# A Recommender System for Complex Real-World Applications with Nonlinear Dependencies and Knowledge Graph Context

Marcel Hildebrandt[1,2(✉)], Swathi Shyam Sunder[1], Serghei Mogoreanu[1], Mitchell Joblin[1], Akhil Mehta[1,3], Ingo Thon[1], and Volker Tresp[1,2]

[1] Siemens AG, Corporate Technology, Munich, Germany
{marcel.hildebrandt,swathi.sunder,serghei.mogoreanu,mitchell.joblin,
akhil.mehta,ingo.thon,volker.tresp}@siemens.com
[2] Ludwig Maximilian University, Munich, Germany
[3] Technical University of Munich, Munich, Germany

**Abstract.** Most latent feature methods for recommender systems learn to encode user preferences and item characteristics based on past user-item interactions. While such approaches work well for standalone items (e.g., books, movies), they are not as well suited for dealing with composite systems. For example, in the context of industrial purchasing systems for engineering solutions, items can no longer be considered standalone. Thus, latent representation needs to encode the functionality and technical features of the engineering solutions that result from combining the individual components. To capture these dependencies, expressive and context-aware recommender systems are required. In this paper, we propose *NECTR*, a novel recommender system based on two components: a tensor factorization model and an autoencoder-like neural network. In the tensor factorization component, context information of the items is structured in a multi-relational knowledge base encoded as a tensor and latent representations of items are extracted via tensor factorization. Simultaneously, an autoencoder-like component captures the non-linear interactions among configured items. We couple both components such that our model can be trained end-to-end. To demonstrate the real-world applicability of *NECTR*, we conduct extensive experiments on an industrial dataset concerned with automation solutions. Based on the results, we find that *NECTR* outperforms state-of-the-art methods by approximately 50% with respect to a set of standard performance metrics.

## 1 Introduction

In the context of recommender systems, collaborative filtering methods predict the interest of a user for items by considering past user-item interactions of similar users. Many popular collaborative filtering methods are based on the idea of learning latent representations for items and users. In that setting, the


P. Hitzler et al. (Eds.): ESWC 2019, LNCS 11503, pp. 179–193, 2019.
https://doi.org/10.1007/978-3-030-21348-0_12

recommendation task is solved by scoring user-item pairs via some function of the latent features. Popular approaches are based on matrix factorization (MF) and have been applied with great success in the past (e.g., in the Netflix challenge [1] and in Amazon's item-to-item collaborative filtering method [10]). Most factorization methods model the relation between users and items via dot products of the corresponding latent features. This implies, that – when either of the two latent factors is fixed – the recommendation engine becomes essentially linear. While this limited expressiveness suffices to capture the relations between the users and standalone items of similar type, MF-based methods are not well suited for dealing with heterogeneous (i.e., diverse in nature) items that interact with each other to form a complex system.

Nevertheless, such scenarios are commonplace when users configure modular products consisting of many different components (e.g., computers or IT infrastructure solutions). This setting imposes additional challenges on collaborative filtering systems. It is often the case that the functionality of the whole system depends on the technical properties of the individual components. Therefore, the need arises to model this context information explicitly. Moreover, instead of encoding preferences that the user has revealed in the past, latent features need to encode the functionality of the system that arises by combining the individual components. Depending on the complexity of the configured system, this may result in higher-order (i.e., non-linear) dependencies among the items.

Deep neural networks have proven to be highly effective in extracting hidden, non-linear representations from raw input data for a variety of tasks (e.g., natural language processing [16], computer vision [9], and speech recognition [8]). In particular, autoencoders are invaluable for learning efficient data representations in an unsupervised manner, and are often used for denoising, as generative models, as well as for producing latent representations that can be employed for further downstream tasks. Related to our use case, [15] and [6] describe architectures that integrate autoencoders into recommender engines. While these methods are capable of taking some context information into account, they only consider a setting with rather homogeneous items (books and movies, respectively) that all share the same (fairly small) set of features. The main challenge when dealing with highly modular, technical systems is, however, to find an efficient way to encode the numerous technical properties of items that belong to different device categories and thus have different feature sets. Moreover, this information must be channeled and merged with historical data efficiently to meet the requirements of real-world, industrial applications.

**Present Work.** In this work, we present *NECTR* (**N**eural **E**ncoders **C**ombined with **T**ensor Decompositions for **R**ecommendations) – a novel hybrid recommender system based on the combination of autoencoders and tensor factorizations. The basic idea is to form a graphical, multi-relational knowledge base, which contains technical information about items as well as historical user-item interactions. By factorizing the resulting adjacency tensor, one obtains semantically meaningful embeddings that preserve local proximity in the graph structure.

