Influence-guided Data Augmentation for
Neural Tensor Completion

Sejoon Oh
soh337@gatech.edu
Georgia Institute of Technology
United States

Sungchul Kim, Ryan A. Rossi
sukim@adobe.com, ryrossi@adobe.com
Adobe Research
United States

Srijan Kumar
srijan@gatech.edu
Georgia Institute of Technology
United States

ABSTRACT
How can we predict missing values in multi-dimensional data (or tensors) more accurately? The task of tensor completion is crucial in many applications such as personalized recommendation, image and video restoration, and link prediction in social networks. Many tensor factorization and neural network-based tensor completion algorithms have been developed to predict missing entries in partially observed tensors. However, they can produce inaccurate estimations as real-world tensors are very sparse, and these methods tend to overfit on the small amount of data. Here, we overcome these shortcomings by presenting a data augmentation technique for tensors. In this paper, we propose DAIN, a general data augmentation framework that enhances the prediction accuracy of neural tensor completion methods. Specifically, DAIN first trains a neural model and finds tensor cell importances with influence functions. After that, DAIN aggregates the cell importance to calculate the importance of each entity (i.e., an index of a dimension). Finally, DAIN augments the tensor by weighted sampling of entity importances and a value predictor. Extensive experimental results show that DAIN outperforms all baseline methods in terms of enhancing imputation accuracy of neural tensor completion on four diverse real-world tensors. Ablation studies of DAIN substantiate the effectiveness of each component of DAIN. Furthermore, we show that DAIN scales near linearly to large datasets.

CCS CONCEPTS
• Computing methodologies → Neural networks; Factorization methods; • Information systems → Data mining.

KEYWORDS
Tensor, Tensor Completion, Neural Network, Data Augmentation, Data Influence, Recommender System, Deep Learning

ACM Reference Format:
Sejoon Oh, Sungchul Kim, Ryan A. Rossi, and Srijan Kumar. 2021. Influence-guided Data Augmentation for Neural Tensor Completion. In Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM ’21), November 1–5, 2021, Virtual Event, QLD, Australia. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482267

1 INTRODUCTION
We observe various tensors on the Web, including images and videos, numerical ratings, social networks, and knowledge bases. Most real-world tensors are very large and extremely sparse [26] (i.e., a few observed entries and many missing values). Thus, tensor completion, i.e., the task of predicting missing values in a tensor, has been actively investigated in diverse areas including recommender systems [8], social networks [6], traffic analysis [32], medical questionnaires [4], and computer vision [5]. For example, consider a movie rating tensor with three dimensions, namely users, movies, and time slices. Each tensor cell (i, j, k) contains the rating score given by user i to movie j during time slice k. The goal of tensor completion is to predict the rating scores of unobserved tensor cells.

Tensor factorization (TF) is a popular technique to predict missing values in a tensor, but most methods [15, 19, 25, 26, 28, 30] exhibit high imputation error while estimating missing values in a tensor. One of the reasons is that many TF models [15, 19, 25, 30] regard missing values in a tensor as zeros. Hence, if those models are trained with a sparse tensor, their predictions would be biased toward zeros, instead of the observed values. Other TF methods [7, 26, 28] improve their accuracy by focusing only on observed entries; however, they suffer from overfitting as the tensor is very sparse.

Neural network-based tensor completion methods [2, 21, 36, 37] have been proposed to enhance the estimation accuracy. They have strong generalization capability [24], and capture non-linearity hidden in a tensor. However, these methods still suffer from data sparsity, and this can become a bottleneck for neural tensor completion methods, which require a large amount of data for training [35]. Moreover, these existing methods cannot generate new data points for data augmentation to solve the sparsity issue.

In this paper, we propose a method that leverages the strength of neural tensor completion, and improves it through the utilization of data augmentation. Data augmentation increases the generalization capability of a model by generating new data points while training and has shown success in various applications of deep learning [29, 33, 40]; however, it has not been explored in the task of tensor completion. Here, we propose an influence-guided data augmentation technique called DAIN (Data Augmentation with INfluence Functions). First, DAIN trains a neural tensor completion model with the input tensor, and utilizes influence functions to estimate the importance of each training cell (e.g., the importance of a rating in a movie rating tensor) on reducing imputation error. Next, DAIN computes the importance of every entity by aggregating the importance values of all its associated cells. For example, to compute the importance of a user i, the importance of all the ratings given by i are aggregated. The importance of an entity signifies its impact in reducing the prediction error. Finally, DAIN generates new data.
we demonstrate the sensitivity of the performance of DAIN with respect to the predictions. Firstly, Yeh et al. [42] leverage the Representer Theorem for explaining deep neural network predictions. Koh and Liang [17] introduce a model-agnostic approach based on influence functions. Amirata and James [9] propose a data Shapley metric to quantify the value of each training datum in the context of supervised learning. More recently, TRACIN [27] is proposed to compute the influence of a training example on a prediction made by a model with training and test loss gradients. In this paper, we decide to use TRACIN for the tensor data augmentation setting since it is scalable, easy to implement, and does not rely on optimality conditions as Influence Functions [17] or Representer Theorem [42] does.

