RL^2: A Call for Simultaneous Representation Learning and Rule Learning for Graph Streams

Qu Liu
University of Massachusetts, Lowell
Lowell, Massachusetts, USA
qliu@cs.uml.edu

Tingjian Ge
University of Massachusetts, Lowell
Lowell, Massachusetts, USA
ge@cs.uml.edu

ABSTRACT

Heterogeneous graph streams are very common in the applications today. Although representation learning has advantages in prediction accuracy, it is inherently deficient in the abilities to interpret or to reason well. It has long been realized as far back as in 1990 by Marvin Minsky that connectionist networks and symbolic rules should co-exist in a system and overcome the deficiencies of each other. The goal of this paper is to show that it is feasible to simultaneously and efficiently perform representation learning (for connectionist networks) and rule learning spontaneously out of the same online training process for graph streams. We devise such a system called RL^2, and show, both analytically and empirically, that it is highly efficient and responsive for graph streams, and produces good results for both representation learning and rule learning in terms of prediction accuracy and returning top-quality rules for interpretation and building dynamic Bayesian networks.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning approaches**.

KEYWORDS

graph streams, representation learning, rules

ACM Reference Format:
Qu Liu and Tingjian Ge. 2022. RL^2: A Call for Simultaneous Representation Learning and Rule Learning for Graph Streams. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539309

1 INTRODUCTION

Graphs are a general and powerful data model, as nearly all data and information can be directly represented as graphs, but a significant portion of it is unwieldy and inefficient, if not impossible, to be represented and manipulated as pure relational tables. Moreover, their flexible schema makes it very easy for data integration from heterogeneous sources. Graph streams, where streaming items are edges [20], are a generalization and flexible representation of relational data streams. In general, such a system may contain both static and dynamic (streaming) edges/relations, capturing interaction events in applications such as social networks, communication networks, traffic, healthcare, among others. Here is an example.

Example 1. *Medical data for patients in the Intensive Care Units (ICU) of hospitals is highly dynamic and rich, and is a heterogeneous graph stream. Specifically, we look into the MIMIC-III (Medical Information Mart for Intensive Care III) data [14] comprising deidentified health-related data associated with over forty thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012. Figure 1 illustrates a snippet of the data from this dataset. First of all, there are many types of entities, shown as vertices in the graph, including patients, diagnosis, procedures, lab events, input events, and prescription drugs. They are shown as nodes of different colors. Then there are both static relationships (solid edges)and dynamic relationships (dashed edges). Static edges are between patients and diagnosis nodes (e.g., Septic shock) and between patients and procedures (e.g., Vascular cath irrigation)—indicating the "properties" of ICU patients without a timestamp. Interaction relations such as lab measurement events and input events (fluids administered to a patient) are very dynamic and timestamped. Two patients may have relations with the same diagnosis, or they may have the same input events but at different times. Learning from such heterogeneous data sources is instrumental for inference and reasoning.*

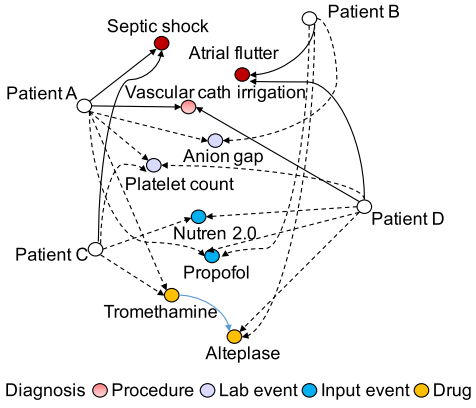

Figure 1: A heterogeneous graph stream system in a hospital.

1.1 Representation Learning & Rule Learning, Symbolic & Connectionist, in One System

Known as the "father of artificial intelligence", Marvin Minsky, the builder of the first neural network simulator (in 1951), wrote a highly visionary and insightful article even far back in 1990 [22], which observes that "Our purely numerical connectionist networks

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539309

are inherently deficient in abilities to reason well; our purely symbolic logical systems are inherently deficient in abilities to represent the all-important 'heuristic connections' between things—the uncertain, approximate, and analogical linkages that we need for making new hypotheses".

Here, the *connectionist networks* refer to neural networks and representation learning that render the best predictive accuracy in many tasks today, while *symbolic systems* refer to symbolic representation of knowledge using logic symbols and rules. Minsky [22] further establishes that "*The versatility that we need can be found only in larger-scale architectures that can exploit and manage the advantages of several types of representations at the same time. Then, each can be used to overcome the deficiencies of the others*".

This vision over three decades ago has also been echoed more recently by some researchers [10, 16]. In particular, recent work from MIT, IBM, and DeepMind [19] shows the power of combining the two approaches, by building a neuro-symbolic concept learner (NS-CL) that has two neural networks for scenes and text-based question-answer pairs, respectively, and a symbolic rule system to answer new questions about a different scene—a type of feat that is far more challenging than can be done using a connectionist approach alone. It is also observed that while connectionist networks may get good accuracy, it is inherently hard for them to explain how they arrive at a solution; for many critical decision-making, symbolic rule systems can help interpret the prediction results.

In a nutshell, connectionist networks are good at learning weak and uncertain associations among data entities, while symbolic/rule systems are strong at interpretation, reasoning, and working with less data. A system would benefit greatly from having both architectures seamlessly integrated in an efficient and natural manner. **The goal of this paper is to show that it is feasible to simultaneously and efficiently perform representation learning (for connectionist networks) and rule learning spontaneously out of the same online training process for graph streams.** Note that we pay special attention to real-time throughput and efficiency for high-speed streams.

To the best of our knowledge, we are the first to study the above feasibility problem in the context of graph streams. As such, it is not our intent to propose a new representation learning scheme for graph streams; nor is it to improve the predictive accuracy. But rather, we adapt an efficient graph representation learning method (TransE [3])—which satisfies our real-time high-throughout requirement of graph streams—with three key novelties: (1) treating the rules that we learn as a *new type of relationship*; (2) devising aggregated subgraph embeddings using *attention* [32] for the head/tail entities of TransE; and (3) *calibrating the probabilities* of the conditional distribution rules by solving an optimization problem **(Sec. 3)**.

The rules that we learn are *temporal conditional distribution* (TCD) rules with respect to the attributes surrounding a graph stream event/edge, in the form $X \rightarrow_t Y$, where X is one or more attributes, Y is a single attribute, $\rightarrow_t$ indicates the time delay between the two sides, and such a rule specifies the conditional distribution $Pr[Y|X]$ (details are in Sec. 2.1). For instance, a TCD rule that we discover from Example 1 contains "Warfarin (= *yes*), Lisinopril (= *yes*) $\rightarrow_+$ Phytonadione (= *yes*)" with a high probability. Indeed, Warfarin and Lisinopril treat high blood pressure and prevent blood clots in veins, and doctors often use Phytonadione later ($\rightarrow_+$ indicates the time delay) to reduce the side effect of Warfarin (excessive blood thinning).

Potential TCD rule candidates can be of a large number; it is unnecessary and impractical to learn all of them. We propose a quality score function using an *information theoretic* measure. Then we devise an efficient online algorithm to continuously and adaptively train and select top-k quality-score rules at any time. The main idea of our online training/selection is to treat the problem as a *Multi-Armed Bandit* (MAB) [34]. Using *Markov chain coupling* [23], we show that our algorithm for solving it is provably fast **(Sec. 4)**.

We perform a systematic experimental evaluation using two large real-world graph stream datasets in different domains, comparing against four baselines. The results show that our RL^2 can efficiently and effectively do representation learning and rule learning together. While prediction accuracy is not our main goal, RL^2's prediction accuracy is competitive with deep learning methods that do not learn rules, but has a two or more orders of magnitude higher throughput, in addition to the obtained TCD rules which can be used for result interpretation and building dynamic Bayesian networks (DBN) **(Sec. 5)**. As a use case of the learned rules, we design an algorithm to retrieve rules to build a DBN in the Appendix. Our code and datasets are accessible at [8, 14, 27].

2 PROBLEM STATEMENT & RELATED WORK

2.1 Problem Statement

We consider a *graph stream* $\mathcal{G} = (\mathcal{E}, \mathcal{R})$ over entities $\mathcal{E} = \{\varepsilon_1, \ldots, \varepsilon_n\}$, each of which corresponds to a vertex. In general, two entities may be connected by either a *static relationship* (between head entity ε_h and tail entity ε_t) or a *dynamic relationship* (i.e., an interaction between ε_h and ε_t at time t) in $\mathcal{R}$ as a directed edge from ε_h to ε_t. There is a set of schemas $\sigma_1, \ldots, \sigma_k$, each of which has a number of attributes from the entities and/or relationships. Some schemas contain attributes specific to one entity, while others across two entities—for the interaction relations between two entities. In general, the attributes A of a schema σ consists of three disjoint subsets $A = A_h \cup A_t \cup A_r$, where A_h is the attributes at the head entity, A_t is at the tail entity, and A_r is from a relationship, e.g., the timestamp. The *temporal conditional distribution* rules (defined below) will be with respect to such attributes A surrounding a graph stream event/edge.

