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

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
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

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.

	NTF [36]	CoSTCo [21]	NTM [2]	TracIn [27]	Aug-Net [20]	**DAIN (Proposed)**
Neural Tensor Completion	✓	✓	✓			✓
Architecture Generalizability				✓	✓	✓
Influence Utilization				✓	✓	✓
Data Augmentation Ability					✓	✓

points by sampling entities proportional to their importance scores. Values of the augmented tensor cells are predicted via a trained neural tensor completion method. This influence-based sampling of entities augments data points using important entities, and thus, can lead to higher test prediction accuracy.

We evaluate DAIN by conducting extensive experiments on four diverse real-world datasets and one synthetic dataset. Results show that DAIN outperforms baseline augmentation methods with statistical significance, and that DAIN improves the prediction performance of a neural tensor completion method as the number of augmentation increases. Via thorough ablation studies, we confirm the effectiveness of each component of DAIN. We also show that DAIN is scalable to large datasets as illustrated by a near-linear relationship between runtime and the amount of augmentation. Finally, we demonstrate the sensitivity of the performance of DAIN with respect to hyperparameter values.

Our main contributions are summarized by the following:

- **Novel problem.** We propose a novel data augmentation technique DAIN for enhancing neural tensor completion. To the best of our knowledge, this is the first data augmentation method for tensor completion.
- **Algorithmic framework.** We propose a new framework for deriving the importance of tensor entities on reducing prediction error using influence functions. With the entity importance values, we create new data points via weighted sampling and value predictions. We also provide complexity analyses of DAIN.
- **Performance.** DAIN outperforms baseline data augmentation methods on various real-world tensors in terms of prediction accuracy with statistical significance.

The code and datasets are available on the project website: `https://github.com/srijankr/DAIN`.

2 RELATED WORK

Neural Tensor Completion. After recent advances in neural networks (NN), there have been many research topics applying NN to tensor completion frameworks. Neural tensor factorization (NTF) [36] is the first model that employs MLP and long short-term memory (LSTM) architectures for a tensor completion task. Neural Tensor Machine (NTM) [2] combines the outputs from generalized CANDECOMP/PARAFAC (CP) factorization and a tensorized MLP model to estimate missing values. Moreover, Liu *et al.* [21] propose a convolutional neural network (CNN)-based tensor completion model, named CoSTCo, which learns entity embeddings via several convolutional layers. However, as shown in Table 1, the above methods cannot augment new data points, do not generalize to multiple neural network architectures, and do not utilize influence functions. DAIN can provide high-quality augmentation for these methods while having these desirable properties. Note that we exclude tensor completion methods [22, 26, 28, 41] that do not utilize neural networks since DAIN is designed for the neural methods.

Influence Estimation. Computing the influence (or importance) of data points on model predictions or decisions has been actively investigated [9, 17, 27, 42]. Influence has been mainly used to explain the prediction results by identifying useful or harmful data points 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.

3 PRELIMINARIES

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

3.1 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

Table 2: Table of symbols.

Symbol	Definition
$\mathcal{X}$	input tensor
N	order of $\mathcal{X}$
I_n	dimensionality/size of the n^{th} dimension of $\mathcal{X}$
$(i_1, ..., i_N)$	cell of $\mathcal{X}$
i_n	entity of the n^{th} dimension of $\mathcal{X}$
Ω_{train}	set of train cells of $\mathcal{X}$
Ω_{val}	set of validation cells of $\mathcal{X}$
Ω_{test}	set of test cells of $\mathcal{X}$
$\boldsymbol{\alpha}$	cell importance tensor
$\alpha^{(1)}, ..., \alpha^{(N)}$	entity importance
E_i^n	embedding of an entity i of the n^{th} dimension
Θ	parameters of a tensor completion model
Θ_t	parameters of a tensor completion model at epoch t
η_i	step size at a checkpoint Θ_{t_i}
$\Theta_{t_1}, ..., \Theta_{t_K}$	K checkpoints saved at epochs $t_1, ..., t_K$
N_{aug}	number of data augmentation
T_Θ, M_Θ	time and space complexity of training entity embeddings
$T_{\Theta_p}, M_{\Theta_p}$	time and space complexity of training a value predictor
T_{infer}	time complexity of a single inference of a value predictor
D	dimension of a gradient vector

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.*, $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$). A tensor cell $(i_1, ..., i_N)$ contains the value $\mathcal{X}_{(i_1,...,i_N)}$, 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 $\mathcal{X}$ ($\in \mathbb{R}^{I_1 \times \cdots \times I_N}$) with training data Ω_{train}, a tensor completion method aims to find model parameters Θ for the following optimization problem.

$$\arg\min_{\Theta} \sum_{\forall (i_1,...,i_N) \in \Omega_{train}} \left(\mathcal{X}_{(i_1,...,i_N)} - \hat{\mathcal{X}}_{(i_1,...,i_N)} \right)^2 \quad (1)$$

where $\hat{\mathcal{X}}_{(i_1,...,i_N)} = \Theta(i_1, ..., i_N)$ 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 $\hat{\mathcal{X}}_{(i_1,...,i_N)}$ (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.