**Data Augmentation.** Data augmentation is widely used to significantly increase the data available for training models without collecting new data [12]. Basic image data augmentation methods include geometric transformations (such as flipping, cropping, rotation, and translation), mixing images by averaging their pixel values [13], and random erasing which randomly selects an \( n \times m \) patch of an image and masks it with certain values [45]. Augmentation Network model [20] finds image augmentation that maximizes the generalization performance of the classification model via trainable image transformation models and influence functions. Earlier data augmentation methods for text data are based on synonym replacement - replacing a random word in a given sentence with its synonym by using external resources such as WordNet [23, 44] and the pre-trained language model [14]. Yuning et al. [38] leverage machine translation to paraphrase a given text. There is a line of research that perturb text via regex-based transformation [3], noise injection [39], and combining words [10]. However, none of the above methods are applicable in a tensor completion task. To the best of our knowledge, ours is the first work that enhances the performance of neural tensor completion via data augmentation.

Table 1: Comparison of our proposed data augmentation framework DAIN against existing methods. A checkmark indicates that a method satisfies a specific criterion. DAIN is the only data augmentation method satisfying all criteria.

| Architecture Generalizability | Influence Utilization | Data Augmentation Ability |
|------------------------------|-----------------------|--------------------------|
| ✓                            | ✓                     | ✓                        |
| ✓                            | ✓                     | ✓                        |
| ✓                            | ✓                     | ✓                        |
| ✓                            | ✓                     | ✓                        |

In this section, we define the preliminary concepts necessary to understand the proposed method. Table 2 summarizes the symbols frequently used in the paper.

## 3 PRELIMINARIES

### 3.1 Tensor

Tensors are multi-dimensional data and a generalization of vectors (1-order tensors) and matrices (2-order tensors) to the higher order. An \( N \)-way or \( N \)-order tensor has \( N \) dimensions, and the dimension
size (or dimensionality) is denoted by I_1 through I_N, respectively. We denote an N-order tensor by boldface Euler script letters (e.g., X ∈ ℝ^I_1×⋯×I_N). A tensor cell (i_1,...,i_N) contains the value X(i_1,...,i_N), and an entity of a tensor refers to a single index of a dimension. For example, in the movie rating tensor case, an entity refers to a user, movie, or time slice, and a tensor cell contains a rating.

### 3.2 Tensor Completion

Tensor completion is defined as the problem of filling the missing values of a partially observed tensor. Tensor completion methods train their model parameters with observed cells (Ω_train) and predict values of unobserved cells (Ω_test) with the trained parameters. Specifically, given an N-order tensor X ∈ ℝ^I_1×⋯×I_N with training data Ω_train, a tensor completion method aims to find model parameters Θ for the following optimization problem.

\[
\arg\min_\Theta \sum_{(i_1,...,i_N) \in \Omega_{train}} \left( X(i_1,...,i_N) - \hat{X}(i_1,...,i_N) \right)^2
\]

where \(\hat{X}(i_1,...,i_N) = \Theta(i_1,...,i_N)\) is a prediction value for a cell (i_1,...,i_N) generated by the tensor completion method Θ. Neural tensor completion methods utilize different neural network architectures to compute \( \hat{X}(i_1,...,i_N) \) (see Section 4.1 for the multilayer perceptron case). Root-mean-square error (RMSE) is a popular metric to measure the accuracy of a tensor completion method [21, 26, 36]. Specifically, we use test RMSE to check how accurately a tensor completion model predicts values of unobserved tensor cells. The formal definition of test RMSE is given as follows. Notice that a tensor completion model with the lower test RMSE is more accurate.

\[
\text{Test-RMSE} = \sqrt{\frac{1}{|\Omega_{test}|} \sum_{(i_1,...,i_N) \in \Omega_{test}} \left( X(i_1,...,i_N) - \hat{X}(i_1,...,i_N) \right)^2}
\]

### 3.3 Influence Estimation with TRACIN

We introduce the state-of-the-art influence estimator: TRACIN [27]. TRACIN calculates the importance of every training data point in reducing test loss. This is done by tracing training and test loss gradients with respect to model checkpoints, where checkpoints are the model parameters obtained at regular intervals during the training (e.g., at the end of every epoch). The influence of a training data point z on the loss of a test data point z' is given as follows (please refer to Section 3 of TRACIN [27] for the details):

\[
\text{Inf}(z, z') = \sum_{i=1}^{K} \eta_i \nabla \ell(\Theta_{t_i}, z) \cdot \nabla \ell(\Theta_{t_i}, z')
\]

where Θ_{t_i}, 1 ≤ i ≤ K are checkpoints saved at epochs t_1, ..., t_K, η_i is a step size at a checkpoint Θ_{t_i}, and \( \nabla \ell(\Theta_{t_i}, z) \) is the gradient of the loss of z with respect to a checkpoint Θ_{t_i}.