For such a graph stream, a relational representation learning (a.k.a. knowledge graph embedding) method will generate embedding vectors for each $\varepsilon_i \in \mathcal{E}$ and each relationship $r_i \in \mathcal{R}$, i.e., producing a function $emb : \mathcal{E} \cup \mathcal{R} \mapsto \mathbb{R}^d$, where d is the dimensionality of the embedding vectors. A *temporal conditional distribution* (TCD) rule is of the form $X_1, ..., X_\ell \rightarrow_t Y$, where X_i, Y are attributes, $\ell \geq 1$, and t can be 0, +, or −, indicating the time relationship between the two sides. The three t values correspond to Y being "within the same time window" (of size δ) as X_i's, "in the next time window", and "in the previous time window", respectively, where δ is application dependent. Each attribute has a finite set of values (e.g., a numeric attribute is discretized into ranges).

A TCD rule specifies a *distribution* of Y conditioned on X_i's, with the corresponding time delay relationship. Each rule describes $Pr[Y|X_1, ..., X_\ell]$, i.e., for every $X_1 = x_1, ..., X_\ell = x_\ell$, it specifies

$Pr[Y = y|x_1, ..., x_\ell]$ that sums up to 1 for all y's. Note that such rules are clearly different from the *association rules* in the literature [29], which do not cover all combinations of X_i's and Y, nor the time relations. For brevity, we also write a rule as $X \rightarrow_t Y$, with X representing one or more attributes. For such a rule ρ, we will design a *quality score* function $q(\rho)$, and adaptively return the top-k rules with the highest scores.

Our problem can be summarized as follows: Given a heterogeneous graph stream $\mathcal{G}$, we simultaneously (1) perform representation learning to produce an embedding function *emb*, and (2) perform rule learning and adaptively return top-k rules with the highest quality scores at any time.

2.2 Related Work and Preliminaries

Some closely related work is presented in Section 1.1; here we survey others.

Representation Learning on Graphs, a.k.a. graph embedding or graph neural networks [5], maps each node (and/or relationship type) to a point (i.e., embedding vector) in a multidimensional space. Each dimension of an embedding vector is a latent feature learned from data.

Multi-Armed Bandit (MAB) Problem [34] is a problem in which a fixed limited set of resources must be allocated between competing (alternative) choices in a way that maximizes their expected gain, when each choice's properties are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to the choice. Thompson sampling [30] is one of the methods to solve the MAB problem. The basic idea is that in each choice round, the player has a belief of the *posterior distribution* of the reward for choosing each of the arms/actions. Then the player samples from each of those distributions and chooses the arm/action that gives the maximum reward sample. This will maximize the expected reward.

Interpretation of graph embedding and prediction has been studied [17, 18, 33]. For example, Liu et al. [18] perform global-view interpretation by constructing a taxonomy of the embedding instances based on hierarchical clustering in the latent space, as well as local-view interpretation by summarizing the unique characteristics in each cluster. In [17], Liu et al. propose an interpretation framework to understand and describe how representation vectors distribute in the latent space, and then a multimodal autoencoder is built for generating the description of a representation instance. Wang et al. [33] generate path representations in knowledge graphs and use paths to infer the underlying rationale of a user-item interaction. The previous work either only tries to make sense of the embedding space and the locations of instances in the space, or uses coarse-grained path structures in the original graph. One use of our TCD rules is to interpret prediction results. In addition, TCD rules incorporate time delays for symbolic reasoning and can be used to build dynamic Bayesian networks.

3 SIMULTANEOUS REPRESENTATION LEARNING AND RULE LEARNING

In this section, we adapt TransE [3] for representation learing and TCD rule learning. In Section 4, we will study how to judiciously select rules to train.

We first discuss how we collect a set of rule candidates based on the stream tuples. For each distinct schema σ that appears in the graph stream $\mathcal{G}$, the left-hand-side (LHS) of a rule can have between 1 and ℓ attributes (where ℓ is the maximum length of LHS), while the right-hand-side (RHS) has exactly 1 attribute (a rule with multiple attributes on the RHS can be decomposed into multiple rules, each with 1 attribute on the RHS). The timing relationship of a rule can be one of the three: $\rightarrow_0$, $\rightarrow_+$, and $\rightarrow_-$. Thus, if there are k distinct schemas σ_i ($1 \le i \le k$), then the number of rule candidates is $3 \sum_{i=1}^{k} \sum_{j=2}^{\ell+1} j\binom{|\sigma_i|}{j}$.

For example, if there are 10 schemas, each of which has 8 attributes, and if the size limit of LHS is 3, then there are 15,120 candidate rules. Moreover, each rule has a number of attribute-value combinations in its conditional probability distribution to be learned. Thus, there can be a large number of rule dependencies to be learned. Since the training data in a stream window is likely sparse at least for some of the rules, we will use a representation learning approach that has generalization capacity.

3.1 Basic Embedding Method

We first enhance the dynamic graph $\mathcal{G}$ associated with the graph stream. Recall that each attribute ϕ is associated with a set of values $\{v_1, v_2, \ldots, v_{|\phi|}\}$. We create a vertex in $\mathcal{G}$ for each attribute-value combination (ϕ, v_i), for $(1 \le i \le |\ \phi\ |)$, and connect an entity to its corresponding attribute-value nodes. In this way, it still follows the triple data model in knowledge graphs. This also facilitates learning a rule $X \rightarrow_t Y$, as X and Y each correspond to a subgraph, for which we can produce an embedding. This is illustrated in Figure 2(a) for a regular relation in the data and Figure 2(b) for a new relation associated with a rule.

We devise a primitive mechanism for embedding a subgraph served as the head/tail of a relation, which is an *attention-based aggregation* of the embedding vectors of a set of attribute-value nodes, as illustrated in Figure 2. In this work, we adapt and build on top of an efficient knowledge graph embedding method TransE [3]. Alternative embedding methods could be used too, as we treat rules as relations and the head/tail of a triple as subgraphs/pooling.

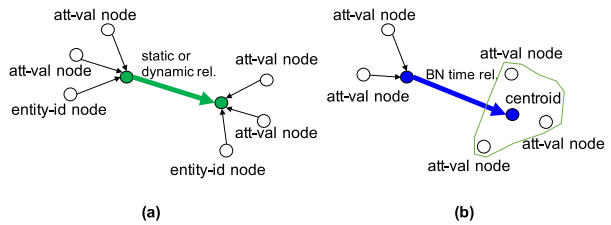

Figure 2: (a) Embedding training of a static or dynamic relationship edge. Head and tail embedding vectors are the aggregate embeddings of attribute-value nodes using attention and connection relations. Head and tail also each contains an entity-id node's embedding. (b) Training of a rule. Head is similar but without an entity-id node, while tail is the centroid of the embedding vectors of possible values of the RHS of the rule, weighted by the probabilities.

Training Primitive. Figure 2(a) is for training over a static or dynamic relationship $r_{\mathcal{G}}$, whose head is at entity E_h and whose tail is at entity E_t. There are a set of attribute-value nodes at E_h and E_t,

respectively, including a special one "entity-id" node that is unique for the entity holding its own embedding vector. The training w.r.t. this relationship $r_{\mathcal{G}}$ is to minimize the loss

$$\| \sum_{x_i \in E_h} a_i(\mathbf{x}_i + \mathbf{r}_i) + \mathbf{r}_{\mathcal{G}} - \sum_{y_i \in E_t} a_i(\mathbf{y}_i + \mathbf{r}_i) \| \quad (1)$$

where $\sum_{x_i \in E_h} a_i(\mathbf{x}_i + \mathbf{r}_i)$ is the *attention-based aggregation* at E_h and x_i iterates over each of the attribute-value nodes, as discussed above. $\mathbf{r}_i$ is the embedding vector of the connecting relationship r_i between x_i and the aggregate node. We use $\mathcal{AG}(E_h)$ to denote this attention-based aggregation at E_h. The treatment at the tail entity E_t is similar.