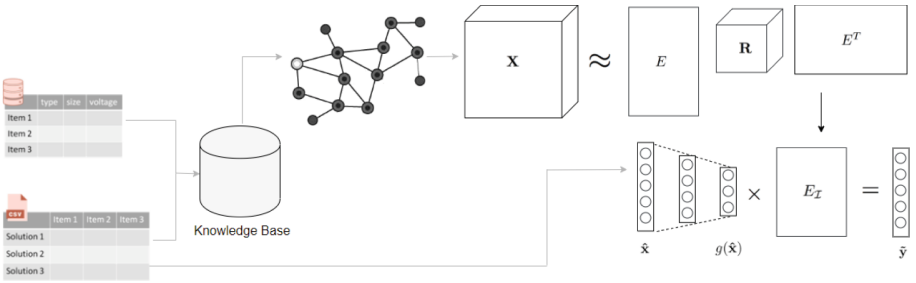


**Fig. 1.** The architecture of NECTR: We merge both historical data and technical information from industrial databases to form a joint multi-relational knowledge graph. Then we extract context-aware embeddings by factorizing the corresponding adjacency tensor. The resulting latent representations of items are employed in the output layer of an autoencoder-like neural network that is trained to score items based on the current configuration.

We leverage this information by coupling the tensor factorization with a deep learning autoencoder via a weight-sharing mechanism. The architecture of NECTR is illustrated in Fig. 1.

In order to demonstrate the practical benefits of *NECTR*, we conduct extensive, large scale experiments in a real-world industrial setting. The underlying data consists of around 50,000 engineering solutions (from the automation domain), 6,000 configurable items, and around half a million technical properties of items.[1] Our main findings are:

- *NECTR* outperforms all state-of-the-art methods that we considered by a large margin. In some of the most important metrics we achieve performance gains of more than 50%.
- *NECTR* can be efficiently trained in an end-to-end fashion. Moreover, recommendations can be executed in real time via a simple forward pass. This is crucial in real-world applications, since we require that the recommender system works in real time while the user is configuring a solution.

## 2 Background and Mathematical Notation

Before proceeding, we define the mathematical notations that we use for this paper and provide the necessary background on automation solutions and knowledge graphs.

### 2.1 Notation

Throughout this work, we use the following notation: Scalars are given by lowercase letters ($x \in \mathbb{R}$), column vectors by bold lowercase letters ($\mathbf{x} \in \mathbb{R}^n$), and

[1] The anonymized data along with implementations of all methods that we consider in this paper can be found under https://github.com/m-hildebrandt/NECTR.

matrices by uppercase letters ($X \in \mathbb{R}^{n_1 \times n_2}$). The $i$-th row and $j$-th column of matrix $X$ are denoted as $X_{i,:} \in \mathbb{R}^{n_2}$ and $X_{:,j} \in \mathbb{R}^{n_1}$, respectively. Further, for $I \subset \{1, 2, \ldots, n_1\}$, $X_I \in \mathbb{R}^{|I| \times n_2}$ indicates the matrix formed by the rows of $X$ indexed by elements in $I$. Third-order tensors are given by bold uppercase letters ($\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$). Slices of a tensor (i.e., two-dimensional sections obtained by fixing one index) are denoted by $\mathbf{X}_{i,:,:} \in \mathbb{R}^{n_2 \times n_3}$, $\mathbf{X}_{:,j,:} \in \mathbb{R}^{n_1 \times n_3}$, and $\mathbf{X}_{:,:,k} \in \mathbb{R}^{n_1 \times n_2}$.

### 2.2 Automation Solutions

Automation systems are closely related to control theory. Usually, the goal is that a set of variables in a system takes a prescribed value or varies in a prescribed way (a classical example is a heating unit with a thermostat and a controller). In this work we are concerned with industrial automation employed in manufacturing processes and machinery. Thereby, the employed solutions consist of a wide variety of components such as sensors, software, controllers, and human machine interfaces. Many of these components are general-purpose and can be applied in various industries, such as car manufacturing, bottling plants, pharmaceutical production, and oil refinery. However, the suitability of some components can also be heavily influenced by non-functional requirements, which arise in certain domains. Examples for such non-functional requirements may include environmental conditions, such as high temperatures and humidity, or legal requirements, as in food, pharmaceutical, or energy production. As an example, a cooling system in the food industry might contain the same functional components as the cooling of a nuclear power plant, however, in the latter case, fail-safe components are required. Selecting a consistent set of components is complicated as there are lots of interdependencies due to protocol standards or technical compatibility constraints. The large amount of such restrictions, the diversity of application domains, and in some cases, also the rather soft character of constraints prohibit the formulation of handcrafted, universally valid rules that are typically used in product configurators for less complex systems. Thus, in the context of automation solutions, recommender systems need to be sufficiently expressive to model the complex interplay of components while also taking technical properties into account.

### 2.3 Knowledge Graphs

In our approach, we make use of a knowledge graph to capture technical information describing configurable items and past solutions. All considered entities correspond to vertices in a directed graph with typed edges. We denote the indexed set of entities by $\mathcal{E}$ and the indexed set of binary relations by $\mathcal{R}$ with $r \subset \mathcal{E} \times \mathcal{E}$ for all $r \in \mathcal{R}$. Further, consider $n_E := |\mathcal{E}|$ and $n_R := |\mathcal{R}|$, i.e., the number of entities and relations, respectively. Figure 2 shows an excerpt of a knowledge graph in an industrial setting.