Influence estimation with TRACIN has been shown to have clear advantages in terms of speed and accuracy over existing methods [27], such as Influence Functions [17] or Representer Point method [42]. We utilize TRACIN to create the Cell Importance Tensor in Section 4.2.

### 4 PROPOSED METHOD: DAIN

In this section, we describe DAIN, a data augmentation pipeline using the importance of entities for efficient tensor completion. Figure 1 shows our proposed method, and Algorithm 1 summarizes the overall data augmentation process of DAIN. We explain the steps of our method in the next subsections.

#### 4.1 Training Entity Embeddings

First, we train a neural network model to learn embeddings for every entity in an input tensor (e.g., user, movie, and time embeddings in the movie rating tensor). The traditional technique of learning one embedding per data point is not scalable in tensors, as the number of data points in the tensor can be very large. Instead, we learn one embedding per entity of each dimension (i.e., in the movie rating tensor example, we learn one embedding for every user, movie, and time slice, respectively). Each tensor cell can then be represented as an ordered concatenation of the embeddings of its entities. For example, a tensor cell (i_1,...,i_N) can be represented as \([E_{i_1}^1, E_{i_2}^2, E_{i_3}^3, ..., E_{i_N}^N] \), where \(E_{i_n}^n\) is an embedding of an entity \(i_n\) of the n-th dimension. The entity embeddings are trained to accurately predict the values of the (training) data points in the tensor.

We train an end-to-end trainable neural network to learn the entity embeddings (line 1 in Algorithm 1). We choose a multilayer perceptron (MLP) model with ReLU activation since it shows the best imputation performance with data augmentation (see Figures 3(a) and 3(b)). Note that existing neural tensor completion models, such as NTF [36] and CoSTCo [21], can be also used to generate these embeddings. Our model’s prediction value for a tensor cell (i_1,...,i_N) is defined by the following.

\[
\begin{align*}
Z_1 &= \phi_1(W_1[E_{i_1}^1, ..., E_{i_N}^N] + b_1), \cdots , \\
Z_M &= \phi_M(W_MZ_{M-1} + b_M) \\
\hat{X}(i_1,...,i_N) &= W_{M+1}Z_M + b_{M+1}
\end{align*}
\]
Figure 1: An overview of our proposed data augmentation method DAIN for a 3-dimensional tensor. DAIN first learns entity embeddings with training data and computes training and validation loss gradients (step 1). Next, it obtains cell importance values by the influence estimator TRACIN and creates the Cell Importance Tensor (step 2). DAIN uniformly distributes cell importance values to the corresponding entities, and each entity calculates its importance by aggregating the assigned cell importance values (step 3). Finally, DAIN performs data augmentation to an input tensor by weighted sampling on those entity importance arrays and value predictions (step 4). Note that DAIN can use any neural tensor completion methods to learn entity embeddings (step 1) or predict values of sampled tensor cells (step 4). Please refer to each subsection in Section 4 for detailed information on each step.

Note that $[E^1_i, ..., E^N_i]$ represents the embedding of a cell $(i_1, ..., i_N)$ obtained by concatenating embeddings of entities $i_1, ..., i_N$. $\hat{X}(i_1, ..., i_N)$ is the imputed output value by the neural network for a cell $(i_1, ..., i_N)$, and $M$ indicates the number of hidden layers. $W_1, ..., W_{M+1}$ and $b_1, ..., b_{M+1}$ are weight matrices and bias vectors, respectively. $\phi_1, ..., \phi_M$ are activation functions (DAIN uses ReLU). The model parameters $W_1, ..., W_{M+1}$ and $b_1, ..., b_{M+1}$ are referred to as $\Theta$. By minimizing the loss function (1) combined with Equation (4), we obtain trained entity embeddings as well as loss gradients for all training and validation cells (lines 2-3 in Algorithm 1). We utilize those gradient vectors to compute cell-level importance values in the next part.

