Knowledge Graphs in AI

Volker Tresp

Winter 2019

Machine Learning, Relational Databases and First-Order Logic

Introduction

- Most of deep learning is concerned with the case where a row in a data matrix corresponds to a data point and a column to an attributes;

$$x_{column=Jack,row=Male} = 1$$

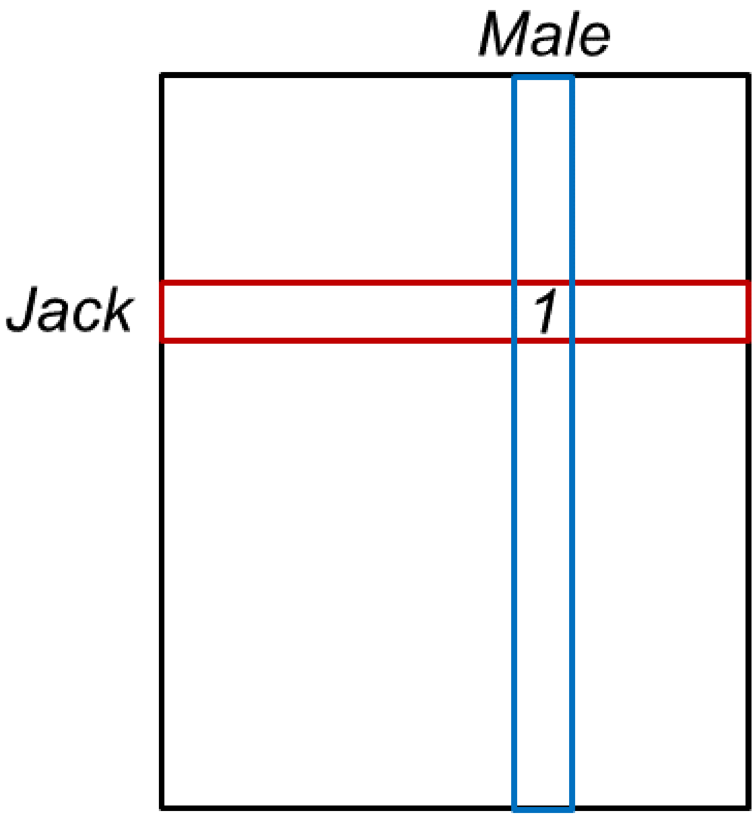
  means that Jack has the attribute male

- Social network analysis: But consider the case of individuals which might or might not know each other and where income can be predicted from the income of the people an individual knows