A triple of the form $(e_i, r_k, e_j)$ with $(e_i, e_j) \in r_k$ is interpreted as a true fact. Given a relation $r_k \in \mathcal{R}$, the characteristic function $\phi_{r_k} : \mathcal{E} \times \mathcal{E} \to \{0, 1\}$

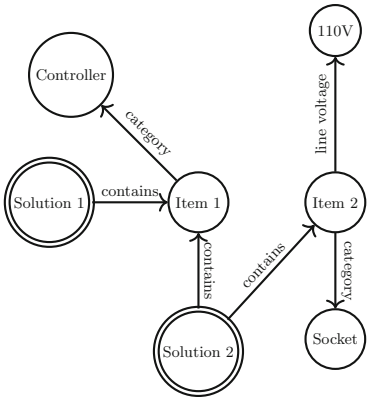


**Fig. 2.** An example of an industrial knowledge graph

indicates the truth value of triples. More concretely, $\phi_{r_k}(e_i, e_j) = 1$ (0) implies, that the corresponding triple $(e_i, r_k, e_j)$ is interpreted as a true (unknown) fact. The entirety of all characteristic functions induces an adjacency tensor $\mathbf{X} \in \mathbb{R}^{n_E \times n_E \times n_R}$ with $\mathbf{X}_{i,j,k} = \phi_{r_k}(e_i, e_j)$.

Most knowledge graphs that are currently in use are far from being complete in the sense that they are missing many true facts about the entities at hand. Therefore, one of the canonical machine learning tasks applied to knowledge graphs consists of predicting new edges (i.e., facts) given the remaining connectivity pattern. This problem of link prediction is sometimes also referred to as knowledge graph completion.

One of the most popular knowledge graph completion methods is given by RESCAL [13]. RESCAL is based on factorizing the frontal slices of the adjacency tensor $\mathbf{X}$ as a product of the factor matrix $E \in \mathbb{R}^{n_E \times d}$ and the frontal slices of a core tensor $\mathbf{R} \in \mathbb{R}^{d \times d \times n_R}$. $d$ is a hyperparameter that indicates the number of latent dimensions. This leads to

$$\mathbf{X}_{i,j,k} \approx E_{i,:} \mathbf{R}_{:,:,k} E_{j,:}^T \,. \tag{1}$$

The parameters of RESCAL are usually obtained by minimizing the sum of the squared residuals between the observed and the predicted entries of $\mathbf{X}$. Hence, the objective function is given by

$$\text{loss}(\mathbf{X}, E, \mathbf{R}) = \sum_{r=1}^{n_R} ||\mathbf{X}_{:,:,r} - E\mathbf{R}_{:,:,r}E^T||_F^2 \,, \tag{2}$$

where $||\cdot||_F$ denotes the Frobenius norm. Usually this optimization problem is solved via alternating least squares. After fitting the model, the rows of $E$ contain latent representations of the entities in $\mathcal{E}$. Similarly, each frontal slice of $\mathbf{R}$ contains the corresponding latent representations of the different relations in $\mathcal{R}$.

## 3 Related Methods

### 3.1 Recommender Systems

Most recommender systems are based on collaborative filtering (e.g., [1,10]). These methods aggregate the preferences of users and generate recommendations based on user similarities in overall preference patterns. However, they sometimes suffer from the sparsity of data, i.e., the lack of adequate preference information. One of the ways to tackle this issue is to consider content information about the users/items as in content-based recommender systems. In their simplest form, content-based recommender systems are classifiers that map user and item information to the selection probability. Refer [11] for a survey of the related literature and state-of-the-art content-based recommender systems. Applying a purely content-based approach to our use case of industrial purchasing systems would ignore previous configuration patterns, which encode implicit requirements for automation solutions.

Hybrid recommender systems are those that use a combination of collaborative filtering and content-based approaches. They aim to balance the drawbacks of each individual approach and thereby achieve better performance. A detailed review of the various hybrid approaches is presented in [4]. [5] propose a hybrid recommender system where item categories are used as a means of determining user preferences. While this method is useful in domains involving homogeneous items, it is not well suited for complex industrial use cases, such as ours, where the customer preferences are guided by implicit requirements of the solution currently being built.

To tackle these use cases, [7] proposes RESCOM – a multi-relational recommender system, which combines historical data about solutions along with context information about items in a joint knowledge base. More concretely, RESCOM employs RESCAL (Sect. 2.3) to generate recommendations by projecting partial solutions into the corresponding low-dimensional space. However, RESCOM suffers from one shortcoming: while acting as a bilinear model for capturing interactions among items, RESCOM becomes linear in the recommendation step. Thus, it is not capable of capturing complex non-linear interactions among the items.

### 3.2 Autoencoders

Autoencoders (AE) are unsupervised neural networks wherein the output aims to reconstruct the input. While different versions of AE have been proposed [2,12,14], they all share two main building blocks: an encoder and a decoder. The encoder network $g : \mathbb{R}^n \rightarrow \mathbb{R}^d$, with $d \ll n$, takes a given input $x \in \mathbb{R}^n$ and maps it to a condensed representation $g(x)$. Subsequently, the decoder network $f : \mathbb{R}^d \rightarrow \mathbb{R}^n$ takes $g(x)$ as input and maps it to a reconstructed version of $x$. During training, the parameters of $f$ and $g$ are tuned such that $f\big(g(x)\big) \approx x$.