### 4.2 Creating Cell Importance Tensor

In Step 2, we create Cell Importance Tensor (CIT) which represents the importance of every tensor cell in reducing the prediction loss. In the movie rating tensor example, CIT stores the importance of ratings. Any influence estimator can be used to compute the CIT, but we choose TRACIN [27] introduced in Section 3.3 since it is scalable, easy to implement, and does not rely on optimality conditions. Moreover, we later show DAIN with TRACIN shows the highest prediction performance empirically compared to other influence estimation methods (see Figures 6(a) and 6(b)). We derive a value of a CIT cell $z$ by the following steps. Computing the CIT is shown in step 2 in Figure 1.

Equation (3) in Section 3.3 computes the influence of a training cell $z$ on the loss of a test cell $z'$. Since we cannot access the test data, we compute the influence $\alpha_z$ of a training cell $z$ on reducing overall validation loss by the following (line 4 in Algorithm 1).

$$\alpha_z = | \sum_{z' \in \Omega_{val}} \ln f(z, z') | = | \sum_{z' \in \Omega_{val}} \eta \nabla \ell(\Theta_{t_i}, z) \cdot \nabla \ell(\Theta_{t_i}, z') |$$

$$\Rightarrow \alpha_z = | \sum_{i=1}^{K} \eta_i \nabla \ell(\Theta_{t_i}, z) \cdot \left( \sum_{z' \in \Omega_{val}} \nabla \ell(\Theta_{t_i}, z') \right) |$$

(5)

where $K$ is the number of checkpoints, $\eta_i$ is a step size at a check-point $\Theta_{t_i}$, and $\nabla \ell(\Theta_{t_i}, z)$ is the gradient of the loss of $z$ with respect to a check-point $\Theta_{t_i}$. We use an absolute value function to the $\alpha_z$ calculation since cells with negative influence can also be important. For example, cells with large negative influence contribute to increasing the validation loss significantly. We can mitigate the loss increase by our data augmentation method, where the absolute function leads DAIN to create more augmentation for these cells.

### 4.3 Calculating Entity Importance

Identifying important entities is crucial. The output of the previous step identifies the cell importance, but the importance of each entity remains unknown. For instance, in the movie rating tensor example,
Algorithm 1: Data Augmentation with DAIN

Input: Tensor \( \mathbf{X} \in \mathbb{R}^{n_1 \times \ldots \times n_N} \), training tensor cells \( \Omega_{\text{train}} \), validation tensor cells \( \Omega_{\text{val}} \), entity embedding training model \( \Theta \), tensor value prediction model \( \Theta_{\text{p}} \), training checkpoints \( \Theta_1, ..., \Theta_K \), step size \( \eta_t \), and the maximum number of augmentation \( N_{\text{aug}} \).

Output: Augmented tensor \( \mathbf{X}_{\text{new}} = \mathbf{X} \cup \mathbf{X}_{\text{aug}} \).

1. Section 4.1
2. Train \( \Theta \) by Equation (1)
3. Obtain entity embeddings for all entities
4. Obtain training and validation loss gradients for all training checkpoints
5. Section 4.2
6. Create Cell Importance Tensor \( \mathbf{a} \) by Equation (5) with \( \Omega_{\text{train}} \) and \( \Omega_{\text{val}} \)
7. Section 4.3
8. Compute the entity importance \( a_i^{(n)} \) of all entities by Equation (6)
9. Section 4.4
10. Train \( \Theta_{\text{p}} \) by Equation (1)
11. for \( i = 1, \ldots, N_{\text{aug}} \) do
12. for \( n = 1, \ldots, N \) do
13. \( z_{\text{new}} \leftarrow \text{weighted sampling on } a_i^{(n)} \) (probability: \( \sum_{i=1}^{n} a_i^{(n)} \))
14. \( \Omega_{\text{aug}} = \Omega_{\text{aug}} \cup \{z_{\text{new}}\} \)
15. \( \mathbf{X}_{\text{aug}} = \mathbf{X} \cup \mathbf{X}_{\text{aug}} \)
16. \( \mathbf{X}_{\text{new}} = \mathbf{X} \cup \mathbf{X}_{\text{aug}} \)

Note that \( a_1^{(1)}, ..., a_N^{(N)} \) are entity importances, \( \lambda \) is a regularization factor, and \( \|\mathbf{X}\| \) is Frobenius norm of a tensor \( \mathbf{X} \). We choose the aggregation scheme to compute the entity importance as the rank-1 CP model can produce inaccurate decomposition results when a given tensor is highly sparse. By conducting extensive experiments, we later show that the aggregation method over the rank-1 CP model with respect to the prediction accuracy (see Figures 7(a) and 7(b)).

The cell importance score captures the importance of the rating on the prediction loss, while it does not reflect the importance of a user, a movie, or a time slice. If we can find the importance of every user, movie, and time slice, we can generate new influential data points to minimize prediction error by combining users, movies, and time slices that have high importance values.

