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 $m$ of non-zero elements in the tensor is much smaller than the total number $n$ of possible entries in the tensor. This creates a problem for large tensors because the computation of the logistic loss has a linear time complexity with $n$. In this work, 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 $O(m)$, as proposed earlier in the restricted 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 trade-off 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 $\text{isFather}(x,y)$ can be considered as a positive example and automatically induces $m - 1$ negative examples of the form $\text{not(isFather}(x,z))$, 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 of 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 [13, 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 \( n^+ \ll n^- \).

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 \( L \) is minimized with respect to the parameter vector \( \theta \):

\[
\min_{\theta \in \Theta} L(\theta) \quad L(\theta) := \sum_{t \in \Omega} \ell(y_t, z_t(\theta)) \tag{1}
\]

where \( \ell(y, z) = -y \log(\sigma(z)) - (1-y) \log(1-\sigma(z)) \) with \( \sigma \) representing the sigmoid function: \( \sigma(z) := 1/(1+e^{-z}) \). 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. The predictions \( z_t \) are obtained by the multilinear product of rank-\( K \) factors.

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 \( L \) 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.
- The alternating optimization procedure, called block coordinate descent, consists in minimizing the loss \( L \) with respect to the \( i \)-th component \( \Theta_i \), keeping all the other components \( \Theta_j, j \neq i \) 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 \(|\Omega|\) iterations to do one pass through the data. For some problems, good predictive performances are obtained even before the first pass.
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:

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

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

where the $K \times K$ matrices $M^{(d)}$ are defined by $M_{kk'}^{(d)} = \frac{1}{K} \sum_{d=1}^d \theta_{j,k}^{(d)} \theta_{j,k'}^{(d)}$. 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 $10^9$ entries and the low-rank approximation has rank $K = 100$. Then, we can see that there are $3 \times 10^{11}$ basic operations in the formula of Equation (3), 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 $\theta$ 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]:

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

where $\lambda(\xi) := \frac{1}{2}(1 + e^{\xi})$ and $\xi$ is a variational parameter. We keep the same value for $\xi$ 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.

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

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

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

The optimization of this bound with respect to $\xi$ gives exactly the Frobenius norm of the tensor:

$$\sum_{t \in \Omega} z_t(\theta)^2 = \min_\xi \tilde{\mathcal{L}}(\theta, \xi) \quad .$$

Note that $Z(\theta)$ is low rank in general. This means that it 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:

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

As usual with bound optimization, we alternate between two steps: 1) minimizing the bound with respect to the variational parameter $\xi$ using closed form updates (e.g. when using Jaakkola’s bound) or dichotomic search where
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 \( b_3 \)).

no closed form solution exists; and 2) minimizing the bound with respect to \( \theta \) using a standard gradient descent algorithm. This algorithm is sometimes referred as Majorization-Minimization algorithm in the literature [10]. The algorithm minimizes the loss with a complexity per iteration equal to \( O(Kn^2 + K^2 \sum d n_d) \) In the experiments, this algorithm is called \textit{Quad-App}. It is detailed in Algorithm [2].

\subsection{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 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 \( B = \{ B_1, \cdots, B_{\|B\|} \} \) of disjoint blocks that partitions the space of possible observation indices \( \Omega \). 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 \( \theta \) is done jointly on all the blocks. This process is illustrated in Figure 2. Formally, each block \( B_b, b \in \{1, \cdots, \|B\|\} \) 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. \( B = \{\Omega\} \), we iteratively refine the blocks using the following two-step procedure, called \textit{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 [11] which uses Al-
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 $\sigma$. 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. $\text{RMSE} = \|X - \hat{X}\|_F$. 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 $10000 \times 5000$, 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

---

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

1. $\hat{\theta} = \text{UBAdaptiveMinimization}(Y, \theta^{(0)}, \varepsilon)$
2. **Inputs**: tensor $Y$, initial $\theta^{(0)}$, tolerance $\varepsilon$
3. **Outputs**: latent factors $\hat{\theta} = \{\hat{\Theta}_1, \cdots, \hat{\Theta}_D\}$
4. Initialize blocks $B^{(0)} = \{\Omega\}$
5. for $i = 1, 2, \cdots$ until improvement less than $\varepsilon$
   6. $\theta^{(i)} = \text{UBMinimization}(Y, \theta^{(i-1)}, \varepsilon/2, B^{(i-1)})$
   7. $B^{(i)} \leftarrow \text{RefineBlocks}(B^{(i-1)}, \theta^{(i)})$
8. end for
9. $\theta = \theta^{(i)}$