AE have also become a popular tool for collaborative filtering. [15] use AE to construct a hybrid recommender system. They incorporate side information by

including item features at various layers of the AE. On a similar note, [6] propose a stacked AE that simultaneously encodes item and user features. It achieves good performance on two benchmark datasets with rather homogeneous items and relatively few features. In this sense, their use case significantly differs from ours: we consider items belonging to different categories (e.g., controllers, panels, and power supplies) and possessing category-specific features (e.g., resolution of a panel or the line voltage of a power supply). Combined with the sheer amount of technical attributes that we consider, encoding all these features and processing them directly via a neural network is not feasible.

## 4 Our Method – NECTR

In this paper, we propose *NECTR*, a recommender system consisting of two main components. One component is a knowledge graph that enables our model to structure context information about items. We make use of this information for recommendation purposes via a tensor factorization method described below. The second component consists of a neural network that acts as an encoder for solutions. We combine the two constituent components analogous to an AE structure resulting in a hybrid recommender engine. First, we introduce the problem setup in which *NECTR* was developed. Then, we proceed by describing the architecture of *NECTR*.

### 4.1 Problem Setup

This work evolved from a real-world industrial R&D project at Siemens. The project was centered around one of the major product configuration tools for automation solutions that has around 60,000 users per month. The purpose of the tool is to simplify the configuration process by providing intelligent selection wizards. Configuring an automation solution, however, is still not an easy task. As stated in Sect. 2, automation solutions can be very complex and be comprised of a wide range of subsystems and components, such as controllers, panels, and software. Each component is equipped with different features that must guarantee the operation of the overall system. Up to this point, without the help of experts, the configurator does not indicate to the user which components are relevant to the current solution. Hence, configuring a suitable solution requires domain expertise and a lot of effort. Our goal is to overcome this obstacle by either recommending a set of items that complement the users' current partial solutions or by reordering the list of all items based on their relevance (i.e., displaying the items that are most relevant first). Therefore, it is necessary to compute relevance scores for all items, such that the scores adjust dynamically depending on the components the user has already configured in the partial solution.

To reach this goal, we consider implicit feedback as training, validation, and testing data. For our purpose, a solution is represented as a vector $\mathbf{x} \in \mathbb{N}^{n_I}$, where $n_I$ denotes the total number of configurable items. The $i$-th entry of $\mathbf{x}$

indicates the multiplicity of the corresponding item (i.e., the number of times an item has been added to the current solution). For training and testing purposes, a partial solution $\hat{\mathbf{x}} \in \mathbb{N}^{n_I}$ is obtained by randomly masking an observed solution by setting half of the positive entries to zero. A model is trained and evaluated with regards to its ability to recover the original solution $\mathbf{x}$ (i.e., to assign a high relative score to the masked items).

The predictive accuracy of a method is measured in terms of metrics that directly relate to our practical use case (see Sect. 5.2 for more details). In other words, the quality metrics, i.e., Mean Rank, Mean Reciprocal Rank, and Hits@k, measure the model's ability to accurately rank items. Furthermore, the scores obtained can be used either to explicitly recommend the highly ranked items or to sort the product list in decreasing order of scores, thereby placing the most relevant items on top. Hence, a model performing well with respect to the above mentioned metrics is also likely to be of practical use when integrated into a solution configurator.

### 4.2 Architecture

We merge both historical data concerning past automation solutions and technical information about components from an industrial database to form a joint multi-relational knowledge graph with entity set $\mathcal{E}$ and relation set $\mathcal{R}$ (recall Sect. 2.3). Moreover, we denote with $\mathbf{X} \in \mathbb{R}^{n_E \times n_E \times n_R}$ the corresponding adjacency tensor. In our use case, an entity can be a particular solution, a particular item, or a technical attribute (e.g., the category or the line voltage) describing an item (see Fig. 2 for an example). Relations are used to describe connections between entities. For example, solutions may be linked to items via the *contains*-relation, which indicates the items that are configured in a particular solution. Other relation types link items to their respective technical attributes (see the next Sect. 5 for more details about the data). To ease the notation, we denote with $\mathcal{I} \subset E$ the set of all entities that correspond to configurable items.