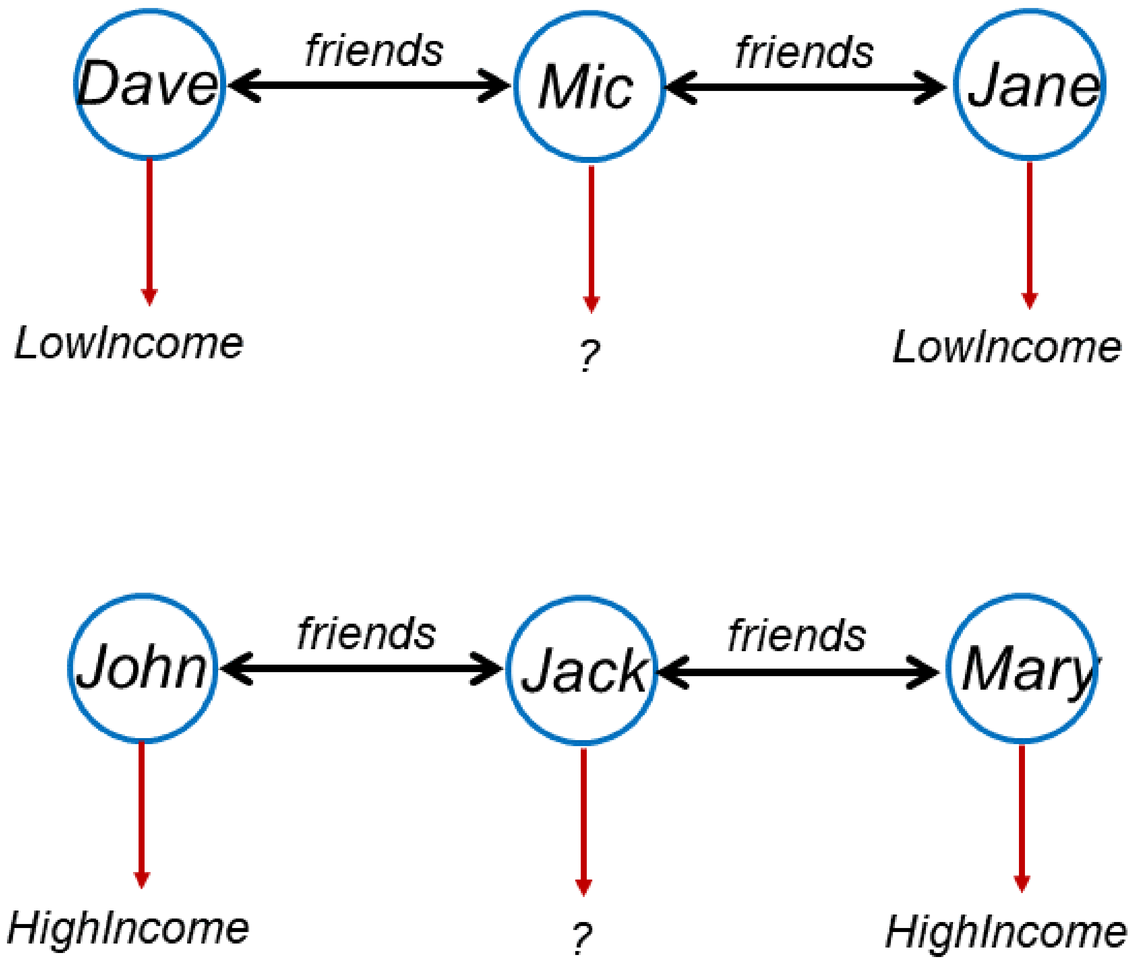
- Recommendation systems: Or consider a rating matrix of users (rows) and movies (columns); without knowing any attributes from the users and the movies, recommendation engines are able to predict unknown ratings for user/movie pairs

- In the latter two examples it is not obvious how a the data can be represented truthfully in a simple table of data points