Figure 2(b) illustrates the training of a rule's vectors, where the relationship is the time relationship r_t, which is r_0 (same time), r_+ (followed by), or r_- (preceded by). The head-entity side is the attention-based aggregation of Figure 2(a), except that there is no "entity-id" node, since this is the aggregation of attribute values for the LHS of a rule.

We let the tail-entity be the *centroid* of the vectors of $y = v_i$ $(1 \le i \le k)$ nodes, where y is the attribute at the RHS of the rule, k is the number of values of y, and the mass/weight of each vector is the corresponding value v_i's probability in the rule. We denote the centroid as $C(dist(Y))$, where $dist(Y)$ is the probability distribution of Y. Intuitively, a value with a greater probability should be closer to the tail vector (centroid)—in the extreme, if a particular value $y = c$ has probability 1, the centroid is right at that node. In Section 3.2, our probability recovery and calibration algorithm will be based on this mechanism, generalizing to unseen or less trained values/value combinations.

Note that our loss function in Equation 1 still follows the TransE [3] algorithm, except that the head or tail entity is a subgraph whose embedding is an attention-based aggregation or a weighted sum based on the probabilities (in the case of the tail of Figure 2(b)). Stochastic gradient descent (SGD) [4] is used to update all the embedding vectors in Equation 1, as well as all the attention parameters a_i.

Algorithm to Train a Chosen Rule. We first present the basic submodule for jointly training a specific rule once, as well as a related dynamic stream edge (relationship), and a randomly chosen static relationship edge. In Section 4, we will discuss how to iteratively and judiciously select a rule to train. This basic training module algorithm is presented in RuleTrainingOnce.

When we parse the stream tuples to identify the rule candidates in Section **??**, we associate with each rule ρ a set of relevant tuples in which ρ appears. Line 1 of the algorithm randomly chooses a tuple among them. Lines 2-3 use attention-based aggregation for head and tail entities and run SGD for a number of iterations over the relationship edge based on Eq. 1. Similarly, line 4 randomly selects a static relationship edge to train. Lines 5-6 are to sample tuples relevant to ρ and incrementally add to the frequency statistics (i.e., the frequency of the Y value given the X value), before the rule is trained in line 8.

Algorithm 1: RuleTrainingOnce $(\mathcal{G}, \rho)$

Input: $\mathcal{G}$: graph stream;
ρ: a chosen rule
Output: updated embedding of $\mathcal{G}$ and statistics of ρ

1 tuple $\tau \leftarrow$ randomly chosen from ρ's relevant tuples in $\mathcal{G}$
2 **if** *τ is graph stream tuple of relationship $r_{\mathcal{G}}$* **then**
3 SGD over triple $\{\mathcal{AG}(E_h), \mathcal{AG}(E_t), r_{\mathcal{G}}\}$ based on Eq. 1
4 randomly pick a static relation r_s and run SGD over triple $\{\mathcal{AG}(E_h), \mathcal{AG}(E_t), r_s\}$
5 select a random set of tuples T from $\mathcal{G}$ relevant to ρ
6 collect $X \rightarrow_t Y$ frequency statistics of ρ from T and add it to current frequency statistics $\mathcal{F}$ of ρ
7 **for** *each $X \rightarrow_t dist(Y)$ in $\mathcal{F}$* **do**
8 SGD over triple $\{\mathcal{AG}(X), C(dist(Y)), r_t\}$

3.2 Probability Recovery and Calibration

Probability calibration, i.e., accurately estimating the probabilities of prediction results, is a much needed but challenging topic for neural networks [24, 26, 28]. Even more intriguing in our problem is that we need to calibrate the probabilities over relations to several attribute-value nodes (i.e., the RHS Y), generalizing to untrained or less trained attribute-values due to the sparsity of training data with respect to the large number of rules and value combinations.

Recall that we treat the centroid of a number of attribute-value nodes (weighted by their probabilities) as the tail entity in a rule. As discussed earlier, the training is based on incomplete data with respect to all value combinations of all rule candidates. The idea is that the representation learning has generalization capacity and generalizes to unseen or rarely seen attribute values (in the RHS of a rule) or value combinations (in the LHS of a rule). Thus, we now discuss how we can recover and calibrate the probability distributions of the rule candidates based on the embedding vectors. We have the following result. The proofs of all the theorems are in Appendix A.

Theorem 1. *Consider a rule $X \rightarrow_t Y$, where Y has k possible values v_i $(1 \le i \le k)$ with embedding vectors $\boldsymbol{v}_i (1 \le i \le k)$, and a value combination x of the attributes in X with an aggregate embedding $\boldsymbol{x}$. Let the embedding vector for the relationship $\rightarrow_t$ be $\boldsymbol{r}$. Then setting the probability vector of the rule for x to be*

$$(p_1, \ldots, p_k) = \boldsymbol{b} \cdot \boldsymbol{A}^{-1}$$

where

$$\boldsymbol{b} = ((\boldsymbol{x}+\boldsymbol{r}) \cdot (\boldsymbol{v}_1 - \boldsymbol{v}_2), \ldots, (\boldsymbol{x}+\boldsymbol{r}) \cdot (\boldsymbol{v}_{k-1} - \boldsymbol{v}_k), 1)$$

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{v}_1 \cdot (\boldsymbol{v}_1 - \boldsymbol{v}_2) & \cdots & \boldsymbol{v}_1 \cdot (\boldsymbol{v}_{k-1} - \boldsymbol{v}_k) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{v}_k (\boldsymbol{v}_1 - \boldsymbol{v}_2) & \cdots & \boldsymbol{v}_k (\boldsymbol{v}_{k-1} - \boldsymbol{v}_k) & 1 \end{bmatrix}$$

minimizes the loss function defined for this rule.

The computational cost for calibrating the probabilities based on Theorem 1 is $O(k^3 d)$, where k is the number of values of the RHS attribute of a rule and d is the dimensionality of embedding vectors. Typically k is small for rules, based on the desired granularity of values or value sets and ranges.

4 SELECTIVE RULE LEARNING

Now we focus on methods to judiciously and efficiently select top rules to learn. We first devise a score function that distinguishes the qualities of different rule candidates. Then we cast this problem as a *Multi-Armed Bandit* (MAB) problem [34], for which we propose a novel solution and prove the correctness and efficiency of our solution.

4.1 Score Function for Rules

A major challenge in learning the rules in Section 3 is that there are so many rule candidates, and the quality of them varies significantly—however, of course, we do not know the quality of a rule until we sufficiently learn it. Let us first define a metric to quantify the quality of a rule. We resort to concepts from information theory, namely *entropy*, *conditional entropy*, and *mutual information* [7].

Definition 1. *We define the* quality score *of a rule* $\rho: X \rightarrow_t Y$, *denoted as* $q(\rho)$, *to be* $1-\frac{H(Y|X)}{H(Y)} = \frac{I(X;Y)}{H(Y)}$, *where* $H(Y)$ *is the entropy of* Y, $H(Y|X)$ *is the entropy of* Y *conditioned on* X, *and* $I(X;Y)$ *is the mutual information between* X *and* Y.

The conditional entropy $H(Y|X)$ measures the uncertainty of the RHS attribute Y conditioned on a given value combination of the attributes X on the LHS. Intuitively, a rule that is more informative, and hence of better quality, is more certain on Y given X, which implies a smaller $H(Y|X)$ (a higher $q(\rho)$ value). However, attribute Y might just be very "certain" by itself (i.e., with a skewed distribution); thus, we need to normalize $H(Y|X)$ with the entropy of Y itself, i.e., $H(Y)$. Interestingly, it is known in information theory that $H(Y) - H(Y|X) = I(X;Y)$, which is the *mutual information* between X and Y. The intuition is that, the more mutual information there is between the LHS and the RHS, the more useful it is to know LHS for the inference of RHS. During training, we will need to repeatedly evaluate the quality of a rule. By definition,

$$H(Y \mid X) = -\sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)}$$
$$= -\sum_{x,y} p(x) p(y \mid x) \log p(y \mid x)$$

and $H(Y) = -\sum_y p(y) \log p(y)$. We will use Laplace smoothing [29] to estimate $p(x)$ and $p(y)$ based on incremental frequency (similar to RuleTrainingOnce in Section 3.1), and use Theorem 1 (probability calibration) to estimate $p(y \mid x)$. The range of a quality score according to our definition is always in $[0, 1]$, with a higher value indicating a better quality.

