performances. It is worth to note that the slight modification that was introduced to perform exact logistic on the smallest pieces improves performances when the methods have nearly converged (after 60 iterations). Note that this experiment was done on matrices, but the differences are even bigger for tensors of high order.

Real Data Experiments In order to evaluate the performances of our methods, we designed link-prediction experiments on standard multi-relational datasets: *Nations* that groups 14 countries (entities) with 56 binary relation types (like 'economic aid', 'treaties' or 'rel diplomacy') representing interactions among them; *Kinships* which is the complex relational structure of Australian tribes' kinship systems. In Kinships dataset, 104 tribe members were asked to provide the kinship terms they used for one another and this results in graph of 104 entities and 26 relation types, each of them depicting a different kinship term. And *UMLS* that contains data from the Unified Medical Language System semantic work used in [11]. This dataset consists in a graph with 135 entities (high-level concepts like 'Disease or Syndrome', 'Diagnostic Procedure') and 49 relation types (verbs depicting causal influence between concepts like 'affect' or 'cause'). In the end, these datasets results in tensors $Y \in \{0,1\}^{14\times14\times56}$, $Y \in \{0,1\}^{104\times104\times26}$ and $Y \in \{0,1\}^{135\times135\times49}$ respectively.

Then, we compared the Area Under the Receiver Operating Characteristic Curve (AUC) and runtime in seconds of piecewise methods to the results of quadratic approximation and logistic loss and also the results of RESCAL [20], SME [2] and LFM [11] that have the best published results on these benchmarks in terms of AUC.

In addition, we test the performances of these methods on three datasets in matrix form: MovieLens [3], Last FM [4] [4] and Sushi Preference [12]. MovieLens dataset contains movie ratings of approximately 1682 movies made by 943 MovieLens users and results in matrix $Y \in \{0,1\}^{943\times1682}$. The Last FM dataset consists of music artist listening information from a set of 1892 users from *Last.fm* online music system. We construct a binary matrix $Y \in \{0,1\}^{1892\times17632}$ from this dataset that contains the artists listened by each user. Lastly, the *Sushi Preference Data Set* includes 4950 users' responses of preference in 100 different kinds of *sushi*. In this dataset, the most disliked kind of sushi represented by 0 and the most preferred one is represented by 1. Eventually, sushi dataset results in matrix $Y \in \{0,1\}^{100\times4950}$.

For the results given in Table 2, we performed 10-fold cross validation and averaged over 10 random splits of the datasets. In addition, we select the optimal regularization parameter λ^* by searching over the set $\{0.01, 0.05, 0.1, 0.5, 1\}$ that maximizes the AUC and we computed rank-20 decomposition of these datasets in order to get comparable results to RESCAL, SME and LFM. The time and accuracy comparisons are given in Table 2 in terms of seconds for time and AUC metric for predicting missing values. These results demonstrate that logistic loss improves accuracy over RESCAL, SME and LFM

[3] www.grouplens.org/node/73

[4] www.grouplens.org/datasets/hetrec-2011/

Table 2: Evaluation results obtained by our approaches, RESCAL [20], SME [2] and LFM [11] on the given datasets.

		Datasets					
	Methods	Nations	Kinships	UMLS	MovieLens	Last FM	Sushi
AUC	EUC-Full	0.7536	0.8193	0.8205	0.8511	0.8965	0.8199
	EUC-Fast	0.7536	0.8193	0.8205	0.8511	0.8965	0.8199
	Logistic	0.9253	0.9592	0.9795	0.9848	0.9454	0.9513
	Quad-App	0.8635	0.9087	0.9169	0.8916	0.9042	0.9078
	PW QuadApp	0.9038	0.9213	0.9387	0.9490	0.9272	0.9200
	PW Quad+Logistic	0.9122	0.9416	0.9566	0.9781	0.9381	0.9373
	RESCAL [20]	0.8400	0.9500	0.9800	0.9601	0.9257	0.9481
	SME [2]	0.8830	0.9070	0.9830	0.9144	0.9328	0.9177
	LFM [11]	0.9090	0.9460	0.9900	0.9790	0.9401	0.9598
Time (sec)	EUC-Full	922.87	8793.8	81039	103454	701819	13490.8
	EUC-Fast	18.37	80.95	167.67	344.81	3688.74	142.63
	Logistic	1483.7	10374.2	75489	111257	721994	12704.6
	Quad-App	18.07	97.13	187.1	431.35	1839.16	142.06
	PieceQuadApp	32.52	169.35	922.65	1095.94	2792.18	643.56
	PW Quad+Logistic	59.71	651.12	1035.47	1349.6	4065.12	755.72
	RESCAL [20]	626.40	3714.6	4142.05	6786.23	9861.50	2632.1
	SME [2]	32.11	135.9	513.03	749.07	1627.56	279.38
	LFM [11]	64.63	1446.22	5097.5	8265.56	13058.1	3438.07

in Nations, Kinships, MovieLens and Last FM datasets while it reaches almost the same score for UMLS and Sushi datasets. On the other side, piecewise methods provide very close approximation to logistic loss on all these datasets and they have a significant advantage in terms of runtime over the other methods. They take a small fraction of logistic loss' running time, especially for large datasets.