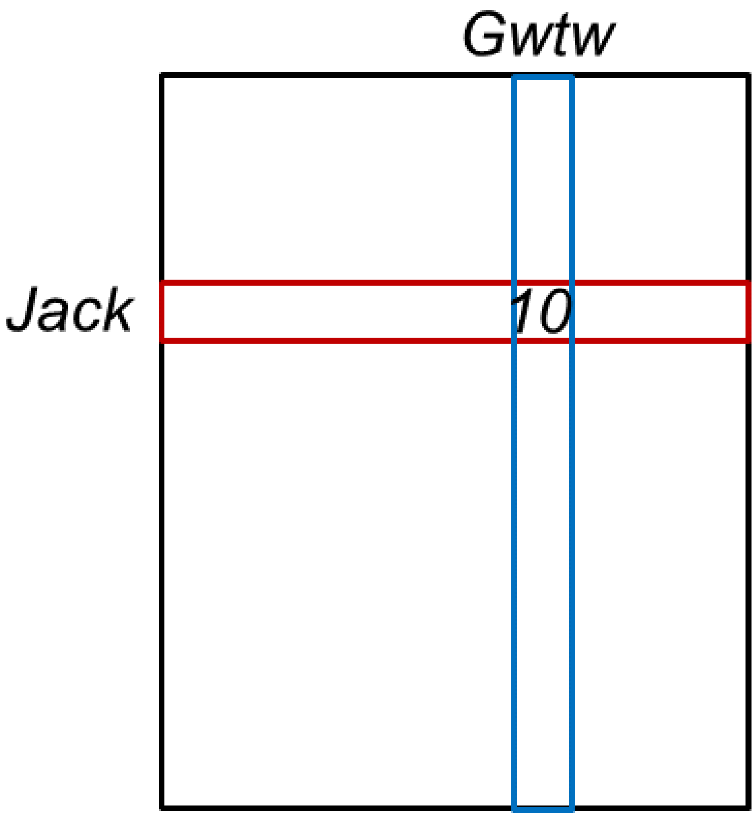
A Data Matrix in Machine Learning



A social network: homophily in friendship relations?



User-Movie-Ratings



A Standard Machine Learning Setting

- Consider a trained classifier for the evaluation of job applicants

$$P(\textit{highPotential}|\textit{hsGradeHigh}, \textit{parentIncHigh}) =$$

$$f_{\mathbf{w}}(\textit{hsGradeHigh}, \textit{parentIncHigh})$$

- It reads: The probability that an applicant has high potential can be predicted from on the applicant's high school grade and the income of the parents of the applicant
- Here, $f_{\mathbf{w}}(.)$ is a trained classifier, e.g., a neural network

Database

- The training data might have been available in a *database*
- The table (i.e., relation) *candidate(cID)* contains the IDs of all candidates
- Similarly, we define the relations *hsGradeHigh(cID)*, *parentIncHigh(cID)*, and *highPotential(cID)*
- Note that each random variable in the classifier corresponds to a unary relation (a relation with one attribute) in the database: Consider the data point $cID = Jack$. If *Jack* is element of the relation *hsGradeHigh*, then the corresponding input to the classifier is 1, otherwise 0

relational database

candidates	hsGradeHigh	parentIncHigh	highPotentials
Jack	Jack	Jack	Jack
John		John	
Mary			Mary
Lisa	Lisa		

data matrix for machine learning

	candidate	hsGradeHigh	parentIncHigh	HighPotential
Jack	1	1	1	1
John	1	0	1	0
Mary	1	0	0	1
Lisa	1	1	0	0

Deductive Database

- One might postulate the rule

$$\forall x.\ \textit{highPotential}(x) \leftarrow \textit{parentIncHigh}(x) \wedge \textit{hsGradeHigh}(x)$$

- Here x is a variable (that stands for objects; not to be confused with a random variable), $\forall$ is a universal quantifier, $\wedge$ is logical connective
- This rule says if we have a candidate whose high school grade is high and whose parents have high income, then we can conclude that this is a candidate with high potential
- The relation names now stand for *predicates*, which are functions that return true or false

Deductive/probabilistic/inductive database

candidates	hsGradeHigh	parentIncHigh	highPotentials
Jack	Jack	Jack	Jack
John		John	
Mary			Mary
Lisa	Lisa		

Deductive Database (cont'd)

- Such a rule might be part of a datalog rule base; datalog is a declarative logic programming language that syntactically is a subset of prolog. It is often used as a query language for deductive databases
- We could invoke the rule at query time: when we ask for students with high potential, we respond with the actual *cIDs* of the students in the relation *highPotential* plus the ones we infer to be high potentials by invoking the rule
- Alternatively, we can materialize (deductive closure) and enter candidates which are both in relation *parentIncHigh* and *hsGradeHigh* to the relation *highPotential*

Probabilistic Database

- The rule becomes probabilistic

$$\forall x.$$

$$P(\textit{highPotential}(x) = 1 | \textit{hsGradeHigh}(x) = 1, \textit{parentIncHigh}(x) = 1) = P_{t|t,t}$$

- Here, $0 \leq P_{t|t,t} \leq 1$
- The statement should read: consider only candidates with *hsGradeHigh* $= 1$ and *parentIncHigh* $= 1$. If we randomly pick a candidate out of that set, the probability that it is a candidate with high potential is $P_{t|t,t}$