4.2 Selecting Rules to Train

4.2.1 Intuition and Approach. We consider the training/learning of a rule candidate as an accumulative process, and divide it into *epochs*, each of which is shown in the RuleTrainingOnce algorithm. Assume that a full training of a rule on average requires β epochs, where β is a hyperparameter. Our goal is to efficiently retrieve the top quality rules, as measured by Definition 1, without fully exhausting all β epochs of every rule candidate. We treat this as a *Multi-Armed Bandit* (MAB) problem [34] where each rule candidate is an arm. Indeed, there is clearly an exploitation vs. exploration tradeoff—one can either keep exploiting a promising rule candidate to fully train it or try more training on other rule candidates to discover even better rules.

We model the *state* of a rule candidate ρ at epoch t in its training process as a triple (q_t, Δ_t, q_u), where q_t is the current quality $q(\rho)$ at time t, Δ_t is the current gradient of the change of $q(\rho)$, and q_u is the (optimistic) upper bound of $q(\rho)$ following the current Δ_t at the end of β epochs, i.e., $q_u = q_t + \Delta_t(\beta - t)$. We use the Thompson sampling method [30] to solve this MAB problem. For each arm (i.e., rule candidate), in general, we set the reward distribution of the Thompson sampling to be a uniform distribution in $[q_t, q_u]$. The intuition is that setting the reward to a sample value in the range of the final quality scores will make us focus on the top quality rules.

However, one problem is that there is a large number of candidate rules, and constantly evaluating a rule's quality range and sampling from it for each selection of arm is too computationally expensive. Instead, we devise a very efficient sampling scheme to accomplish the Thompson sampling.

4.2.2 Selective Learning Algorithm. Our key idea to improve the efficiency of the Thompson sampling, i.e., sampling from $[q_t, q_u]$ of each rule candidate, is to *only sample the score range of one rule candidate* in each training epoch rule-selection round, instead of sampling the score ranges of all rules in each round. For other rule candidates, we simply re-use their sample values from the previous rounds.

Clearly, this will improve the speed dramatically. A natural question is: How does it affect correctness? In other words, are we still following Thompson sampling for the MAB problem? We show in Section 4.3 that our iterative algorithm follows a Markov chain that converges to a stationary distribution that is optimal (i.e., the same as Thompson sampling). Moreover, we prove that the convergence speed (called the *mixing time*) is very fast. Thus, our sampling is very efficient and effective.

The overall rule candidate selection algorithm for iterative and incremental learning is presented in SelectiveRuleLearning. In line 1, we initialize a max-heap (priority queue) $\mathcal{H}$, which will be used to maintain and select a rule with the maximum quality score sample to perform an epoch of training in each round. Lines 2-9 perform initial training of each rule candidate for two epochs to get its initial state (q_t, Δ_t, q_u). We then get an initial quality score sample for each rule in lines 5 and 8 and place them in $\mathcal{H}$. Note that lines 2-9 are only performed once over an initial portion of the graph stream to initially push the rules into the heap $\mathcal{H}$. By contrast, the loop starting from line 10 is continuously performed with the incoming graph stream, so that the top rules are adaptive to the dynamic graph stream.

Note that here, as well as in the main body of iterative sampling in lines 13-16, the score ranges where a sample is drawn from are slightly different between the top-k rules (based on the current score q_t) and those outside top-k, where k is a rough estimate of the number of top rules that will be retrieved. There is a slight disadvantage for the current top-k rules, as the lower bound of the sampling range is 0. The intuition is that we should give higher priority to rules outside top-k so that we are not stuck at repeatedly selecting the current top-k and miss the potentially better ones simply because they are not trained enough yet. Indeed, if a high-quality rule temporarily falls out of top-k, it can be picked up again

Algorithm 2: SelectiveRuleLearning ($\mathcal{G}, \mathcal{R}, k$)

Input: $\mathcal{G}$: graph stream;
$\mathcal{R}$: set of rule candidates;
k: number of top rules needed
Output: a sequential list of top rules upon request

1 initialize a max-heap $\mathcal{H}$
2 **for** *rule* $\rho \in \mathcal{R}$ **do**
3 invoke RuleTrainingOnce ($\mathcal{G}, \rho$) twice to get its state (q_t, Δ_t, q_u)
4 **if** q_t *is among top* k *in* $\mathcal{R}$ **then**
5 $w \leftarrow$ pick value in $[0, q_u]$ uniformly at random
6 add ρ to $\mathcal{H}$ with weight w
7 **else if** $q_u >$ *k-th ranked* q_t *in* $\mathcal{R}$ **then**
8 $w \leftarrow$ pick value in $[q_t, q_u]$ uniformly at random
9 add ρ to $\mathcal{H}$ with weight w
10 **while** *true* **do**
11 $\rho \leftarrow$ random node $\in \mathcal{H}$
12 re-evaluate ρ's state (q_t, Δ_t, q_u)
13 **if** q_t *is among top* k *in* $\mathcal{R}$ **then**
14 $\rho.w \leftarrow$ pick value in $[0, q_u]$ uniformly at random
15 **else**
16 $\rho.w \leftarrow$ pick value in $[q_t, q_u]$ uniformly at random
17 $r \leftarrow$ root node of $\mathcal{H}$
18 **if** $\rho.w > r.w$ `//change ρ to be root`
19 **then**
20 $P \leftarrow$ list of nodes on the path from r to ρ in $\mathcal{H}$
21 circularly move each node in P to the one below
22 root of $\mathcal{H} \leftarrow \rho$
23 RuleTrainingOnce ($\mathcal{G}$, root of $\mathcal{H}$)
24 **if** *request of top rules* **then**
25 **return** rules from $\mathcal{H}$ in descending order of q_t

and go back to top-k. We call this process "chasing top-k". When the process stabilizes, so will the top-k membership.

Each loop starting from line 10 is a round of sampling and training a selected rule for one epoch. In line 11, we select a rule from $\mathcal{H}$ uniformly at random. We re-evaluate the selected rule's state because its embedding (based on attribute-value nodes, relationships, and attentions) might have changed from the last time its state is evaluated. In lines 17-22, we compare the new score sample of this node ρ with the score sample of the current root node of $\mathcal{H}$ (which has the previously highest score sample). If ρ's score sample turns out to be higher, it will replace the current root node in lines 18-22. Finally, the rule at the root node of $\mathcal{H}$ is selected to be trained for one epoch in line 23. At any time, upon request, lines 24-25 return rules in descending order of their quality scores q_t. In particular, the rules will be retrieved in Section B.1 to form a DBN.

4.3 Analysis of Correctness and Efficiency

Let us now analyze the behavior of SelectiveRuleLearning. We show that its state (i.e., which rules get selected for training for an epoch) follows a Markov chain, which converges to a stationary distribution that is optimal and the same as our intention of Thompson sampling for the Multi-Armed Bandit. Furthermore, we prove that this Markov chain has a nice property of converging very fast. Note that, in practice, most Markov chains do not have provable bounds of convergence time, even if they converge (i.e., they can be converging very slowly).

Theorem 2. *Given a set of n rules in the heap $\mathcal{H}$, let the state of a continuous execution of SelectiveRuleLearning be the ID of the rule that is at the root of $\mathcal{H}$. Then the execution states follow a Markov chain that has a unique stationary distribution $\boldsymbol{\pi}^\star = (\pi_1^\star, \pi_2^\star, \ldots, \pi_n^\star)$, where $\pi_i^\star$ $(1 \le i \le n)$ is the probability that rule i's quality score sample ranks the highest.*

Our analysis in Theorem 2 establishes the correctness of SelectiveRuleLearning, as it is Thompson sampling for MAB. We now show that this execution-state Markov chain converges very fast, and therefore it is efficient and effective.

Theorem 3. *In the Markov chain stated in Theorem 2, starting from any initial state, the variation distance between the distribution of the state of the chain after $n \ln \frac{n}{\varepsilon}$ steps and the stationary distribution is at most ε, where n is the number of rules in $\mathcal{R}$.*

Now that we have shown the correctness and sampling efficiency, we further point out that, due to the fact that we perform efficient graph neural embedding, SelectiveRuleLearning can be run continuously with the data stream. Thus, the results, specifically the top rules, are adaptive to the graph stream $\mathcal{G}$.

5 EXPERIMENTS

We use the following real-world datasets in two different domains, namely medicine and human mobility and social communication: (1) MIMIC data (Medical Information Mart for Intensive Care III [14]), and (2) FF data (the Friends and Family dataset [1]). We compare against the following baselines: (1) Multivariate LSTM model [11], (2) State-of-the-art temporal knowledge graph neural network model RE-Net [13], and (3) A variant of our method, where all the rule candidates are selected uniformly for training. We describe the details of the datasets, baselines, and machine setup in Appendix C.