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

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):

$$Inf(z, z') \approx \sum_{i=1}^{K} \eta_i \nabla \ell(\Theta_{t_i}, z) \cdot \nabla \ell(\Theta_{t_i}, z') \quad (3)$$

where $\Theta_{t_i}, 1 \leq i \leq K$ 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 $[E^1_{i_1}, E^2_{i_2}, E^3_{i_3}, \ldots]$, where $E^n_{i_n}$ 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.

$$\begin{aligned} Z_1 &= \phi_1(W_1[E^1_{i_1}, \ldots, E^N_{i_N}] + b_1), \cdots, \\ Z_M &= \phi_M(W_M Z_{M-1} + b_M) \\ \hat{\mathcal{X}}_{(i_1,...,i_N)} &= W_{M+1} Z_M + b_{M+1} \end{aligned} \quad (4)$$

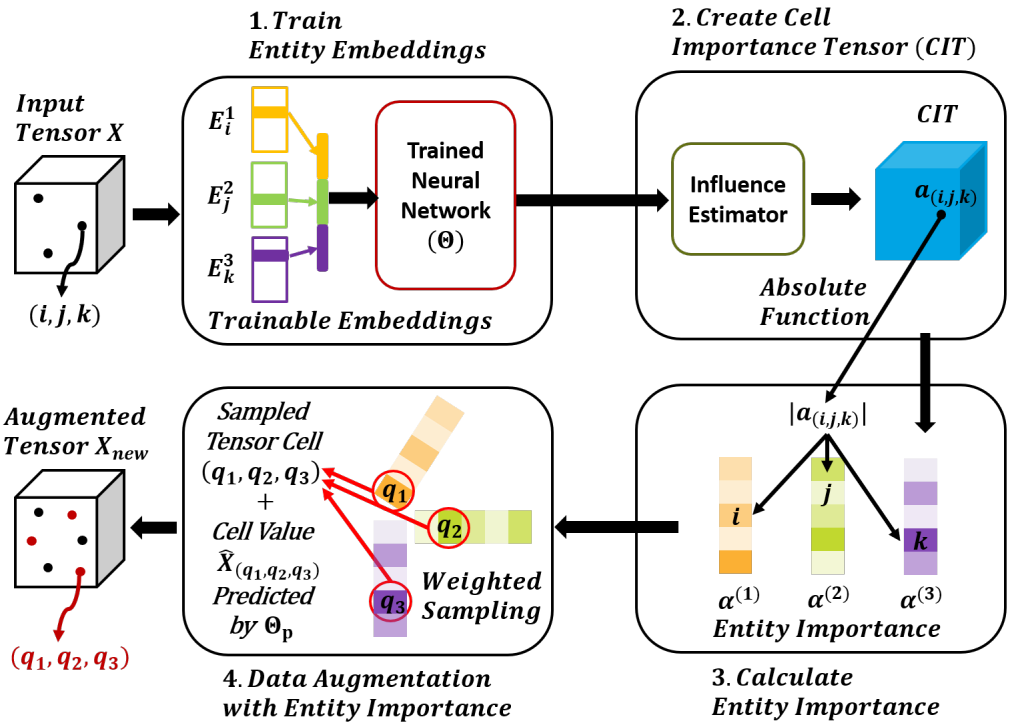


Figure 1: An overview of our proposed data augmentation method DAIN for a 3-dimensional tensor. DAIN first learns entity embeddings with training data and computes training and validation loss gradients (step 1). Next, it obtains cell importance values by the influence estimator TRACIN and creates the 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 $[E^1_{i_1}, \ldots, E^N_{i_N}]$ represents the embedding of a cell $(i_1, ..., i_N)$ obtained by concatenating embeddings of entities $i_1, ..., i_N$, $\hat{\mathcal{X}}_{(i_1,...,i_N)}$ 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 Θ. 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 $\boldsymbol{\alpha}_z$ of a training cell z on reducing overall validation loss by the following (line 4 in Algorithm 1).