We use RESCAL (as described in Sect. 2.3) to generate embeddings for all entities contained in the factor matrix $E$. In particular, $E_{\mathcal{I}}$ contains the latent features of all configurable items. Hence, if items are similar from a technical point of view or if they are often configured together, they will be close to each other in the latent feature space. In RESCAL, we employ the modified loss function

$$\mathcal{L}_C = \sum_{\substack{(e_i, r_k, e_j) \\ (e_i, e_j) \in \phi_{r_k}^{-1}(\{1\})}} (1 - E_{i,:}\mathbf{R}_{:,:,k}E_{j,:}^T)^2 + (E_{i,:}\mathbf{R}_{:,:,k}E_{l,:}^T)^2 \,, \tag{3}$$

where the second summand corresponds to a negative sample drawn from a contrastive distribution. Thereby, we draw the $l$-th entity in the negative sample uniformly at random from the set of entities that appear as object in an observed triple with respect to the $k$-th relation. This sampling procedure can be interpreted as an implicit type constraint which teaches the algorithm to discriminate between known triples on one side and unknown but plausible triples

on the other side. The main advantage of this formulation is that we can deal with very high levels of sparsity since Eq. (3) balances positive and negative examples. Since our data is extremely sparse (in fact, only about $3 \cdot 10^{-6}\%$ of the entries in $\mathbf{X}$ are non-zero), we empirically found that minimizing the loss given by Eq. (2) results in fitting mostly the zeros in $\mathbf{X}$ leading to degenerate results.

In order to obtain embeddings for the automation solutions, we process the partial solutions $\hat{\mathbf{x}} \in \mathbb{R}^{n_I}$ (see Sect. 4.1) via a feed forward neural network. More concretely, we use a dense neural network $g : \mathbb{R}^{n_I} \to \mathbb{R}^d$ in order to extract high-level representations of solutions that capture non-linear interactions between the configured items. For simplicity, we constrain the output dimension of all hidden layers to be equal to $d$. Further, we employ ReLU activation functions in all hidden layers. No non-linearity is applied in the output layer.

Moreover, consider the binary version $\mathbf{y} \in \{0, 1\}^{n_I}$ derived from $\mathbf{x}$ with $\mathbf{y}_i = \mathbb{1}_{\mathbf{x}_i > 0}$. That means each entry of $\mathbf{y}$ indicates whether or not the corresponding item is contained in the solution. *NECTR* is fitted such that it recovers the original, binary solution $\mathbf{y}$ from the partial solution $\hat{\mathbf{x}}$ containing the item counts. The underlying rationale behind this setup is that we want *NECTR* to learn whether an item is relevant in the context of a partial solution and in turn recommend this item to the user. Considering the actual counts in $\mathbf{x}$ as target variables would have resulted in an unwanted bias towards items that are frequently configured in large number (such as cables or sockets).

In a regular AE, one would employ a decoder network which mirrors the action of $g$ in order to obtain a reconstruction of $\hat{\mathbf{x}}$. We, however, introduce a so-called context layer which induces side information into the AE architecture. This is realized by multiplying the latent representation of $\hat{\mathbf{x}}$ with the item embeddings obtained from the tensor factorization, followed by a non-linearity. More concretely, we obtain

$$\sigma(E_{\mathcal{I}} g(\hat{\mathbf{x}})) =: \tilde{\mathbf{y}}, \tag{4}$$

where $\sigma(\cdot)$ denotes the sigmoid function. Then *NECTR* is fitted such that the output approximates the original, binary solution, i.e., $\tilde{\mathbf{y}} \approx \mathbf{y}$. Entries of the reconstruction can be interpreted as scores indicating the relevance of an item for a given partial solution. They can be used to produce a ranked list and items that have the highest rank can be recommended to the user.

Equation (4) can be interpreted as an implicit similarity search in the latent feature space, where the discrepancy between solutions and items is measured in terms of the cosine distance. However, this might not be appropriate in a setting where the items are very heterogeneous and belong to different categories. Moreover, [7] find that after fitting the parameters of RESCAL, items of industrial automation solutions cluster in the latent feature space according to their categories. That is why it may not be appropriate to consider a single latent representation of a solution and compare it to all items across different categories. As a remedy to this problem, we propose category-specific transformations of the solution and compute the reconstructed version for each category separately.

To ease the notation, let $\text{Cat}_i \subset \mathcal{I}$ denote the index set of items that belong to the $i$-th category. A category-specific reformulation of Eq. (4) is given by

$$\sigma(E_{\text{Cat}_i} A_i g(\hat{\mathbf{x}})) =: \tilde{\mathbf{y}}_{\text{Cat}_i}, \tag{5}$$

where $A_i \in \mathbb{R}^{d \times d}$ denotes a category-specific linear mapping. The reconstructed solution $\tilde{\mathbf{y}}$ is obtained by concatenating all $\tilde{\mathbf{y}}_{\text{Cat}_i}$. Note that this alternative formulation does not lead to an increase in the computational complexity. We refer to the resulting method as $NECTR_{cat}$.

The parameters of the AE-like component are obtained by minimizing the recommendation loss given by

$$\mathcal{L}_R = -\sum_{i=1}^{n_S} \sum_{j=1}^{n_I} \mathbf{y}_j^{(i)} \log(\tilde{\mathbf{y}}_j^{(i)}) + (1 - \mathbf{y}_j^{(i)}) \log(1 - \tilde{\mathbf{y}}_j^{(i)}), \tag{6}$$

where $n_S$ denotes the number of solutions in the training data. The index sets of the two sums are ranging over all solutions and all items, respectively. Further, after consulting domain experts in the field of industrial engineering solutions, we conjectured that nonlinear interactions between items are present, but not necessarily numerous. That is the reason for adding an additional $L_1$-regularizer on the weight matrices of $g$ to induce sparsity.