Probabilistic Database (cont'd)

- To be complete, we need in total 4 probabilities: $P_{t|t,t}$, $P_{t|t,f}$, $P_{t|f,t}$, $P_{t|f,f}$
- In (Relational) Bayesian Networks, it is assumed that these 4 numbers are given by an expert

Inductive Database

- An inductive database is the same as a probabilistic database, only that the probabilities are learned from data,

$$\forall x.\ P(\mathit{highPotential}(x)|\mathit{hsGradeHigh}(x), \mathit{parentIncHigh}(x)) =$$

$$f_{\mathbf{w}}(\mathit{hsGradeHigh}(x), \mathit{parentIncHigh}(x))$$

  where $f_{\mathbf{w}}(.)$ might be a neural network

- This is addressed in the field of *Relational Learning*

- The term *Statistical Relational Learning* emphasizes the statistical nature of the dependencies. Example, *Probabilistic Relational Models* (PRMs)

- In *Inductive Logic Programming* the learned rules are deterministic or close to deterministic. Example: FOIL

Relations with Higher Arity

- So what is the big deal: the training data is extracted from a database. So what?
- Well, there are not only unary relations (i.e., relations with one column) but also relations with higher arity, such as binary relations. Example:

$$\textit{knows(Person, Person)}$$

- How do we predict that? Two people might know each other when they have similar attributes: this can be modelled with standard ML as

$$\textit{Knows} = f_w(A_{1,1}, A_{1,2}, ..., A_{2,1}, A_{2,2}, ...)$$

- Another approach is to derive a kernel based on some similarity measure (see SVM, Gaussian processes)

Introducing a binary relation

candidates
Jack
John
Mary
Lisa

hsGradeHigh
Jack
Lisa

parentIncHigh
Jack
John

highPotentials
Jack
Mary

friends	
Jack	John
John	Jack
Mary	Lisa
Lisa	Mary

Higher-Order Relations (cont'd)

- But then, two people might know each other with higher likelihood if there is a common person both know (in social networks this is called a triangle rule)
- Also: if the two persons have common ancestors might be relevant; finding out if two people have common ancestors requires complex database queries as a preprocessing step
- Another example, in recommendation systems one considers the relation *rating(user, movie)* and one explores patterns in movie preferences without looking at the attributes of the user or the movie
- So, despite the fact that it is often possible to turn a relational learning problem into a standard instance based learning problem e.g., by some for of smart preprocessing and aggregation, this is not elegant and also often leads to suboptimal solutions

First-Order Logic

- To understand the challenges of machine learning in relational domains we start with a look at *first-order logic* (FOL) (predicate logic)
- In FOL the available information ("training data" and more) is available in form of a *theory*, a.k.a. a *knowledge base*, consisting of *sentences* / *assertions* / *statements* (A *formula* with no free variable occurrences is called a sentence)
- A theory is like a book which tells us something about the domain of interest
- *Syntax* is concerned with the question, if every sentence in the theory is grammatically correct
- *Semantics* associates the theory with the (intended) (real) world M (called a *model*) consisting of the constants or entities (i.e., the *domain*) and the *interpretation* (which states which predicate instances (ground atoms) are true and which ones are false)

First-Order Logic (cont'D)

- Of course: every sentence in the theory (the book) should be true in the model: A first-order model that satisfies all sentences in a given theory is said to be a *model of the theory*. In other words: each sentence in the book is correct

- Deduction: In reality, we only have the theory and cannot query the model to get more information. Is it still possible to infer additional statements? A deductive system is used to demonstrate, on a purely syntactic basis, that one formula is a logical consequence of another formula (logical inference)

- *Theorems* are sentences that can be derived from *axioms*; *axioms* are sentences that cannot be derived; in principe, a theory only needs to contain the axioms

Example

- A theory (book) typically consists of *atomic sentences* (here ground atoms), such as *candidate(Jack)*, *hsGradeHigh(Jack)*, *parentIncHigh(Jack)*