$$\boldsymbol{\alpha}_z = |\sum_{z' \in \Omega_{val}} Inf(z, z')| = |\sum_{z' \in \Omega_{val}} \sum_{i=1}^{K} \eta_i \nabla \ell(\Theta_{t_i}, z) \cdot \nabla \ell(\Theta_{t_i}, z')|$$
$$\Longleftrightarrow \boldsymbol{\alpha}_z = |\sum_{i=1}^{K} \eta_i \nabla \ell(\Theta_{t_i}, z) \cdot \left(\sum_{z' \in \Omega_{val}} \nabla \ell(\Theta_{t_i}, z') \right)| \quad (5)$$

where K is the number of checkpoints, 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}. We use an absolute value function to the $\boldsymbol{\alpha}_z$ 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 $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, training tensor cells Ω_{train}, validation tensor cells Ω_{val}, entity embedding training model Θ, tensor value prediction model Θ_p, training checkpoints $\Theta_{t_1}, ..., \Theta_{t_K}$, step size η_i, and the maximum number of augmentation N_{aug}.

Output : Augmented tensor $\mathcal{X}_{new} = \mathcal{X} \cup \mathcal{X}_{aug}$.

▷ **Section 4.1**

1 Train Θ by Equation (1)

2 Obtain entity embeddings for all entities

3 Obtain training and validation loss gradients for all training checkpoints

▷ **Section 4.2**

4 Create Cell Importance Tensor $\boldsymbol{\alpha}$ by Equation (5) with Ω_{train} and Ω_{val}

▷ **Section 4.3**

5 Compute the entity importance $\alpha_i^{(n)}$ of all entities by Equation (6)

▷ **Section 4.4**

6 Train Θ_p by Equation (1)

7 $\Omega_{aug} = \emptyset$

8 **for** $i = 1, \ldots, N_{aug}$ **do**

9 $\quad z_{new} = \emptyset$

10 $\quad$ **for** $n = 1, \ldots, N$ **do**

11 $\quad\quad i_n \longleftarrow$ weighted sampling on $\alpha^{(n)}$ (probability: $\frac{\alpha_{i_n}^{(n)}}{\sum_{i=1}^{I_n} \alpha_i^{(n)}}$)

12 $\quad\quad z_{new} = z_{new} \cup \{i_n\}$

13 $\quad \Omega_{aug} = \Omega_{aug} \cup \{z_{new}\}$

14 let $\mathcal{X}_{aug}$ be a tensor in $\mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with cells Ω_{aug}, where the values of Ω_{aug} are imputed by the trained value predictor Θ_p

15 $\mathcal{X}_{new} = \mathcal{X} \cup \mathcal{X}_{aug}$

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 $\boldsymbol{\alpha}_z$ 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^{th} dimension by aggregating cell importance scores as follows:

$$\alpha_i^{(n)} = \sum_{\forall (i_1,...,i_N) \in \Omega_{train}, i_n = i} \boldsymbol{\alpha}_{(i_1,...,i_N)} \tag{6}$$

Recall that $\boldsymbol{\alpha}$ indicates the Cell Importance Tensor, and Ω_{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 (CANDECOMP/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.

$$L(\alpha^{(1)}, ..., \alpha^{(N)}) = \sum_{\forall (i_1,...,i_N) \in \Omega_{train}} \left(|\boldsymbol{\alpha}_{(i_1,...,i_N)}| - \prod_{n=1}^{N} \alpha_{i_n}^{(n)} \right)^2 + \lambda \sum_{n=1}^{N} \|\alpha^{(n)}\|^2 \tag{7}$$

Note that $\alpha^{(1)}, ..., \alpha^{(N)}$ are entity importances, λ is a regularization factor, and $\|\mathcal{X}\|$ is Frobenius norm of a tensor $\mathcal{X}$. 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)).

4.4 Data Augmentation with Entity Importance

In the final step, we identify the tensor cells and values for data augmentation using the entity importance scores and a value predictor, respectively. This is illustrated in step 4 of Figure 1 and lines 6–15 in Algorithm 1. A high entity importance score signifies that the corresponding entity plays an important role in improving the validation set prediction. Thus, we create new cells using these important entities.