Here, we describe an aggregation technique to calculate the entity importance values from cell importance values (shown in step 3 of Figure 1 and line 5 in Algorithm 1). First, we uniformly distribute a cell’s importance value to its associated entities. For instance, given a training cell \( z = (i_1, ..., i_N) \), we uniformly distribute \( \mathbf{a}_z \) to its \( N \) entities \( \{i_1, ..., i_N\} \). After performing the allocation for all training cells, we compute the entity importance score \( \alpha_i^{(n)} \) for an entity \( i \) of the \( n^{\text{th}} \) dimension by aggregating cell importance scores as follows:

\[
\alpha_i^{(n)} = \sum_{(i_1, ..., i_N) \in \Omega_{\text{train}}, i_n = i} \alpha_{(i_1, ..., i_N)}
\]

Recall that \( \mathbf{a} \) indicates the Cell Importance Tensor, and \( \Omega_{\text{train}} \) represents a set of training cells from an original tensor. In the movie rating tensor example, the above equation signifies that a user’s entity importance is the aggregation of the importance scores of all the ratings the user gives. Similarly, a movie’s importance is the sum of the importance of the ratings it receives, and the importance of a time slice is the sum of the importance of all the ratings given during the time slice.

Another way of computing the entity importance is applying rank-1 CP (CANDDECOMP/PARAFAC) factorization [18] on the Cell Importance Tensor. The output factor matrices from the CP model are entity importances. Specifically, a value of each output array indicates the importance of the corresponding entity on predicting values in a training tensor. The loss function of the rank-1 CP model is given as follows:

\[
L(a_1^{(1)}, ..., a_N^{(N)}) = \sum_{(i_1, ..., i_N) \in \Omega_{\text{train}}} \left( \sum_{n=1}^{N} \alpha_{(i_1, ..., i_N)} - \prod_{n=1}^{N} \alpha_{(i_1, ..., i_N)} \right)^2 + \lambda \sum_{n=1}^{N} \|a_n^{(n)}\|^2
\]
4.5 Complexity Analysis

In this subsection, we analyze the time and space complexity of DAIN.

- **Time complexity.** The first step of DAIN (Section 4.1) is training a neural tensor completion model \( \Theta \) to generate entity embeddings and gradients, and it takes \( O(T_\Theta) \), assuming \( O(T_\Theta) \) is the time complexity of training \( \Theta \) as well as gradient calculations. The second step of DAIN (Section 4.2) is computing the cell importance \( \alpha_z \) for each training cell \( z \in \Omega_{\text{train}} \). A naive computation of \( \alpha_z \) in Equation (5) for all training cells takes \( O(KD|\Omega_{\text{train}}|\Omega_{\text{valid}}|) \), where \( K \) and \( D \) are the number of check points and the dimension of the gradient vector, respectively. We accelerate the computation to \( O(KD(|\Omega_{\text{train}}|+|\Omega_{\text{valid}}|)) \) by precomputing \( \sum_{z \in \Omega_{\text{valid}}} \nabla\ell(\Theta_I, z') \) for all check points \( \Theta_1, \ldots, \Theta_K \) in Equation (5). The third step of DAIN (Section 4.3) is calculating the entity importance with the aggregation technique by Equation (6). It takes \( O(N|\Omega_{\text{train}}|) \) since we distribute \( \alpha_z, \forall z \in \Omega_{\text{train}} \) to its entities and aggregate the assigned values. Finally, the data augmentation step (Section 4.4) takes \( O(T_\Theta + N_{\text{aug}}(N\log I + T_{\text{infer}})) \), where \( O(T_\Theta) \) and \( O(T_{\text{infer}}) \) are the training and single inference time complexities of the value prediction model \( \Theta_p \), respectively. \( O(N_{\text{aug}}N\log I) \) term indicates the time complexity of weighted sampling \( N_{\text{aug}} \) cells without replacement \([34]\) from \( N \) dimensions (assuming \( I_1 = \cdots = I_N = I \)). The final time complexity of DAIN is \( O(T_\Theta + T_\Theta + (KD + N)|\Omega_{\text{train}}| + KD|\Omega_{\text{valid}}| + N_{\text{aug}}(N\log I + T_{\text{infer}})) \).