5.1 Experimental Results

For the MIMIC data, we treat the *diagnoses* and *procedures* of each patient as static relationships, and timestamped data such as *input events* (fluids administered to the patient), *lab events* (lab measurements), and *prescriptions* as dynamic graph streams. The attributes in the conditional distribution rules are associated with the heterogeneous events, outcomes, and prescription drugs.

For the FF data, we treat relationships among subjects such as the *couple* (family) relations and the *friends* relations as static relationships, and dynamic information and interactions such as *accelerometer readings* (sampled up to 4 times per second), *proximity of others* by scanning Bluetooth devices, *location updates* and *speed estimates*, *phone calls*, and *text messages* as local and graph streams. Some attributes are derived from an aggregate of the original raw tuples, such as the "surrounding crowd size", which is the number of distinct people (scanned Bluetooth devices) within a period of time.

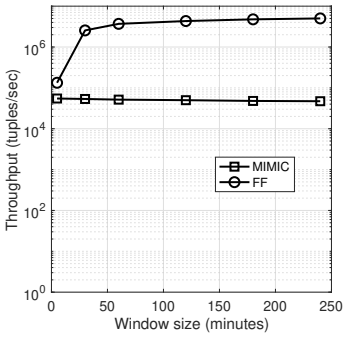

Fig 3 Getting rule candidates

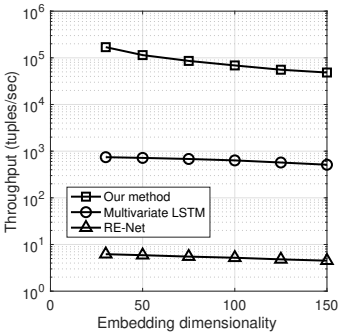

Fig 4 Training throughput (MIMIC)

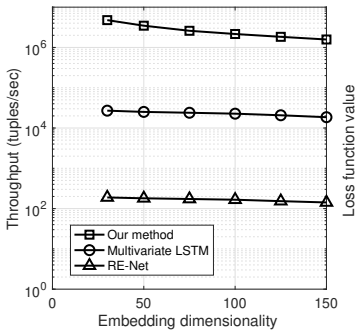

Fig 5 Training throughput (FF)

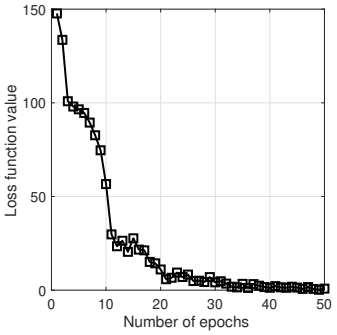

Fig 6 Loss function (MIMIC)

5.1.1 Representation Learning and Rule Learning. In the first set of experiments, we examine the step of obtaining a set of TCD rule candidates from the heterogeneous graph streams, which will be subsequently used for selection of top rules. A parameter here is the size of a sliding window within which we consider as the "same time unit" dependency $\rightarrow_0$ (dependencies with the next and previous windows are $\rightarrow_+$ and $\rightarrow_-$, respectively). We vary this sliding window size and measure the stream system throughput when continuously performing this step. The results are shown in Figure 3 for both the MIMIC and the FF datasets.

The throughput is measured in the number of *raw* stream tuples processed (among all local and graph streams) per second. We can see that, while the throughputs for both datasets are high (over fifty thousand tuples per second), it is much faster for the FF data than the MIMIC data for the same window size that is not too small (e.g., above 5 minutes). This is because the FF data contains some very fast streams, e.g., the accelerometer readings (sampled 4 times per second), and we perform fast aggregation within the time window to get stream attribute values that are not too dense; hence the throughput is much higher with respect to the original raw tuples. Moreover, as the window size increases, the throughput with the MIMIC data slightly decreases, as a window contains more events and hence more correlations need to be examined. This is not the case with the FF data since we aggregate more raw tuples into discrete events as window size increases.

It is worth noting that the rule candidate generation step does not have to be run very often, as the set of rule candidates is relatively stable—although the conditional distribution rules themselves and which ones have top quality scores might be more dynamic and can be continuously learned.

We then proceed to our SelectiveRuleLearning algorithm for simultaneous graph neural embedding and rule training. We first measure the system throughput as the learning algorithm is continuously and adaptively executed with the incoming streams. The results are shown in Figure 4 for the MIMIC dataset and in Figure 5 for the FF dataset. One parameter here is the dimensionality of the embedding vectors, which we vary from 30 to 150 dimensions. We also compare against the training throughput of the two baseline methods using Multivariate LSTM and RE-Net, respectively.

From Figures 4 and 5, we can see that our simultaneous graph embedding and selective rule learning algorithm is very efficient, achieving throughput over 100K tuples per second and 3 million tuples per second when the dimensionality is 50 (our default) for the MIMIC and FF datasets, respectively. The higher throughput with the FF data is again due to the aggregation of raw tuples. As the embedding dimensionality increases, the throughput decreases. This is because many of the stochastic gradient descent training steps have a cost proportional to the dimensionality of the embedding vector. In subsequent experiments, we will see the tradeoff between this dimensionality cost and inference accuracy.

Furthermore, our embedding and rule learning method is about 2 and 4 orders of magnitude faster than the Multivariate LSTM model and the RE-Net model training, respectively. Those two baselines are deep and complex neural network models that are known to be computationally intensive. Especially, RE-Net consists of multiple layers of graph convolution networks and sequence models. Moreover, they do not learn the rules as we do.

5.1.2 Learning Effectiveness and Rule Quality. We then examine the validity and effectiveness of our neural embedding and selective rule learning. The first aspect of the evaluation is to look at the stochastic gradient descent training process with the attention-based aggregation, by checking the change of the loss function value discussed in Section 3.1 over the training epochs of a top rule. This is plotted in Figure 6 for the MIMIC data and Figure 7 for the FF data.

We can see that the training process is very effective. The loss function value drops precipitously over the initial epochs, and flattens after 30 to 50 epochs. In addition, it can be observed that the training is somewhat easier for the MIMIC data than for the FF data, as the loss function curve levels off and stays low after only 20 or 30 epochs. This is mainly due to the nature and predictability of the data—arguably, the medical domain is more predictable than human mobility and social communication. Interestingly, this also coincides with rule quality scores and prediction accuracy which we examine subsequently.

As another aspect of training effectiveness evaluation, we now look at how the average quality score of top 30 rules evolves over their training epochs. The results are shown in Figure 8 for the MIMIC data and in Figure 9 for the FF data. We can see that the rule quality scores increase sharply during the initial epochs. Moreover, it is easier and faster to get to a level of better quality scores for the MIMIC data than for the FF data. This is consistent with the result of loss function value change in Figures 6 and 7 as discussed above.

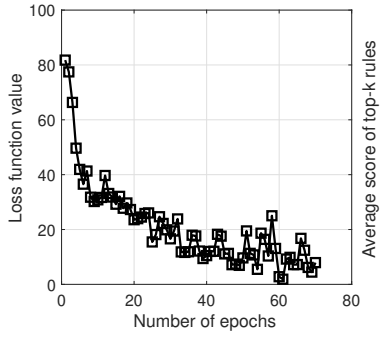

Fig 7 Loss function (FF)

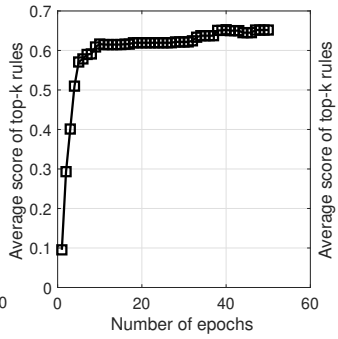

Fig 8 Rule scores (MIMIC)

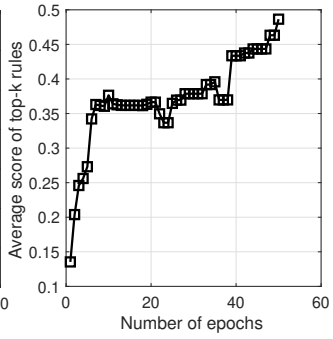

Fig 9 Rule scores (FF)

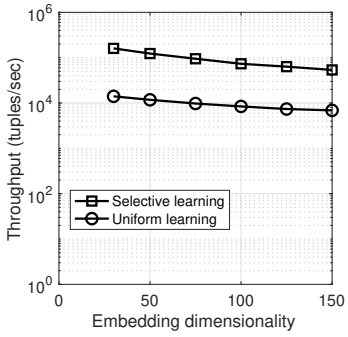

Fig 10 Throughput (MIMIC)