We first train a neural tensor completion model Θ_p with the original training cells (line 6), which will be used later for value predictions. After that, we conduct weighted sampling on every entity importance array (line 11) and select one entity from each dimension (line 12). Mathematically, an entity i of the n^{th} dimension (*e.g.*, a user among all users) has a probability $\frac{\alpha_i^{(n)}}{\sum_{i=1}^{I_n} \alpha_i^{(n)}}$ to be sampled. We combine the sampled entities from all dimensions to form one tensor cell (line 13). We repeat the process several times to create the required number of data points for augmentation (lines 8–13).

Once indices of the cells are sampled, their values need to be determined (line 14). Trivially, one can use the overall average value (*i.e.*, $\frac{1}{|\Omega_{train}|} \sum_{\forall (i_1,...,i_N) \in \Omega_{train}} \mathcal{X}_{(i_1,...,i_N)}$) or find the most similar index in the embedding space and take its value. However, these heuristics can be inaccurate and computationally expensive, respectively. An advanced way of assigning the values of the augmented data points is by predicting the values using a tensor completion model (either the previously trained Θ or training another model Θ_p). Specifically, we can use the trained embeddings from Θ to predict the values, or we can train an off-the-shelf prediction method Θ_p with the training data and predict the values with it. Reusing the trained neural network Θ is computationally cheaper since it only needs to do a forward pass for inference, which is fast. However, this can cause overfitting in the downstream model since the resulting augmentation cell values are likely to be homogeneous to the original tensor. On the other hand, using another off-the-shelf method Θ_p can increase the generalization capability of a downstream model by generating more heterogeneous data compared to Θ. Thus, we choose to train a new off-the-shelf model Θ_p. We use the state-of-the-art algorithm CoSTCo [21] as Θ_p to predict the values of the augmented tensor cells, since CoSTCo value predictor exhibits high prediction accuracy than other methods empirically (see Section 5.3 and Figure 5).

After we obtain all tensor cell indices and values needed for augmentation, we add them to the input tensor to obtain an augmented tensor $\mathcal{X}_{new}$ (line 15). This augmented tensor can be used for downstream tasks.

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 Θ to generate entity embeddings and gradients, and it takes $O(T_\Theta)$ assuming $O(T_\Theta)$ is the time complexity of training Θ as well as gradient calculations. The second step of DAIN (Section 4.2) is computing the cell importance $\boldsymbol{\alpha}_z$ for each training cell $z \in \Omega_{train}$. A naive computation of $\boldsymbol{\alpha}_z$ in Equation (5) for all training cells takes $O(KD|\Omega_{train}||\Omega_{val}|)$, where K and D are the number of checkpoints and the dimension of the gradient vector, respectively. We accelerate the computation to $O(KD(|\Omega_{train}|+|\Omega_{val}|))$ by precomputing $\sum_{z' \in \Omega_{val}} \nabla \ell(\Theta_{t_i}, z')$ for all checkpoints $\Theta_{t_1}, ..., \Theta_{t_K}$ 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 $O(N|\Omega_{train}|)$ since we distribute $\boldsymbol{\alpha}_z, \forall z \in \Omega_{train}$ to its entities and aggregate the assigned values. Finally, the data augmentation step (Section 4.4) takes $O(T_{\Theta_p} + N_{aug}(N \log I + T_{infer}))$, where $O(T_{\Theta_p})$ and $O(T_{infer})$ are the training and single inference time complexities of the value prediction model Θ_p, respectively. $O(N_{aug} N \log I)$ term indicates the time complexity of weighted sampling N_{aug} cells without replacement [34] from N dimensions (assuming $I_1 = \cdots = I_N = I$). The final time complexity of DAIN is $O(T_\Theta + T_{\Theta_p} + (KD + N)|\Omega_{train}| + KD|\Omega_{val}| + N_{aug}(N \log I + T_{infer}))$.
- **Space complexity.** The first step of DAIN (Section 4.1), obtaining entity embeddings and gradients, takes $O(M_\Theta + KD(|\Omega_{train}| + |\Omega_{val}|))$ space, assuming $O(M_\Theta)$ is the space complexity of training a neural tensor completion model Θ (including entity embeddings). $O(KD(|\Omega_{train}| + |\Omega_{val}|))$ space is required to store D-dimension gradients of training and validation cells for all K checkpoints. The second step of DAIN (Section 4.2), computing the cell importance $\boldsymbol{\alpha}_z$, takes $O(|\Omega_{train}|)$ 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 $O(M_{\Theta_p} + N_{aug}N)$ space since we need $O(M_{\Theta_p})$ space for training the value predictor Θ_p and $O(N_{aug}N)$ space for storing the data augmentation with N_{aug} cells. The final space complexity of DAIN is $O(M_\Theta + M_{\Theta_p} + KD(|\Omega_{train}| + |\Omega_{val}|) + N(I + N_{aug}))$.