6 RELATED WORK

In order to deal with learning on various forms of structured data such as large-scale knowledge bases, time-varying networks or recommendation data, tensor factorizations have become increasingly popular [3, 21, 23]. Recently, Nickel et al presented RESCAL [20], an upgrade over previous tensor factorization methods, which has been shown to achieve state-of-the-art results for various relational learning tasks such as link prediction and entity resolution. Independently, a similar logistic extension of the RESCAL factorization has been proposed in [14]. [21] is an extension to the RESCAL algorithm on the YAGO ontology and is based on alternating least-squares updates of the factor matrices, has been shown to scale up to large knowledge bases via exploiting the sparsity of relational data. Among the existing works, [18] is the most similar work with our, which is the logistic extension of RESCAL. It demonstrates that the logistic loss improves the prediction results significantly but their algorithm requires to compute the dense matrix and cannot scale to large data. In RESCAL, entities are modeled by real-valued vectors and relations by matrices. Bordes *et al* has further improved this idea in the Structured Embeddings (SE) framework [3] by learning a model to represent elements of any knowledge base (KB) into a relatively low dimensional embedding vector space by tensor factorization method. Latent Factor Model (LFM) [11] is based on a bilinear structure, which captures various orders of interaction of the data, and also shares sparse latent factors across different relations. In [2], they present a new neural network designed to embed multi-relational graphs into a flexible continuous vector space via a custom energy function (SME) in which the original data is kept and enhanced. In all of these studies [3, 11, 2], the data is extremely skewed i.e., the number of negative examples $\gg$ the number of positive examples. To overcome the sparsity, they first select a positive training triplet at random, then create a negative triplet by sampling an entity from the set of all entities. Unlike these approaches, we argue that it is in general more appropriate to consider *all* the negative examples.

Maaten et al [15] derive an upper bound to logistic loss which can be minimized as surrogate loss for linear predictors on binary labels. Khan et al [13] used Jaakkola's bound for binary observations and Bohning's bound for multinomial observations [1]. Our work can be easily extended to take into account Bohning's bound. In addition, piecewise bounds have the important property: reducing the error as the number of pieces increase. Marlin et al. proposed an improvement on the logistic-loss with piecewise linear bounds, but this is a local approach and does not apply in our setting since we need a global quadratic bound to apply the squared norm trick [17].

7 CONCLUSION

There were several important techniques used in this paper: 1) the decomposition of the loss into a small positive part and a large but structured negative space; 2) the use of the squared norm trick that reduces the complexity of squared loss computation and 3) the use of the partitioning technique to gradually reduce the gap introduced by the usage of quadratic upper bounds for non-quadratic losses, particularly useful in the case of binary or count data. This combination of techniques can be applied in a broad range of other problems, such as probabilistic CCA, collective-matrix factorization, non-negative matrix factorization, as well as non-factorial models such as time series [7].

References

[1] D. Bohning. Multinomial logistic regression algorithm. *Annals of the Inst. of Statistical Math.*, 44:197200, 1992.

[2] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning: Special Issue on Learning Semantics*, 2013.

[3] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *AAAI*, 2011.

[4] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, New York, NY, USA, 2011. ACM.

[5] Michael Collins, Sanjoy Dasgupta, and Robert E Schapire. A generalization of principal component analysis to the exponential family. In *NIPS*, volume 13, page 23, 2001.

[6] Richard A Harshman and Margaret E Lundy. The parafac model for three-way factor analysis and multidimensional scaling. *Research methods for multimode data analysis*, pages 122–215, 1984.

[7] Joo F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *ECCV (4)*, pages 702–715, 2012.

[8] Balzs Hidasi and Domonkos Tikk. Fast ALS-Based tensor factorization for context-aware recommendation from implicit feedback. ECML PKDD, page 6782, Bristol, UK, 2012.

[9] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, page 263272, 2008.

[10] T Jaakkola and M Jordan. A variational approach to bayesian logistic regression models and their extensions. In *Sixth International Workshop on Artificial Intelligence and Statistics*, 1997.

[11] Rodolphe Jenatton, Nicolas Le Roux, Antoine Bordes, and Guillaume Obozinski. A latent factor model for highly multi-relational data. In *NIPS*, pages 3176–3184, 2012.

[12] Toshihiro Kamishima. Nantonac collaborative filtering: Recommendation based on order responses. In *Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining*, page 583588. ACM, August 2003.

[13] Mohammad Emtiyaz Khan, Benjamin M. Marlin, Guillaume Bouchard, and Kevin P. Murphy. Variational bounds for mixed-data factor analysis. In *NIPS*, pages 1108–1116, 2010.

[14] Ben London, Theodoros Rekatsinas, Bert Huang, and Lise Getoor. Multi-relational learning using weighted tensor decomposition with modular loss. *arXiv preprint arXiv:1303.1733*, 2013.

[15] Laurens Maaten, Minmin Chen, Stephen Tyree, and Kilian Q Weinberger. Learning with marginalized corrupted features. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, page 410418, 2013.

[16] Julien Mairal. Optimization with first-order surrogate functions. In *ICML*, 2013.

[17] Benjamin M. Marlin, Mohammad Emtiyaz Khan, and Kevin P. Murphy. Piecewise bounds for estimating bernoulli-logistic latent gaussian models. In *ICML*, pages 633–640, 2011.

[18] Maximilian Nickel and Volker Tresp. Logistic tensor factorization for multi-relational data. *CoRR*, abs/1306.2084, 2013.

[19] Maximilian Nickel and Volker Tresp. Tensor factorization for multi-relational learning. In *Machine Learning and Knowledge Discovery in Databases*, page 617621. Springer, 2013.

[20] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, page 809816, 2011.

[21] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing YAGO: scalable machine learning for linked data. In *WWW*, page 271280, Lyon, France, 2012.

[22] Istvn Pilszy, Dvid Zibriczky, and Domonkos Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *ACM RecSys*, page 7178, 2010.

[23] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, page 811820, 2010.

[24] Matthias Seeger and Guillaume Bouchard. Fast variational bayesian inference for non-conjugate matrix factorization models. *Journal of Machine Learning Research - Proceedings Track*, 22:1012–1018, 2012.

[25] Ajit P. Singh and Geoffrey J. Gordon. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, page 650658. ACM, New York, NY, USA, 2008.