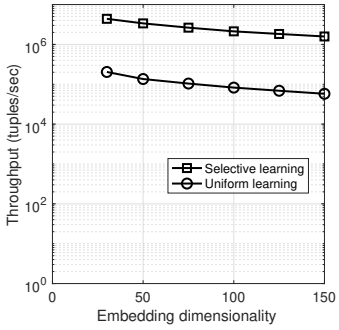

Fig 11 Throughput comparison (FF)

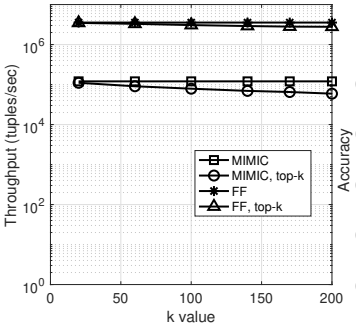

Fig 12 Retrieving DBN rules

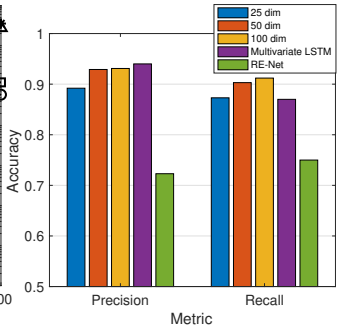

Fig 13 Accuracy (MIMIC)

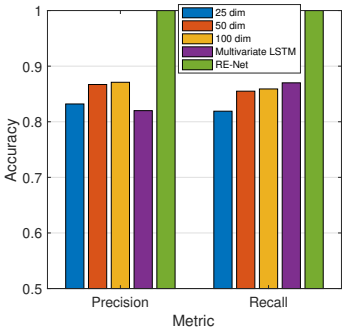

Fig 14 Accuracy (FF)

5.1.3 More Baseline Comparisons and Interpretable Inference. Recall that our SelectiveRuleLearning algorithm treats the training as a Multi-Armed Bandit problem and uses Thompson and Markov chain sampling to improve the performance. We now compare it against our second baseline method that does uniform learning over all rule candidates. The throughput results are shown in Figure 10 for the MIMIC data and in Figure 11 for the FF data. We can see that SelectiveRuleLearning has between 1 and 2 orders of magnitude higher throughput than the uniform rule learning, since it judiciously selects the rules, focusing on learning the most promising ones for top quality scores, and pruning the low quality ones early.

We now continue to iterating through the top conditional distribution rules as in BuildingDBN (use case presented in Appendix B). We first look at the impact of this to system throughput. The result is shown in Figure 12 with both datasets, for different k parameter values (the number of top rules retrieved). We can see that it only slightly decreases the throughput, even for the larger k values.

Last but not least, we proceed to the final aspect of evaluating the effectiveness of our neural embedding and rule learning by examining the accuracy of predicting missing attribute values. We reserve some tuples for testing and remove them from the training set. As typical for graph embedding, the prediction goes by selecting the attribute value that would result in a smaller loss function value compared to the alternative values.

We also compare the accuracy against the Multivariate LSTM and the RE-Net baseline methods. Of course, a major advantage of our RL^2 approach is that we have auxiliary conditional distribution rules that provide explanations of the prediction result. The results are shown in Figure 13 for the MIMIC data and in Figure 14 for the FF data. We use two standard metrics *precision* and *recall* [29] for accuracy. In addition, we examine the accuracy for three different dimensionality values of embedding vectors, 25, 50, and 100. We can see that the prediction accuracy of our method is competitive with that of the Multivariate LSTM and RE-Net. We also find that the accuracy performance of RE-Net is not very stable—it is significantly worse than Multivariate LSTM and our method for the MIMIC dataset, but significantly better for the FF dataset. This is because RE-Net is a very sophisticated model involving multiple layers of graph convolution networks and sequence models, which depends heavily on the suitability and amount of stream data used for training.

Moreover, for our method, as the dimensionality of embedding vectors increase from 25 to 50, the accuracy gain is much more significant than the increase from 50 to 100. Our experiments earlier indicate that a higher dimensionality implies a higher computational cost. Thus, there is a tradeoff. For our method, the precision and recall values are around 0.8 and 0.9. In addition, the accuracy values for the MIMIC data are slightly higher than those for the FF data. This is consistent with our observations earlier for loss function values and rule quality scores during the training.

Case Study of Using Rules to Interpret. An advantage of our approach is that we also obtain the symbolic TCD rules. We perform a case study of using the obtained TCD rules to interpret prediction results based on the connectionist networks (graph embedding).

We give some examples of the rules returned by our algorithms that provide explanations of the prediction results we get. For the MIMIC data, one prediction of the prescription Ondansetron can be explained by a conditional distribution rule we have found "Simvastatin, 0.9% Sodium Chloride → Ondansetron". Indeed, the co-use of Simvastatin and Ondansetron for certain diseases has been documented in the literature [6]. Another example is the prediction of Aspirin explained by the probabilistic rule "Pantoprazole, Ferrous Sulfate, 0.9% Sodium Chloride → Aspirin", as also verified in [35]. Yet another example is to explain the prediction result of "Phytonadione" usage by the usage of Warfarin and Lisinopril at an earlier time and the rule "Warfarin, Lisinopril $\rightarrow_+$ Phytonadione". Specifically, Warfarin and Lisinopril treat high blood pressure and prevent blood clots in veins, and doctors often use Phytonadione later to reduce the side effect of Warfarin (excessive blood thinning).

For the FF dataset, an example is the prediction result of a "long phone call" (defined as longer than two minutes) is explained by a conditional distribution rule that we have found "CallFriend → LongCall". As another example, the prediction result of "surrounding crowd size of 2 to 3 people" is probabilistically explained by a rule that we find from the data "if someone is next to a friend, with higher probability the surrounding crowd size is medium (no more than 3) than large (4 or above)". These distribution rules make intuitive sense, at least based on the data from which they are discovered. By contrast, the prediction results using a model like Multivariate LSTM or RE-Net lack such interpretability power due to the deep hidden layers.

6 CONCLUSIONS

In this paper, we propose an approach RL^2 to provide synergistic symbiosis of representations and rules in graph streams. We are the first to demonstrate the feasibility of simultaneous representation learning and rule learning efficiently and effectively out of the same online training process. We devise neural embedding and rule learning with attention-based aggregation, probability calibration, and a novel Markov chain sampling based Multi-Armed Bandit solution. Our experimental evaluation shows its efficiency, effectiveness, and superiority over baselines.

Acknowledgments. This work is supported by NSF grants IIS-1633271, IIS-2124704, OAC-2106740, and New England Transportation Consortium project 20-2.

REFERENCES

[1] Nadav Aharony, Wei Pan, Cory Ip, Inas Khayal, and Alex Pentland. 2011. SocialfMRI: Investigating and shaping social mechanisms in the real world. *Pervasive and Mobile Computing* 7 (2011), 643–659.

[2] Kasun Bandara, Peibei Shi, Christoph Bergmeir, Hansika Hewamalage, Quoc Tran, and Brian Seaman. 2019. Sales Demand Forecast in E-commerce Using a Long Short-Term Memory Neural Network Methodology. In *International Conference on Neural Information Processing*.

[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc.

[4] Léon Bottou. 2004. Stochastic Learning. In *Advanced Lectures on Machine Learning: ML Summer Schools*. Springer Berlin Heidelberg, 146–168.

[5] HongYun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. 2017. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. *CoRR* abs/1709.07604 (2017). http://arxiv.org/abs/1709.07604

[6] I. B. Chaudhry, N. Husain, M. O. Husain, J. Hallak, R. Drake, A. Kazmi, R. u. Rahman, M. M. Hamirani, T. Kiran, N. Mehmood, J. Stirling, G. Dunn, and B. Deakin. 2013. Ondansetron and simvastatin added to treatment as usual in patients with schizophrenia: study protocol for a randomized controlled trial. *Trials* 14 (2013).

[7] Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory, 2nd Edition*. Wiley.

[8] Dataset. 2022. Available at http://realitycommons.media.mit.edu/friendsdataset2.html.

[9] Josef Fagerstrom, Magnus Bang, Daniel Wilhelms, and Michelle S. Chew. 2019. LiSep LSTM: A Machine Learning Algorithm for Early Detection of Septic Shock. *Scientific Reports, Nature Research* 9, 15132 (2019).

[10] Ashok K. Goel. 2021. Looking Back, Looking Ahead: Symbolic versus Connectionist AI. *AI Magazine* 42 (2021), 83–85.

[11] Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[12] Yadigar N. Imamverdiyev and Fargana J. Abdullayeva. 2020. Condition Monitoring of Equipment in Oil Wells using Deep Learning. *Advances in Data Science and Adaptive Analysis* 12, 1 (2020).