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, 10000, 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

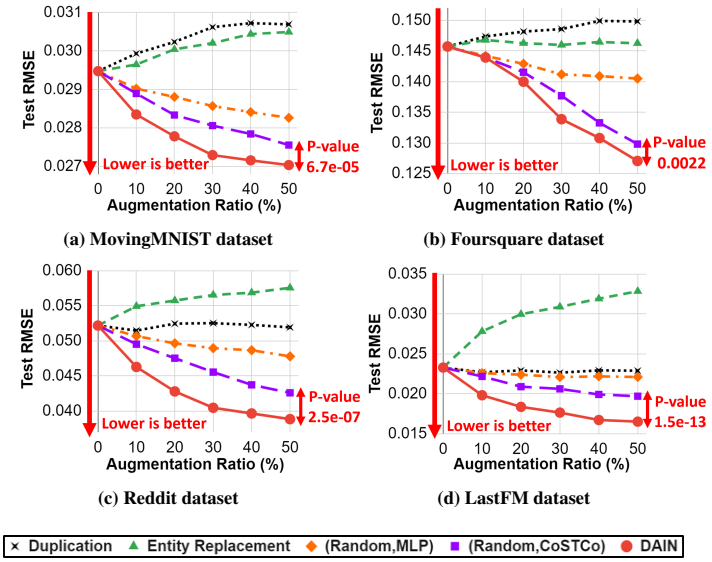


Figure 2: *Effectiveness of* DAIN. Test RMSE of the neural tensor completion method (MLP) after applying various data augmentation methods on real-world datasets. For all tensors, DAIN outperforms all baselines and shows the lowest test RMSE value at the maximum augmentation with statistical significance.

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

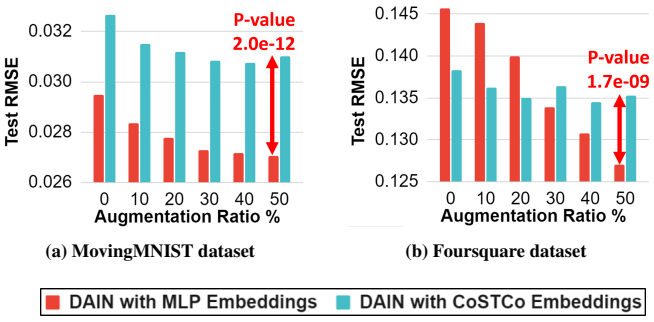


Figure 3: *Ablation study of entity embedding generators of* DAIN. We test the effectiveness of the MLP embedding model of DAIN on MovingMNIST and Foursquare tensors. The MLP model outperforms the CoSTCo embedding model in terms of test RMSE at the maximum augmentation with statistical significance.

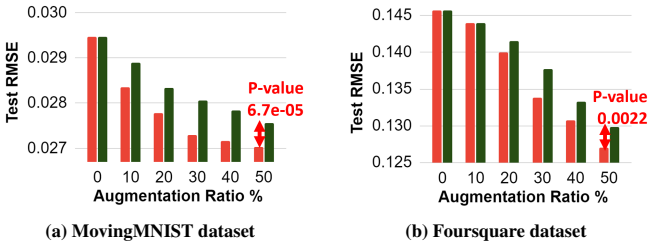


Figure 4: *Ablation study of cell creation methods of* DAIN. We test the effectiveness of the cell creation component of DAIN on MovingMNIST and Foursquare tensors. The entity importance-based cell creation of DAIN outperforms the random cell creation baseline in terms of test RMSE at the maximum augmentation with statistical significance.

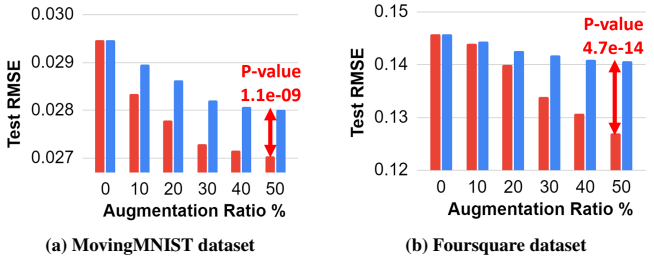


Figure 5: *Ablation study of value prediction methods of* DAIN. We test the effectiveness of the value prediction component of DAIN on MovingMNIST and Foursquare tensors. The value prediction module of DAIN outperforms the MLP baseline in terms of test RMSE at the maximum augmentation with statistical significance. We choose the MLP baseline since it shows higher prediction accuracy than the random one.