- **Space complexity.** The first step of DAIN (Section 4.1), obtaining entity embeddings and gradients, takes \( O(M_\Theta + KD(|\Omega_{\text{train}}| + |\Omega_{\text{valid}}|)) \) space, assuming \( O(M_\Theta) \) is the space complexity of training a neural tensor completion model \( \Theta \) (including entity embeddings). \( O(KD(|\Omega_{\text{train}}| + |\Omega_{\text{valid}}|)) \) space is required to store \( D \)-dimension gradient vectors and validation cells for all \( K \) check points. The second step of DAIN (Section 4.2), computing the cell importance \( \alpha_z \), takes \( O(|\Omega_{\text{train}}|) \) space since we need to store all cell importance values. The third step of DAIN (Section 4.3), calculating the entity importance with the aggregation method, takes \( O(NI) \) space since we need to store importance scores of all entities from \( N \) dimensions (assuming \( I_1 = \cdots = I_N = I \)). Finally, the data augmentation step (Section 4.4) takes \( O(M_\Theta + N_{\text{aug}}N) \) space since we need \( O(M_\Theta) \) space for training the value predictor \( \Theta_p \) and \( O(N_{\text{aug}}N) \) space for storing the data augmentation with \( N_{\text{aug}} \) cells. The final space complexity of DAIN is \( O(M_\Theta + M_\Theta + KD(|\Omega_{\text{train}}| + |\Omega_{\text{valid}}|) + N(I + N_{\text{aug}})) \).

5 EXPERIMENTS

In this section, we evaluate DAIN to answer the following questions.

1. **Effectiveness of DAIN (Section 5.2).** How much does our proposed data augmentation technique enhance the accuracy of neural tensor completion compared to the baseline augmentation methods?
2. **Ablation studies of DAIN (Section 5.3).** DAIN consists of three major components: training entity embeddings, generating new tensor cells, and predicting values of the new cells. How much does each component of DAIN contribute to boosting the neural tensor completion accuracy?
3. **Comparisons of influence estimators (Section 5.4).** How much do different influence estimators impact the prediction accuracy improvements of DAIN?
4. **Comparisons of entity importance algorithms (Section 5.5).** How much do different entity importance methods affect the prediction accuracy improvements of DAIN?
5. **Scalability of DAIN (Section 5.6).** Does the running time of DAIN linearly scale with the number of data augmentation?
6. **Hyperparameter sensitivity (Section 5.7).** How much do model hyperparameters of DAIN, such as embedding dimension and layer structures, affect the prediction accuracy?

We first describe the datasets and experimental settings in Section 5.1, and then answer the above questions.

Table 3: Summary of real-world tensor datasets used in our experiments.

| Name                      | Order | Dimensionality | Nonzeros |
|---------------------------|-------|----------------|----------|
| MovingMNIST               | 4     | (100, 20, 64, 64) | 819,200  |
| Foursquare                | 3     | (2321, 5596, 378) | 388,210  |
| Reddit                    | 3     | (10000, 984, 744) | 336,222  |
| LastFM                    | 3     | (980, 1000, 1587) | 258,620  |

5.1 Experimental Settings

5.1.1 Datasets. We use four real-world tensor datasets to evaluate the performance of our proposed data augmentation method and baselines. As summarized in Table 3, we use MovingMNIST [31], Foursquare [43], Reddit [1], and LastFM [11]. MovingMNIST is a video tensor represented by (sequence, length, width, height; intensity), and we use 10% of the total data. Foursquare is a point-of-interest tensor represented by (user, location, timestamp; visit). Reddit includes the posting history of users on subreddits represented by (user, subreddit, timestamp; posted). LastFM contains the music playing history of users represented by (user, music, timestamp; played). For all tensors having timestamps, we converted their timestamps to unique days, so that all timestamps in the same day would have the same converted number. We randomly added negative samples with random indices and zero values to original data for Foursquare, Reddit, and LastFM tensors.

5.1.2 Baselines. To the best of our knowledge, there are no existing methods for data augmentation designed specifically for tensors. Therefore, we create a few baselines based on the broader literature in data augmentation by the following:

- **Duplication or Oversampling:** This simple data augmentation method randomly copies tensor cells and values from the existing training data, and adds them to the original tensor; there can be multiple tensor cells with the same values.
- **Entity Replacement:** This data augmentation method randomly selects existing tensor cells and replaces their indices...
one by one with one of the top-10 closest entities in the embedding space, while keeping their tensor values.

- (Random, MLP) and (Random, CoSTCo): These baselines generate data augmentation by randomly sampling missing tensor cells and predicting their values via the MLP and CoSTCo [21] models trained with the original tensor, respectively. Recall that neural tensor completion methods such as CoSTCo [21] and NTF [36] cannot generate new data points by themselves. Thus, these methods are not included in our baselines. The influence estimator TrACIN [27] also only computes the cell importance but cannot create new data augmentation. We exclude non-neural tensor completion algorithms [22, 26, 28, 41] to test the data augmentation since DAIN is optimized for neural tensor completion methods.