[13] Woojeong Jin, Meng Qu, Xisen Jin, and Xiang Ren. 2020. Recurrent Event Network: Autoregressive Structure Inference over Temporal Knowledge Graphs. In *EMNLP*.

[14] AEW Johnson, TJ Pollard, L Shen, L Lehman, M Feng, M Ghassemi, B Moody, P Szolovits, LA Celi, and RG Mark. 2016. MIMIC-III, a freely accessible critical care database. *Scientific Data* (2016).

[15] D. Kingma and J. Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*.

[16] Will Knight. 2019. Two rival AI approaches combine to let machines learn about the world like a child. *MIT Technology Review* (2019).

[17] Ninghao Liu, Mengnan Du, and Xia Hu. 2019. Representation Interpretation with Spatial Encoding and Multimodal Analytics. In *WSDM*.

[18] Ninghao Liu, Xiao Huang, Jundong Li, and Xia Hu. 2018. On Interpretation of Network Embedding via Taxonomy Induction. In *KDD*.

[19] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. 2019. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *ICLR*.

[20] Andrew McGregor. 2014. Graph stream algorithms: A survey. *ACM SIGMOD Record* 43, 1 (2014), 9–20.

[21] S.P. Meyn and R.L. Tweedie. 2012. *Markov Chains and Stochastic Stability*. Springer London. https://books.google.com/books?id=LlTlBwAAQBAJ

[22] Marvin Minsky. 1991. Logical vs. analogical or symbolic vs. connectionist or neat vs. scruffy. *AI Magazine* 12, 2 (1991), 34–51.

[23] M. Mitzenmacher and E.Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

[24] Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting Good Probabilities With Supervised Learning. In *Proceedings of the 22nd International Conference on Machine Learning*.

[25] Subendhu Rongali, Adam J Rose, David D McManus, Adarsha S Bajracharya, Alok Kapoor, Edgard Granillo, and Hong Yu. 2020. Learning Latent Space Representations to Predict Patient Outcomes: Model Development and Validation. *Journal of Medical Internet Research* 22, 3 (2020).

[26] Tara Safavi, Danai Koutra, and Edgar Meij. 2020. Evaluating the Calibration of Knowledge Graph Embeddings for Trustworthy Link Prediction. In *The 2020 Conference on Empirical Methods in Natural Language Processing*.

[27] Source code. 2022. Available at https://drive.google.com/file/d/1l3aDHi620yDKgOGKxalZyEqP-MqXFcbC/view?usp=sharing.

[28] Pedro Tabacof and Luca Costabello. 2020. Probability Calibration FOR KNOWLEDGE GRAPH EMBEDDING MODELS. In *The International Conference on Learning Representations (ICLR)*.

[29] Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar. 2018. *Introduction to Data Mining* (2nd ed.). Pearson.

[30] W. Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 3/4 (1933), 285–294.

[31] I. B. Vapnyarskii. 2010. *Lagrange multipliers*. Springer.

[32] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph Attention Networks. *ICLR* (2018).

[33] Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. 2019. Explainable Reasoning over Knowledge Graphs for Recommendation. In *AAAI*.

[34] R. Weber. 1992. On the Gittins index for multi-armed bandits. *Annals of Applied Probability* (1992).

[35] P. Wei, Y. G. Zhang, L. Ling, Z. Q. Tao, L. Y. Ji, J. Bai, B. Zong, C. Y. Jiang, Q. Zhang, Q. Fu, and X. J. Yang. 2016. Effects of the short-term application of pantoprazole combined with aspirin and clopidogrel in the treatment of acute STEMI. *Experimental and therapeutic medicine* 12 (2016), 2861–2864.

[36] J. Xu, R. Rahmatizadeh, L. Boloni, and D. Turgut. 2018. Real-Time Prediction of Taxi Demand Using Recurrent Neural Networks. *IEEE Transactions on Intelligent Transportation Systems* 19, 8 (2018), 2572–2581.

A PROOFS OF THEOREMS

A.1 Proof of Theorem 1

Proof. For brevity, let $\boldsymbol{h} = \boldsymbol{x} + \boldsymbol{r}$. Then the loss function for the training of this rule is $\ell = \sum_{i=1}^{d}(h_i - p_1 v_{1i} - \ldots - p_k v_{ki})^2$, where we use L2 loss function, v_{ji} is the i'th dimension of $\boldsymbol{v_j}$, and d is the dimensionality of embedding vectors. We also have the constraint $p_1 + \ldots + p_k = 1$. Applying Lagrange multipliers [31], we get the Lagrangian function

$$\mathcal{L}(p_1, \ldots, p_k, \lambda) = \sum_{i=1}^{d}(h_i - \sum_{j=1}^{k} p_j v_{ji})^2 - \lambda(\sum_{i=1}^{k} p_i - 1)$$

Setting the derivatives to 0, we get

$$\frac{\partial \mathcal{L}}{\partial p_1} = -2\sum_{i=1}^{d}[(h_i - p_1 v_{1i} - \ldots - p_k v_{ki})v_{1i}] - \lambda = 0$$

$$\vdots$$

$$\frac{\partial \mathcal{L}}{\partial p_k} = -2\sum_{i=1}^{d}[(h_i - p_1 v_{1i} - \ldots - p_k v_{ki})v_{ki}] - \lambda = 0$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 1 - p_1 - \ldots - p_k = 0$$

Subtracting the respective succeeding equation from each of the first $k-1$ equations above, we get a new system of equations

$$p_1 \boldsymbol{v_1} \cdot (\boldsymbol{v_i} - \boldsymbol{v_{i+1}}) + \ldots + p_k \boldsymbol{v_k} \cdot (\boldsymbol{v_i} - \boldsymbol{v_{i+1}}) = \boldsymbol{h} \cdot (\boldsymbol{v_i} - \boldsymbol{v_{i+1}})$$

for $1 \le i \le k-1$, which, along with $p_1 + \ldots + p_k = 1$, can be rewritten as $(p_1, \ldots, p_k) \cdot \boldsymbol{A} = \boldsymbol{b}$, where $\boldsymbol{A}$ and $\boldsymbol{b}$ are in the form as stated in the theorem. This concludes the proof. □

A.2 Proof of Theorem 2

Proof. We first show that the execution follows a Markov chain, where the state is the ID of the rule that is at the root of the max heap $\mathcal{H}$. We call each iteration of the loop at line 10 a *step*. In step t, let the rule at the root of $\mathcal{H}$ be r_i ($1 \le i \le n$), i.e., the Markov chain is at state i. Let $\sigma_{i,t}$ denote the sample of rule r_i in step t. Then the probability that the chain will transition to state j ($1 \le j \le n$ and $j \ne i$) in step $t+1$, i.e., the root of $\mathcal{H}$ will be r_j, is

$$P_{ij} = Pr[r_j\ chosen] \cdot p_1 + Pr[r_i\ chosen] \cdot p_2 \tag{2}$$

where $Pr[r_j\ chosen] = Pr[r_i\ chosen] = \frac{1}{n}$ is the probability that r_j (resp. r_i) is chosen in line 11, and

$$p_1 = Pr[\sigma_{j,t+1} > \sigma_{i,t}] \mid \sigma_{i,t}\ largest] \tag{3}$$

$$p_2 = Pr[\sigma_{j,t}\ 2_{nd}\ largest, \sigma_{i,t+1} < \sigma_{j,t} \mid \sigma_{i,t}\ largest] \tag{4}$$

Moreover, $P_{ii} = 1 - \sum_{j \ne i} P_{ij}$. Given the uniform distributions of each rule, the probabilities p_1, p_2, and hence P_{ij} are well defined. Thus, the execution follows a Markov chain. We further observe that this chain is finite, *irreducible*, and *aperiodic*, and thus it is an *ergodic* chain [21]. Specifically, there is a finite number of states, and from any state, there is a positive probability to reach any other state—making it irreducible. The chain is also aperiodic as from any state, there is also a non-zero probability to stay in that state in the next step (i.e., the rule at the root is not replaced). It then follows that this chain has a unique stationary distribution [21], which is the sought-after $\boldsymbol{\pi^\star}$ stated in the theorem as it satisfies $\boldsymbol{\pi^\star}\mathbf{P} = \boldsymbol{\pi^\star}$, where $\mathbf{P}$ is the transition matrix based on the P_{ij}'s above. □

A.3 Proof of Theorem 3

Proof. We use the Markov chain *coupling* technique [23] to prove the convergence speed. We run two copies X_t and Y_t of the Markov chain in Theorem 2 at the same time, but they may have different initial states, since the weight w is chosen randomly from the range. We let the first chain X_t run as usual, and let the second chain Y_t make exactly the same random choices as X_t in each *step* (a step is one iteration of the loop from line 10)—i.e., the same rule from $\mathcal{H}$ is chosen and the same random value w is picked from its distribution. This coupling is valid because Y_t also follows the same transition probabilities $\mathbf{P}$. Hence, the two copies X_t and Y_t will surely become *coupled* when every rule is chosen at least once in line 11, i.e., X_t and Y_t will always be in the same states after that.