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

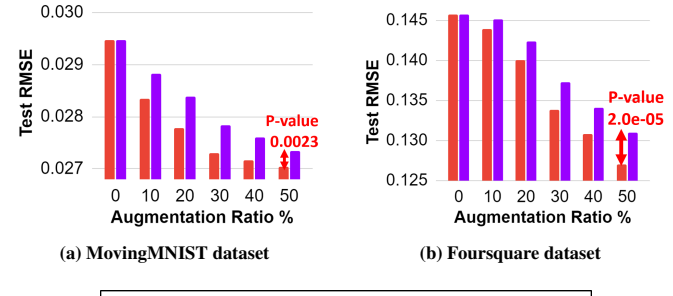
At a high-level view, DAIN consists of three components—the first is training entity embeddings with a neural tensor completion method Θ (step 1 in Figure 1), the second component is selecting new tensor cells for augmentation based on entity importance (steps 2 and 3), and the final is predicting values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

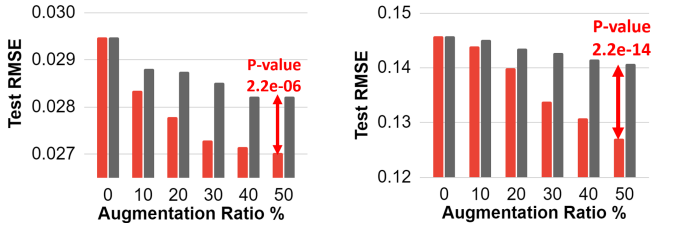
[1] Please note that we also conducted experiments with CoSTCo as the embedding generator and MLP as the augmented cell's value predictor, but its prediction performance was significantly lower (with p-value < 0.05). Thus, in this subsection, we conduct ablation studies on the proposed model where MLP is the embedding generator and CoSTCo is the value predictor.



(a) MovingMNIST dataset (b) Foursquare dataset

DAIN with TracIn | DAIN with Representer Theorem

Figure 6: ***Comparisons of influence estimators.*** **We compare two influence estimators with respect to test RMSE improvements on MovingMNIST and Foursquare tensors. We find the TracIn influence estimator is more suitable for our data augmentation framework.**



(a) MovingMNIST dataset (b) Foursquare dataset

DAIN with Aggregation Method | DAIN with Rank-1 CP

Figure 7: ***Comparisons of entity importance algorithms.*** **We compare two entity importance algorithms with respect to test RMSE improvements on MovingMNIST and Foursquare tensors. We find entity importance calculation by the aggregation scheme is more suitable for our data augmentation framework.**

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.

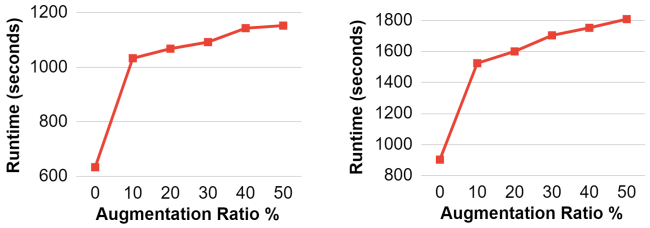
In summary, all of the components of DAIN prove useful by reducing test RMSE values significantly compared to baseline components.

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



(a) MovingMNIST dataset (b) Synthetic-1M dataset

Figure 8: *Scalability of* DAIN. This plot measures the running time of DAIN at different augmentation levels. The sudden jump in total running time from 0% to 10% augmentation occurs due to the cost for the cell and entity importance calculation (see time complexity analysis in Section 4.5); the runtime increases linearly thereafter.

performance than the rank-1 CP factorization (see Figures 7(a) and 7(b)) with extensive test RMSE differences at 50% augmentation. The main reason for this gap is that rank-1 CP factorization cannot decompose the Cell Importance Tensor accurately due to its high sparsity, while the aggregation algorithm does not suffer from the sparsity issue.