The overall loss function of our recommender system is given by

$$\mathcal{L} = \mathcal{L}_R + \lambda \mathcal{L}_c + \text{regularization term}, \tag{7}$$

where $\lambda > 0$ balances the completion and the recommendation loss. This leads to the interpretation that the inclusion of the knowledge graph embeddings acts as a context-aware regularization. The whole model can be trained end-to-end by minimizing the loss given by Eq. (7) using stochastic gradient descent. Figure 1 summarizes the architecture of *NECTR*.

## 5 Real-World Experimental Study

### 5.1 Data

The evaluation has been performed on the real-world dataset collected in the context of an internal R&D project at Siemens. We distinguish between two sources of information: historical shopping data and product descriptions. Historical shopping data contains the specifications of 49,829 engineering solutions that have been configured by customers in the past. While some additional structural information is available, for the purposes of this study, we treat each engineering solution as a list of products used along with their respective quantities. Each of the 6,136 products that has been used in one of the historical solutions comes with a spec sheet, which summarizes its technical characteristics. Further, we distinguish between 5,062 possible attributes – with only a small part being relevant for any given item. One example of an attribute of particular importance is the product category. We distinguish between 8 categories, namely Industrial Communication, IPC, I/O System, Panel, Cables and Plugs, Software, Controller, and Industrial Control. In certain cases, a more granular categorization is available, e.g. Mobile Panel, Basic Panel, and Comfort Panel.

### 5.2 Evaluation

To guarantee a fair evaluation in a realistic setting, we follow the same experimental setup and evaluation scheme: First, we split the solutions in our dataset into training, validation, and test sets in the ratio 70-20-10. In particular, solutions that are assigned to the validation or test set are neither included in the knowledge graph nor are processed by the neural network $g$ during training. Next, for each method, we perform cross-validation in order to choose the most suitable set of hyperparameters, i.e., the one yielding the highest mean reciprocal rank. The number of the latent dimensions for the embeddings is chosen from the range $\{10, 30, 50, 70, 90\}$. For the neural network, the number of hidden layers and neurons per layer are chosen among $\{1, 2, 3\}$ and $\{5, 15, 25, 35, 45, 50\}$, respectively. Moreover, the $L_1$-regularization strength was set to 0.01 and $\lambda$ was chosen from $\{0.01, 0.1, 1\}$. Equation (7) was minimized using Adam with learning rate given by 0.01. Finally, we evaluate each method on the test set using the best hyperparameters and report the results.

For evaluation, we construct an adjacency matrix of solutions and configured items. For each solution, we randomly choose half of the configured items and mask the corresponding entries. Let $\mathcal{M} = (s, i)$ denote the set of masked items per solution and $n_M := |\mathcal{M}|$ denotes the number of masked items. Further, since we find that the solution data is rather skewed, we perform a log-transformation on the raw-input (i.e., the item counts). We then perform a forward pass through the neural network followed by a multiplication with the item embeddings obtained from the tensor factorization (see Eq. (4)), to obtain a completed matrix. Each row in this matrix corresponds to a solution and the entries in each row are interpreted as scores that determine the likelihood of configuration of the corresponding items in the solution. Based on this interpretation, the items are reordered row-wise, in decreasing order of their scores, resulting in a rank $R_{i,s}$ for each item. Based on these predicted ranks, we compute the following performance metrics:

- Mean rank – Average of the ranks predicted for the items, given by

$$\text{Mean rank} = \frac{1}{n_M} \sum_{(s,i)\in\mathcal{M}} R_{i,s}.$$

- Mean reciprocal rank (MRR) – Average of the multiplicative inverse of the predicted ranks, as given by

$$\text{MRR} = \frac{1}{n_M} \sum_{(s,i)\in\mathcal{M}} \frac{1}{R_{i,s}}.$$

  The choice of the MRR is motivated by its stronger robustness to outliers as compared to the Mean rank.
- Hits@k – The proportion of correct entities ranked in the top $k$, i.e.,

$$\text{Hits@k} = \frac{|R_{i,s} < k|}{n_M}.$$

  More specifically, we report the values for k = 1, 3, 5, 10, and 10%.

Furthermore, to tackle the bias introduced by the true triples (as noted in [3]), we report all the above metrics in two settings: raw (where all triples are included in ranking) and filtered (where all true triples, except the test triple in consideration, are excluded from the ranking).

### 5.3 Results

Table 1 displays the results of the recommendation task for all methods under consideration. As described previously in Sect. 1, matrix factorization, especially non-negative matrix factorization (NMF), has consistently produced good results in recommender systems. So we include results of evaluating NMF in our experimental setting. Recommending items corresponds to proposing new edges of type contains in the knowledge graph (see Fig. 2). Hence, the recommendation task is equivalent to the graph completion task restricted to proposing links between solutions and items. That is the reason for also evaluating TransE [3], a translational model popularly used for link prediction. In addition, we present the results of evaluating RESCOM – a tensor factorization-based recommendation engine developed in the context of engineering solutions (see Sect. 3). The best hyperparamters for *NECTR* (*NECTR*$_{cat}$) are determined via the cross-validation setting described in the previous subsection. The reported results correspond to the following hyperparameter values: The number of latent dimensions is 90 (30), the number of hidden layers is equal to 1 (3), the number of hidden neurons are given by 45 (30), $\lambda = 0.1$ (0.01), and the $L_1$-regularization strength is equal to 0.01 (0.01).