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

1. $\hat{\theta} = \text{UBMinimization}(Y, \theta^{(0)}, \varepsilon, B)$
2. **Inputs**: tensor $Y$, initial $\theta^{(0)}$, tol. $\varepsilon$, blocks $B$
3. **Outputs**: latent factors $\hat{\theta} = \{\hat{\Theta}_1, \cdots, \hat{\Theta}_D\}$
4. for $i = 1, 2, \cdots$ until improvement less than $\varepsilon$
5. for $b = 1, 2, \cdots, |B|$ do
6. $\hat{\xi}_b^{(i)} \leftarrow \arg \min_{\xi_b} \hat{\bar{L}}_b(\theta^{(i-1)}, \xi_b, B_b)$
7. end for
8. $\theta^{(i)} \leftarrow \arg \min_{\theta} \sum_b \hat{\bar{L}}_b(\theta, \hat{\xi}_b^{(i)}, B_b)$
9. end for
10. $\theta = \theta^{(i)}$

---

---

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

<table>
<thead>
<tr>
<th>Noise Level</th>
<th>Dimension</th>
<th>Sparsity</th>
<th>EUC-Full</th>
<th>EUC-Fast</th>
<th>Logistic</th>
<th>Quad-App</th>
<th>PW QuadApp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Noise</td>
<td>( n_1 \times n_2 )</td>
<td>10%</td>
<td>0.6970</td>
<td>0.6970</td>
<td>0.3561</td>
<td>0.6421</td>
<td>0.4377</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1%</td>
<td>0.6792</td>
<td>0.6792</td>
<td>0.3050</td>
<td>0.4568</td>
<td>0.1918</td>
</tr>
<tr>
<td>High Noise</td>
<td>( n_1 \times n_2 )</td>
<td>1%</td>
<td>0.7251</td>
<td>0.7251</td>
<td>0.7163</td>
<td>0.7067</td>
<td>0.5247</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1%</td>
<td>0.4483</td>
<td>0.4483</td>
<td>0.0497</td>
<td>0.2029</td>
<td>0.0911</td>
</tr>
</tbody>
</table>

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 \( Y \in \{0,1\}^{14 \times 14 \times 56} \), \( Y \in \{0,1\}^{104 \times 104 \times 26} \) and \( Y \in \{0,1\}^{135 \times 135 \times 49} \) 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] and Sushi Preference [12]. MovieLens dataset contains movie ratings of approximately 1682 movies made by 943 MovieLens users and results in matrix \( Y \in \{0,1\}^{943 \times 1682} \). 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 \( Y \in \{0,1\}^{1892 \times 17632} \) 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 \( Y \in \{0,1\}^{100 \times 4950} \).

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 \( \lambda^* \) 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 prediction.

\[ \text{www.grouplens.org/node/73} \]
\[ \text{www.grouplens.org/datasets/hetrec-2011/} \]

<table>
<thead>
<tr>
<th>Methods</th>
<th>Nations</th>
<th>Kinships</th>
<th>UMLS</th>
<th>MovieLens</th>
<th>Last FM</th>
<th>Sushi</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PW Quad+Logistic</td>
<td>0.8830</td>
<td>0.9070</td>
<td>0.9830</td>
<td>0.9144</td>
<td>0.9328</td>
<td>0.9177</td>
</tr>
<tr>
<td>SME [2]</td>
<td>0.8400</td>
<td>0.9500</td>
<td>0.9800</td>
<td>0.9601</td>
<td>0.9272</td>
<td>0.9481</td>
</tr>
<tr>
<td>LFM [11]</td>
<td>0.9090</td>
<td>0.9460</td>
<td>0.9900</td>
<td>0.9790</td>
<td>0.9401</td>
<td>0.9598</td>
</tr>
<tr>
<td>Time (sec)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PW Quad+Logistic</td>
<td>59.71</td>
<td>651.12</td>
<td>1035.47</td>
<td>1349.6</td>
<td>1483.7</td>
<td>64.63</td>
</tr>
<tr>
<td>RESCAL [20]</td>
<td>626.40</td>
<td>3714.6</td>
<td>4142.05</td>
<td>6786.23</td>
<td>9861.50</td>
<td>59.71</td>
</tr>
<tr>
<td>SME [2]</td>
<td>32.11</td>
<td>135.9</td>
<td>513.03</td>
<td>749.07</td>
<td>1095.94</td>
<td>279.38</td>
</tr>
<tr>
<td>LFM [11]</td>
<td>64.63</td>
<td>1446.22</td>
<td>5097.5</td>
<td>8265.56</td>
<td>13058.1</td>
<td>64.63</td>
</tr>
</tbody>
</table>

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, [13] 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 Bohnig’s bound for multinomial observations [11]. 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].

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, [13] 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 Bohnig’s bound for multinomial observations [11]. Our work can be easily extended to take into account Bohnig’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].

ACKNOWLEDGEMENTS

This work has been supported by the Fupol and Fusepool FP7 European project. We thank anonymous reviewers, Jean-Marc Andreoli and Jos-Antonio Rodriguez for their comments.

References