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.*, $O(T_\Theta + T_{\Theta_p} + (KD+N)|\Omega_{train}| + KD|\Omega_{val}|$ in Section 4.5) resulted from steps (1), (2), and (3), and linearly increasing costs (*i.e.*, $O(N_{aug}(N \log I + T_{infer}))$ 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 a model. 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

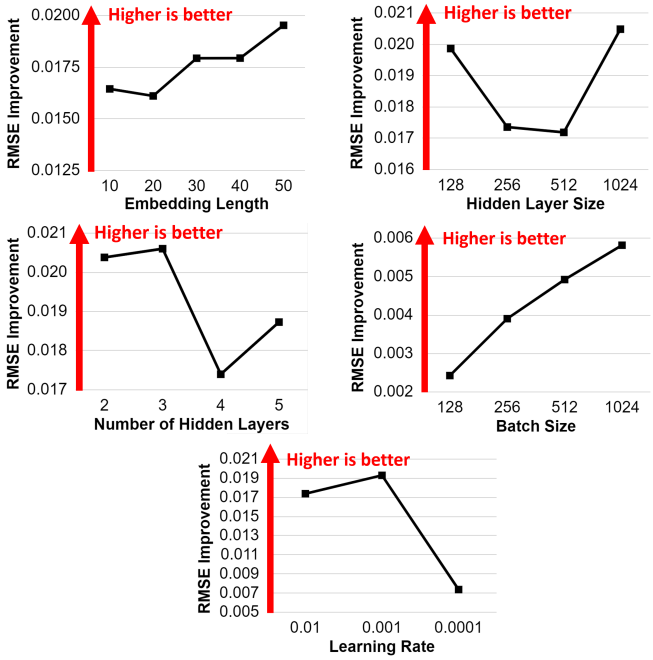


Figure 9: *Hyperparameter sensitivity plots.* We vary one of the hyperparameters of DAIN, namely embedding length, hidden layer size, number of hidden layers, batch size, and learning rate, while fixing all the others to default values. We measure the validation RMSE improvements after 50% augmentation on the Foursquare dataset. We find medium-sized layers with large embedding dimensions, and proper learning rates are key factors for the prediction accuracy.

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, IDEaS, and Microsoft. S.O. was partly supported by ML@GT, Twitch, and Kwanjeong fellowships. We thank the reviewers for their feedback.

REFERENCES

[1] 2020. Reddit data dump. http://files.pushshift.io/reddit/.

[2] Huiyuan Chen and Jing Li. 2020. Neural Tensor Model for Learning Multi-Aspect Factors in Recommender Systems. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. International Joint Conferences on Artificial Intelligence Organization, 2449–2455.

[3] Claude Coulombe. 2018. Text data augmentation made simple by leveraging nlp cloud apis. *arXiv preprint arXiv:1812.04718* (2018).

[4] Justin Dauwels, Lalit Garg, Arul Earnest, and Leong Khai Pang. 2012. Tensor factorization for missing data imputation in medical questionnaires. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2109–2112.

[5] Renwei Dian, Leyuan Fang, and Shutao Li. 2017. Hyperspectral image super-resolution via non-local sparse tensor factorization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5344–5353.

[6] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. 2011. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 5, 2 (2011), 1–27.

[7] Marko Filipović and Ante Jukić. 2015. Tucker factorization with missing data with application to low-n-rank tensor completion. *Multidimensional systems and signal processing* 26, 3 (2015), 677–692.

[8] Evgeny Frolov and Ivan Oseledets. 2017. Tensor methods and recommender systems. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 3 (2017), e1201.

[9] Amirata Ghorbani and James Zou. 2019. Data Shapley: Equitable Valuation of Data for Machine Learning. In *International Conference on Machine Learning*. 2242–2251.

[10] Hongyu Guo, Yongyi Mao, and Richong Zhang. 2019. Augmenting data with mixup for sentence classification: An empirical study. *arXiv preprint arXiv:1905.08941* (2019).

[11] Balázs Hidasi and Domonkos Tikk. 2012. Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 67–82.

[12] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. 2019. Population based augmentation: Efficient learning of augmentation policy schedules. In *International Conference on Machine Learning*. 2731–2741.

[13] Hiroshi Inoue. 2018. Data augmentation by pairing samples for images classification. *arXiv preprint arXiv:1801.02929* (2018).

[14] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for Natural Language Understanding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*. 4163–4174.

[15] Oguz Kaya and Bora Uçar. 2015. Scalable sparse tensor decompositions in distributed memory systems. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–11.

[16] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[17] Pang Wei Koh and Percy Liang. 2017. Understanding Black-box Predictions via Influence Functions.. In *ICML (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1885–1894.

[18] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* 51, 3 (2009), 455–500.