**Table 1.** Results of all evaluated methods on the test set of the automation solution dataset.

| Metric | Mean rank | | MRR | | Hits@10% | | Hits@1 | | Hits@3 | | Hits@5 | | Hits@10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Eval. setting | Raw | Filt. | Raw | Filt. | Raw | Filt. | Raw | Filt. | Raw | Filt. | Raw | Filt. | Raw | Filt. |
| TransE | 392.01 | 385.79 | 0.06 | 0.07 | 0.73 | 0.83 | 0.02 | 0.02 | 0.05 | 0.07 | 0.08 | 0.11 | 0.14 | 0.16 |
| NMF | 223.2 | 215.87 | 0.10 | 0.25 | 0.92 | 0.92 | 0.016 | 0.16 | 0.08 | 0.27 | 0.17 | 0.32 | 0.31 | 0.42 |
| RESCOM | 160.63 | 152.66 | 0.08 | 0.19 | **0.95** | **0.96** | 0.003 | 0.08 | 0.05 | 0.2 | 0.13 | 0.28 | 0.28 | 0.41 |
| *NECTR* | 193.04 | 185.20 | 0.13 | **0.31** | 0.94 | 0.94 | 0.03 | **0.22** | 0.11 | **0.34** | 0.20 | **0.40** | **0.37** | **0.50** |
| *NECTR*$_{cat}$ | **148.82** | **142.17** | **0.16** | 0.27 | **0.95** | 0.95 | **0.06** | 0.18 | **0.16** | 0.28 | **0.23** | 0.35 | 0.36 | 0.46 |

The filtered setting produces lower Mean ranks and higher values of MRR and Hits@k for all the methods, as expected. As described in Sect. 5.2, this setting offers a better evaluation of the real-world performance. *NECTR* (or *NECTR*$_{cat}$) outperforms all methods in all of the standard metrics except for Hits@10%. In particular, the performance gains measured by MRR and Hits@k are substantial. Related to our use case, this indicates that *NECTR* is capable of placing relevant

items on top of the list with the highest probability. Furthermore, we find that $NECTR_{cat}$ outperforms all other methods with respect to the Mean Rank by a large margin, while yielding comparable results to *NECTR* with respect to the other performance measures.

The large improvements that *NECTR* achieves compared to other methods show the presence of non-negligible, higher order interactions as well as the necessity to consider context information. RESCOM deviates from *NECTR* in the sense that it employs RESCAL directly to produce recommendations (see Sect. 3). Thus, RESCOM is not able to model non-linear effects among the configured items. Hence, the large performance gains of *NECTR* can be attributed to this feature. Moreover, *NECTR* also comprises a model that does not take context information into account. This is achieved by setting $\lambda$ in Eq. (7) to zero. We found that this model is strictly dominated by a context-aware version of *NECTR* with respect to all performance measures (result not shown).

### 5.4 Additional Experiments

In this section, we evaluate and compare *NECTR* on the standard benchmark dataset Movielens 1M.[2] After preprocessing, it contains around 1 million ratings from 6,038 users on 3,533 movies along with demographic information about users (age, occupation, and gender) and the genres of the movies. Movielens 1M differs from the automation solution dataset in the sense that the items under consideration (movies) are stand-alone homogeneous items. Further, the context information is a lot less rich and less diverse (recall, items in the automation solution use case come with more than 5,000 different attributes). Thus, we do not expect that *NECTR* can demonstrate its advantages as in the automation solution use case. Following [6], we binarize explicit data by keeping the ratings of four or higher and interpret them as implicit feedback that the user liked the corresponding movies. In order to inject the context information about users into *NECTR*, we consider the implicit feedback movies liked by a user concatenated with the demographic information about the corresponding user as input vector $\hat{\mathbf{x}}$ to the neural encoder $g$. The experimental setting and the evaluation scheme are identical to the previous subsections. The chosen hyperparamters for *NECTR* are as follows. The number of latent dimensions is 10, the number of hidden layers is equal to 2, the number of hidden neurons are given by 40, $\lambda = 0.1$, and the $L_1$-regularization strength equals 0.01.

The results are presented in Table 2. As argued above, given the limited complexity of the data, it cannot be expected that *NECTR* leads to similar performance gains like in Sect. 5.3. Nevertheless, *NECTR* outperforms all other methods with respect to the Mean rank by a large margin and leads to a comparable performance with respect to the other metrics.

[2] https://grouplens.org/datasets/movielens/1m/.