To analyze the random process above, after X_t and Y_t run $n \ln n + cn$ steps (where c is a constant), the probability that a specific rule has not been selected in line 11 at least once is at most

$$\left(1 - \frac{1}{n}\right)^{n \ln n + cn} \le e^{-(\ln n + c)} = \frac{e^{-c}}{n} \tag{5}$$

Then by the union bound, the probability that *any* rule has not been selected at least once is at most e^{-c}. Set $c = \ln \frac{1}{\varepsilon}$. Thus, after X_t and Y_t run $n \ln n + n \ln \frac{1}{\varepsilon} = n \ln \frac{n}{\varepsilon}$ steps, the probability that the two chains have not coupled is at most $e^{-c} = e^{-\ln \frac{1}{\varepsilon}} = \varepsilon$. From the Coupling Lemma [23], we obtain the result. □

B ADDITIONAL USE CASE OF TCD RULES

B.1 Retrieving Rules to Build DBN

While we are picking the top quality-score rules based on their q_t values, we must avoid adding a rule that could create a directed cycle, which is a requirement of the DBN for inference. Hence, we cannot add a rule into our DBN that entails an edge from attribute x to attribute y, if there is already a directed path from y to x, as that would form a cycle. Thus, we maintain a reachability matrix P, a two-dimensional bit array as illustrated in Figure B.1(b), to keep track of reachability—whether there is a directed path from an attribute to another, subject to the transitions of temporal states due to the DBN rules as shown in Figure B.1(a). The matrix P is $2\alpha \times 2\alpha$, where α is the number of attributes that may appear in all the rules. The first α rows (resp. columns) are called the t_0 rows (resp. columns), while the second α rows (resp. columns) are called the t_+ rows (resp. columns). t_0 and t_+ represent two successive temporal states, i.e., either *present* and *future*, respectively, in Figure B.1(a), or *past* and *present*, respectively.

Thus, P is divided into 4 equal-sized blocks, separated by the two dashed lines in Figure B.1(b), which are called the (t_0, t_0) block, and so on. Intuitively, before adding a rule that would create a $x \rightarrow_+ y$ (resp. $x \rightarrow_- y$) edge, we check the corresponding bit corresponding to (y, x) in the (t_+, t_0) block (i.e., bottom-left) (resp. the (t_0, t_+) block). If it is set, adding the rule would create a cycle. If it is a $x \rightarrow_0 y$ edge, we will check both (t_0, t_0) and (t_+, t_+) blocks. The precise meaning of each block will be made clear shortly. We present the algorithm in BuildingDBN, and analyze its correctness.

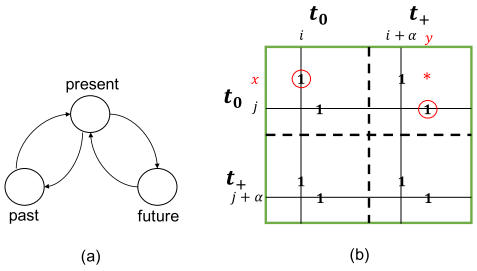

Figure B.1: (a) Capturing the transition among three time states when considering acyclic DBN. (b) A reachability binary matrix with paths of temporal transitions indicated in (a). The matrix has 4 equal-sized sub-blocks (separated by dashed lines).

Algorithm 3: BUILDINGDBN ($\mathcal{H}$)

Input: $\mathcal{H}$: heap from SELECTIVERULELEARNING to retrieve rules;
Output: a DBN

```
1  R ← ∅ //initialize the set of rules to be returned as DBN
2  initialize a 2α × 2α bit array P //α is number of attributes
3  while stopping_condition(R, H) is not met do
4      pop a rule X →_t Y from H
5      for attribute x ∈ X do
6          let attribute x and attribute of Y be the i-th and j-th
7          if t = + then
8              i' ← i; j' ← j + α
9          else if t = − then
10             i' ← i + α; j' ← j
11         else
12             i' ← i + α; j' ← j + α
13         if P[j', i'] = 1 or (t = 0 and P[j, i] = 1) then
14             rollback any changes to P and continue to next
               rule
15         P[i', j'] ← 1
16         ShortCircuit (P, i', j')
17         if t = 0 then
18             P[i, j] ← 1
19             ShortCircuit (P, i, j)
20     R ← R ∪ {X →_t Y}
21 return R
22 Function ShortCircuit (P, i, j)
23     for cell (x, i) set in column i of P do
24         for cell (j, y) set in row j of P do
25             P[x, y] ← 1
```

THEOREM 4. *Algorithm* BUILDINGDBN *maintains the following invariants. (1) A bit $P[i, j]$ in the (t_0, t_+) block (resp. (t_+, t_0) block) is 1 if and only if there is a directed path from the i-th attribute to the j-th attribute in the succeeding (resp. preceding) temporal state. (2) A bit $P[i, j]$ in the (t_0, t_0) block (resp. (t_+, t_+) block) is 1 if and only if there is a directed path from the i-th attribute to the j-th attribute in the same temporal state and the first cross-time edge (if any) must be to the succeeding (resp. preceding) temporal state.*

PROOF. We prove by induction on each new edge. Initially the two invariants must be true as all bits of P are 0 and there are no directed paths. For the induction, suppose we are to add a new edge from the i-th to j-th attribute. If it is r_+, then the (i', j') entry in line 8 corresponds to a bit in the (t_0, t_+) block of P to be set, as $0 \le i' < \alpha$ and $\alpha \le j' < 2\alpha$. Line 13, according to the induction hypothesis, is equivalent to checking whether there is a directed path from the j-th to i-th attribute in the preceding temporal state. If true, we would not add the new edge which would form a cycle. Otherwise we add it. SHORTCIRCUIT in line 16 ensures that, for each pair of directed paths from x to i' and from j' to y, there must be a directed path from x to y due to the new edge. For the induction step, we can verify that both invariants are still true after setting these bits, in all the four cases of (x, y) based on which of the four blocks of P (x, y) is in, as well as which block (i, j) is in due to the t value in the rule. □

C DETAILS OF EXPERIMENT DATASETS, BASELINES, AND SETUP

(1) MIMIC data. The MIMIC-III (Medical Information Mart for Intensive Care III) dataset [14] is a large, freely-available database comprising deidentified health-related data associated with over forty thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012. It includes high temporal resolution data such as vital sign measurements, laboratory test results, procedures, medications, caregiver notes, imaging reports, demographics, and others. The total size is 27GB.
(2) FF data. The Friends and Family (FF) dataset [1] is from an experimental study of how people make decisions with emphasis on the social aspects involved, and how to empower people to make better decisions using personal and social tools. The subjects were members of a young-family residential community adjacent to a major university in North America. Their behavior and interactions were closely tracked with their personal Android-based mobile phones, with signals including accelerometer reading, apps running, bluetooth devices nearby, battery information, call log, GPS, WiFi access points nearby and SMS log. Their behavior and interactions were also tracked with surveys. The size is 4.6GB.

We have implemented the algorithms in Java. We have also implemented the following baseline algorithms: (1) The Multivariate LSTM model [11] with a flat relational tuple at each time step, in Python 3.6.9, Keras version 2.3.1 with Tensorflow version 1.14.0 backend. The Keras LSTM has a learning rate of 0.001, using the Adam optimizer [15] with the mean absolute error (MAE) loss function. LSTM was shown to be the state-of-the-art model for MIMIC-III data [25]. LSTM was also shown to give the state-of-the-art prediction results for multi-relational sequence data in different domains in recent studies such as [2, 9, 12, 36]. (2) State-of-the-art temporal knowledge graph neural network model RE-Net [13] based on the source-code provided by the authors. (3) A variant of our method, where all the rule candidates are selected uniformly for training. The experiments are performed on a MacBook Pro machine with OS X version 10.11.4, a 2.5 GHz Intel Core i7 processor, a 16 GB 1600 MHz DDR3 memory, and a Macintosh hard disk. The RE-Net baseline is run under Python 3 with Pytorch 1.6, CUDA 10.1, an Nvidia GTX1080Ti GPU (11GB GPU memory).