[19] Tamara G Kolda and Jimeng Sun. 2008. Scalable tensor decompositions for multi-aspect data mining. In *2008 Eighth IEEE international conference on data mining*. IEEE, 363–372.

[20] Donghoon Lee, Hyunsin Park, Trung Pham, and Chang D. Yoo. 2020. Learning Augmentation Network via Influence Functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 10958–10967.

[21] Hanpeng Liu, Yaguang Li, Michael Tsang, and Yan Liu. 2019. CoSTCo: A Neural Tensor Completion Model for Sparse Tensors. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 324–334.

[22] Yuanyuan Liu, Fanhua Shang, Wei Fan, James Cheng, and Hong Cheng. 2014. Generalized higher-order orthogonal iteration for tensor decomposition and completion. *Advances in Neural Information Processing Systems* 27 (2014), 1763–1771.

[23] Jonas Mueller and Aditya Thyagarajan. 2016. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.

[24] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. 2017. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*. 5947–5956.

[25] Jinoh Oh, Kijung Shin, Evangelos E Papalexakis, Christos Faloutsos, and Hwanjo Yu. 2017. S-hot: Scalable high-order tucker decomposition. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. 761–770.

[26] Sejoon Oh, Namyong Park, Sael Lee, and U Kang. 2018. Scalable tucker factorization for sparse tensors-algorithms and discoveries. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1120–1131.

[27] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. 2020. Estimating Training Data Influence by Tracing Gradient Descent. *Advances in Neural Information Processing Systems* 33 (2020).

[28] Kijung Shin, Lee Sael, and U Kang. 2017. Fully Scalable Methods for Distributed Tensor Factorization. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2017), 100–113.

[29] Connor Shorten and Taghi M. Khoshgoftaar. 2019. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data* 6, 1 (06 Jul 2019), 60. https://doi.org/10.1186/s40537-019-0197-0

[30] Shaden Smith and George Karypis. 2017. Accelerating the tucker decomposition with compressed sparse tensors. In *European Conference on Parallel Processing*. Springer, 653–668.

[31] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. 2015. Unsupervised learning of video representations using lstms. In *International conference on machine learning*. 843–852.

[32] Huachun Tan, Guangdong Feng, Jianshuai Feng, Wuhong Wang, Yu-Jin Zhang, and Feng Li. 2013. A tensor-based method for missing traffic data completion. *Transportation Research Part C: Emerging Technologies* 28 (2013), 15–27.

[33] Jason Wang and Luis Perez. 2017. The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit* 11 (2017), 1–8.

[34] Chak-Kuen Wong and Malcolm C. Easton. 1980. An efficient method for weighted sampling without replacement. *SIAM journal on computing* 9, 1 (1980), 111–113.

[35] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. 2016. Understanding data augmentation for classification: when to warp?. In *DICTA*. IEEE, 1–6.

[36] Xian Wu, Baoxu Shi, Yuxiao Dong, Chao Huang, and Nitesh V. Chawla. 2019. Neural Tensor Factorization for Temporal Interaction Learning. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 537–545.

[37] Kun Xie, Huali Lu, Xin Wang, Gaogang Xie, Yong Ding, Dongliang Xie, Jigang Wen, and Dafang Zhang. 2020. Neural Tensor Completion for Accurate Network Monitoring. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1688–1697.

[38] Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. 2020. Unsupervised Data Augmentation for Consistency Training. *Advances in Neural Information Processing Systems* 33 (2020).

[39] Ziang Xie, Sida I. Wang, Jiwei Li, Daniel Lévy, Aiming Nie, Dan Jurafsky, and Andrew Y. Ng. 2017. Data Noising as Smoothing in Neural Network Language Models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=H1VyHY9gg

[40] Yan Xu, Ran Jia, Lili Mou, Ge Li, Yunchuan Chen, Yangyang Lu, and Zhi Jin. 2016. Improved relation classification by deep recurrent neural networks with data augmentation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. 1461–1470.

[41] Chaoqi Yang, Navjot Singh, Cao Xiao, Cheng Qian, Edgar Solomonik, and Jimeng Sun. 2021. MTC: Multiresolution Tensor Completion from Partial and Coarse Observations. *arXiv preprint arXiv:2106.07135* (2021).

[42] Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. 2018. Representer point selection for explaining deep neural networks. In *Advances in neural information processing systems*. 9291–9301.

[43] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat Thalmann. 2013. Time-aware point-of-interest recommendation. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. 363–372.

[44] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*. 649–657.

[45] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2020. Random erasing data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 13001–13008.