**Table 2.** Results of all evaluated methods on the MovieLens 1M dataset.

| Metric | Mean rank | | MRR | | Hits@10% | | Hits@1 | | Hits@3 | | Hits@5 | | Hits@10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Eval. setting | Raw | Filt. | Raw | Filt. | Raw | Filt. | Raw | Filt. | Raw | Filt. | Raw | Filt. | Raw | Filt. |
| TransE | 642.90 | 539.77 | 0.01 | 0.02 | 0.44 | 0.51 | 0.001 | 0.002 | 0.004 | 0.008 | 0.01 | 0.01 | 0.02 | 0.03 |
| NMF | 556.43 | 440.15 | 0.012 | 0.06 | 0.58 | 0.68 | 0.0002 | 0.02 | 0.002 | 0.05 | 0.004 | 0.08 | 0.02 | 0.12 |
| RESCOM | 431.18 | 316.62 | 0.02 | **0.07** | **0.66** | **0.77** | 0.002 | **0.03** | 0.008 | **0.07** | 0.02 | **0.1** | 0.04 | **0.15** |
| *NECTR* | **398.87** | **294.36** | **0.03** | **0.07** | 0.64 | 0.74 | **0.006** | **0.03** | **0.02** | 0.06 | **0.03** | 0.08 | **0.05** | 0.14 |

## 6 Conclusion

Traditional collaborative filtering methods are not expressive enough in a setting where multiple components form a complex system and technical properties of items play a crucial role. This is the case for industrial engineering solutions where the individual components interact with each other, resulting in higher-order dependencies. Popular methods, such as matrix factorization, are not able to capture such non-linear effects. To address these shortcomings, we have proposed *NECTR* – a novel recommender system, which combines deep representation learning and context information structured in a knowledge graph. While a tensor factorization exploits context information of the available items, an autoencoder models the non-linear dependencies inherent to automation solutions.

We conducted extensive experiments on a real-world dataset for industrial automation solutions. The main findings are:

1. *NECTR* outperforms all baseline methods with respect to most metrics. This shows the presence of non-linear interactions as well as the necessity to include context information.
2. *NECTR* can be trained efficiently end-to-end. Since the recommendation step is performed via a simple forward pass through a neural network, *NECTR* can efficiently work in real-time. This is crucial with regard to its practical applicability.

Finally, we would like to stress that this work evolved from a real-world industrial R&D project. In the following months, we will take further steps towards integrating *NECTR* into one of the major industrial engineering solution configurators at Siemens.

## References

1. Bell, R.M., Koren, Y., Volinsky, C.: The Bellkor 2008 solution to the Netflix prize. Statistics Research Department at AT&T Research 1 (2008)
2. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. ArXiv e-prints, June 2012
3. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in Neural Information Processing Systems, pp. 2787–2795 (2013)

4. Burke, R.: Hybrid recommender systems: survey and experiments. User Model. User-Adapt. Interact. **12**(4), 331–370 (2002)
5. Choi, S.M., Han, Y.S.: A content recommendation system based on category correlations. In: 2010 Fifth International Multi-Conference on Computing in the Global Information Technology (ICCGI), pp. 66–70. IEEE (2010)
6. Dong, X., Yu, L., Wu, Z., Sun, Y., Yuan, L., Zhang, F.: A hybrid collaborative filtering model with deep structure for recommender systems. In: AAAI, pp. 1309–1315 (2017)
7. Hildebrandt, M., Sunder, S.S., Mogoreanu, S., Thon, I., Tresp, V., Runkler, T.: Configuration of industrial automation solutions using multi-relational recommender systems. In: Brefeld, U., et al. (eds.) ECML PKDD 2018. LNCS (LNAI), vol. 11053, pp. 271–287. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-10997-4_17
8. Hinton, G., Deng, L., Yu, D., Dahl, G.: Deep neural networks for acoustic modeling in speech recognition. Signal Process. Mag. **29**, 82–97 (2012)
9. Krizhevsky, A.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, vol. 25 (2012)
10. Linden, G., Smith, B., York, J.: Amazon.com recommendations: item-to-item collaborative filtering. IEEE Internet Comput. **7**(1), 76–80 (2003)
11. Lops, P., de Gemmis, M., Semeraro, G.: Content-based recommender systems: state of the art and trends. In: Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.) Recommender Systems Handbook, pp. 73–105. Springer, Boston (2011). https://doi.org/10.1007/978-0-387-85820-3_3
12. Meng, Q., Catchpoole, D., Skillicorn, D., Kennedy, P.J.: Relational autoencoder for feature extraction. ArXiv e-prints, February 2018
13. Nickel, M., Tresp, V., Kriegel, H.P.: A three-way model for collective learning on multi-relational data. In: ICML, vol. 11, pp. 809–816 (2011)
14. Rifai, S., Vincent, P., Muller, X., Glorot, X., Bengio, Y.: Contracting auto-encoders: explicit invariance during feature extraction. In: Proceedings of the Twenty-Eight International Conference on Machine Learning (ICML 2011) (2011)
15. Strub, F., Gaudel, R., Mary, J.: Hybrid recommender system based on autoencoders. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pp. 11–16. ACM (2016)
16. Weston, J., Chopra, S., Adams, K.: TagSpace: semantic embeddings from hashtags. In: Empirical Methods in Natural Language Processing (2014)