### 5.1.3 Experiment Setup
We randomly split our datasets to use 90% data for training and 10% for test. We randomly sample 20% of training data as the validation set. We stop the training of a neural network when the validation accuracy is not improving anymore, with a patience value of 10. We repeat all experiments 10 times and report the average test RMSE. To measure statistical significance, we use a two-sided test of the null hypothesis that two independent samples have identical average (expected) values. If we observe a small p-value (e.g., < 0.01), we can reject the null hypothesis.

We fine-tune neural network hyperparameters of DAIN with validation data. Specifically, embedding dimension, batch size, layer structure, and learning rate are set to 50, 1024, [1024,1024,128], 0.001, respectively. We use Adaptive Moment Estimation (Adam) [16] optimizer for the neural network training, and the maximum number of epochs for training a neural network is set to 50. To compute the Cell Importance Tensor, we set checkpoints and step size to every epoch and 0.001, respectively. We take gradients with respect to the last fully connected layer. We execute all our experiments on Azure Standard-NC24 machines equipped with 4 NVIDIA Tesla K80 GPUs with Intel Xeon E5-2690 v3 processor. The data augmentation module is implemented in Python with the PyTorch library.

### 5.2 Effectiveness of DAIN
To test the effectiveness of our proposed data augmentation method, we measure and compare test RMSE values of the neural
tensor completion model (MLP) after applying DAIN and baselines on real-world tensors. Figure 2 shows their performance. We omit error bars since they have small standard deviation values.

Both the baseline methods, Duplication and Entity Replacement, lead to a reduction in performance, i.e., an increase in test error, with any amount of augmentation. More augmentation leads to a greater reduction in performance. A potential reason is that the feature spaces of original and augmentation are very similar, so the augmentation cannot increase the generalization capability of a model. Although the other baselines (Random, MLP) and (Random, CoSTCo) outperform Duplication or Entity Replacement, they have limitations in reducing test RMSE compared to DAIN.

Our proposed method DAIN leads to improvements in performance across all datasets with augmentation. DAIN performs the best across all five methods and presents the lowest test RMSE at the maximum augmentation with statistical significance, which clearly demonstrates the effectiveness of DAIN. The performance improves as more data is added to the original one. A key reason for the high performance of DAIN is that it combines important entities from each dimension, which is leading to the augmentation of more influential and heterogeneous data points to the original tensor.

5.3 Ablation Studies of DAIN

At a high-level view, DAIN consists of three components—the first is training entity embeddings with a neural tensor completion method \( \Theta \) (step 1 in Figure 1), the second component is selecting new tensor cells for augmentation based on entity importance (steps 2 and 3), and the final is predicting values of the augmented cells using a neural tensor completion method \( \Theta_p \) (step 4). As mentioned previously, in DAIN, we use MLP, entity importance, and CoSTCo-based predictor in the three components, respectively. 

We perform ablation studies to investigate the contribution of each component in DAIN’s performance by replacing the component with the baseline component, while fixing all the others. Figures 3, 4, and 5 show ablation study results of the three components of DAIN on the two largest tensor datasets, namely MovingMNIST and Foursquare.

In Figure 3, DAIN with MLP embedding generator presents lower test RMSE than a baseline of DAIN with CoSTCo embedding generator at 50% augmentation ratio. A potential reason is that using CoSTCo in both step 1 and step 4 (for augmented cell’s value prediction) can lead to overfitting. On the other hand, using MLP in step 1 and CoSTCo in step 4 gives generalization capability to the model. The entity embedding module has a huge impact on performance improvements since entity embeddings and gradients generated by the module affect all the subsequent modules.

In Figure 4, DAIN with entity importance-based cell creation beats the model with random cell creation. The result substantiates the usefulness of using entity importance in creating tensor cells for augmentation.

In Figure 5, DAIN with CoSTCo for augmented cell’s value prediction outperforms DAIN with a MLP-based value predictor. This is again because of overfitting where the MLP predictor produces homogeneous data points since the entity embedding model in step 1 is an MLP as well. On the other hand, using a CoSTCo-based value predictor produces heterogeneous data points giving generalizability.

In summary, all of the components of DAIN prove useful by reducing test RMSE values significantly compared to baseline components.

5.4 Comparison of Influence Estimators

In this subsection, we explore how different influence estimators affect the prediction accuracy improvements of DAIN on the two largest real-world tensors, namely MovingMNIST and Foursquare datasets. We compare two influence calculation methods: TRACIN [27] and Representer Theorem [42]. We exclude influence function [17] since it shows worse prediction accuracy than the Representer Theorem method. As shown in Figures 6(a) and 6(b), TRACIN consistently outperforms Representer Theorem with any amount of augmentation (with statistical significance at 50% augmentation). One possible explanation for this gap is that TRACIN directly computes the influence of a training cell in reducing validation loss, while Representer Theorem cannot measure the direct contribution.

5.5 Comparison of Entity Importance Methods

Similar to Section 5.4, we confirm how different entity importance calculators affect the prediction accuracy improvements of DAIN on the two largest real-world tensors, namely MovingMNIST and Foursquare datasets. Between two entity importance calculators introduced in Section 4.3, the aggregation algorithm shows superior
We randomly generated factor matrices and reconstructed a synthetic tensor from them. Note that the total running time for 0% augmentation is equivalent to the neural network training time. On the other hand, our proposed data augmentation method involves four runtime-related components: (1) initial neural network training, (2) cell importance calculation, (3) entity importance calculation, and (4) weighted sampling and value inference. Therefore, there are static costs (i.e., $O(\Theta + \Theta_p + (KD + N)|\Omega_{train}| + KD|\Omega_{aug}|$ in Section 4.5) resulted from steps (1), (2), and (3), and linearly increasing costs (i.e., $O(N_{\text{aug}}(N \log I + T_{\text{infer}}))$ in Section 4.5) proportional to the amount of augmentation for step (4).

Figure 8 exhibits the total running time of DAIN on MovingMNIST and Synthetic-1M tensors. As expected before, we find a sudden increase in total running time from 0% to 10% augmentation on both tensors due to steps (2) and (3) above. After that, the running time increases linearly. This shows that DAIN is highly scalable and can be applied to large datasets.

### 5.6 Scalability of DAIN

In this subsection, we measure the runtime of DAIN on the largest real-world, namely MovingMNIST, and a synthetic tensor, namely Synthetic-1M. We construct the synthetic tensor with dimensionality (1000, 1000, 1000, 1000) and 1,000,000 non-zero cells. We randomly generated factor matrices and reconstructed a synthetic tensor from them. Note that the total running time for 0% augmentation is equivalent to the neural network training time. On the other hand, our proposed data augmentation method involves four runtime-related components: (1) initial neural network training, (2) cell importance calculation, (3) entity importance calculation, and (4) weighted sampling and value inference. Therefore, there are static costs (i.e., $O(\Theta + \Theta_p + (KD + N)|\Omega_{train}| + KD|\Omega_{aug}|$ in Section 4.5) resulted from steps (1), (2), and (3), and linearly increasing costs (i.e., $O(N_{\text{aug}}(N \log I + T_{\text{infer}}))$ in Section 4.5) proportional to the amount of augmentation for step (4).

Figure 8 exhibits the total running time of DAIN on MovingMNIST and Synthetic-1M tensors. As expected before, we find a sudden increase in total running time from 0% to 10% augmentation on both tensors due to steps (2) and (3) above. After that, the running time increases linearly. This shows that DAIN is highly scalable and can be applied to large datasets.

### 5.7 Hyperparameter Sensitivity

In this subsection, we investigate how much model hyperparameters affect the prediction accuracy of our neural network hyperparameters include the length of entity embeddings, the number of hidden layers, the size of a hidden layer, learning rate, and batch size. We vary one hyperparameter while fixing all the others to default values mentioned in Section 5.1.3.

Figure 9 shows the hyperparameter sensitivity of DAIN with respect to the prediction accuracy on the Foursquare dataset with 50% augmentation ratio. The RMSE improvement indicates how much validation RMSE values are enhanced after the augmentation (higher is better). As the embedding size increases, we observe performance improvements since large embeddings contain more useful information about training data. Regarding the number of hidden layers, 3 hidden layers are appropriate since 1 or 2 layers may not fully learn the augmented tensor, and 4 layers may overfit. Too small or too large of learning rates degrade the prediction accuracy since a small one leads to slower convergence, and a large one can find a low-quality local optimum.

### 6 CONCLUSION

In this paper, we proposed a novel data augmentation framework DAIN for enhancing neural tensor completion. The key idea is to augment the original tensor with new data points that are crucial in reducing the validation loss. Experimental results on real-world datasets show that DAIN outperforms baseline methods in various augmentation settings with statistical significance. Ablation studies of DAIN demonstrate the effectiveness of the key components of DAIN. We also verify that DAIN scales near linearly to large datasets. Future directions of this work include exploring the effectiveness of DAIN in downstream tasks such as anomaly detection and dataset cleanup. Deriving theoretical guarantees of the performance boost from DAIN is worth exploring as well.

### ACKNOWLEDGMENT

This research is supported in part by Adobe, Facebook, NSF IIS-2027689, Georgia Institute of Technology, IDEA5, and Microsoft. S.O. was partly supported by ML@GT, Twitch, and Kwanjeong fellowships. We thank the reviewers for their feedback.