- It might also contain *complex sentences* as

$$\forall x.\ \textit{highPotential}(x) \leftarrow \textit{parentIncHigh}(x) \wedge \textit{hsGradeHigh}(x)$$

- The atomic sentences, together with the complex sentences form the axioms

- From the theory we can conclude the *theorem* that *highPotential(Jack)* must be true

Comments

- Sometimes a theory contains a logical contradiction: *then it is not satisfiable* (so there is a problem in the book). A theory is *consistent* if it is not possible to prove a contradiction from the axioms of the theory. In the example: if Jack has rich parents and a high grade in high school but it is known that he does not have a high potential, then this is a contradiction

- Maybe at a later stage the theory becomes larger: the model provides more sentences; in a way these can be considered the test data: *candidate(Mary)*, *hsGradeHigh(Mary)*, *parentIncHigh(Mary)*

- Even when the test data is included, the model should remain to be a model of the extended theory and no logical contradictions should occur

- Also, in the extended theory we might be able to derive new theorems: *highPotential(Mary)*

What is the Problem with Deductive Approaches

- FOL and deduction were the AI champions from the late sixties to the mid eighties of the last century: sometimes the term GOFAI (good old fashioned AI) is used for that period. 1984: Collapse of many Silicon Valley start-ups (beginning of the AI winter);

- A knowledge base (theory) contains *atomic sentences*, in particular ground atoms. These are like training data and it might be reasonable to assume in the application that the data is correct (ML is typically stable w.r.t. a few incorrect data points); so here is not the problem!

- A knowledge base (theory) might contain *complex sentences*

- If the domain is human generated, sometimes complex sentences exist and can be defined. For example, humans might define that a dog is a mammal. If everyone follows that definition, then the rule is useful. In biomedicine experts maintain different medical ontologies (Gene Ontology, ICD codes)

What is the Problem with Deductive Approaches (cont'd)

- Similarly, the production of a car might follow rules: a particular engine is always used with a particular transmission; no exception
- In fact, one main application of deductive reasoning is model checking: Given a model of a system, exhaustively and automatically check whether this model meets a given specification: One tries to prove that two trains can meet heads on, and hopefully thr proof fails
- Rules are effective as instructions in human-generated procedural code; but it might be difficult to argue that adding an if-statement turns a program into an AI system
- A problem is that it is often extremely difficult and cumbersome to obtain useful rules from experts
- In natural domains, like social networks and user/movie preferences, dependencies have more of a statistical character

- Consider that probability theory is a mathematical theory and statistics is its application to certain aspects of reality; Bayesian statisticians and non-Bayesian statisticians disagree on how this should be done

- Similarly: FOL is a mathematical theory; its application to aspects of reality is more limited than it was hoped for

Subsets of FOL

- Inference in FOL can be computationally demanding, so often one only works with subsets of FOL
- For example, pure prolog is restricted to Horn clauses (i.e., some form of rules) where effective forward chaining and backward chaining is applicable
- Forward chaining: rules "fire" when the premise (if-part) becomes true for a binding of variables and make the conclusion true; in the example, the binding $x/Jack$ makes the conclusion *highPotential(Jack*) true; my goal is reached!
- Backward chaining: is an inference works backward from the goal. It is used in automated theorem provers, inference engines, proof assistants, and other artificial intelligence applications; the goal is to show that *highPotential(Jack)* is true; there is a rule which tells me that this is true when *hsGradeHigh(Jack)* and *PatientIncHigh(Jack)* is true; fortunately, these statements are in my theory, so I am done!
- One of the advantages of forward-chaining over backward-chaining is that the reception of new data can trigger new inferences, which makes the engine better suited to dynamic situations in which conditions are likely to change

- The Semantic Web formalisms are based on Description Logic, another subset of FOL

FOIL: Inductive Logic Programming

Rule Learning in ILP

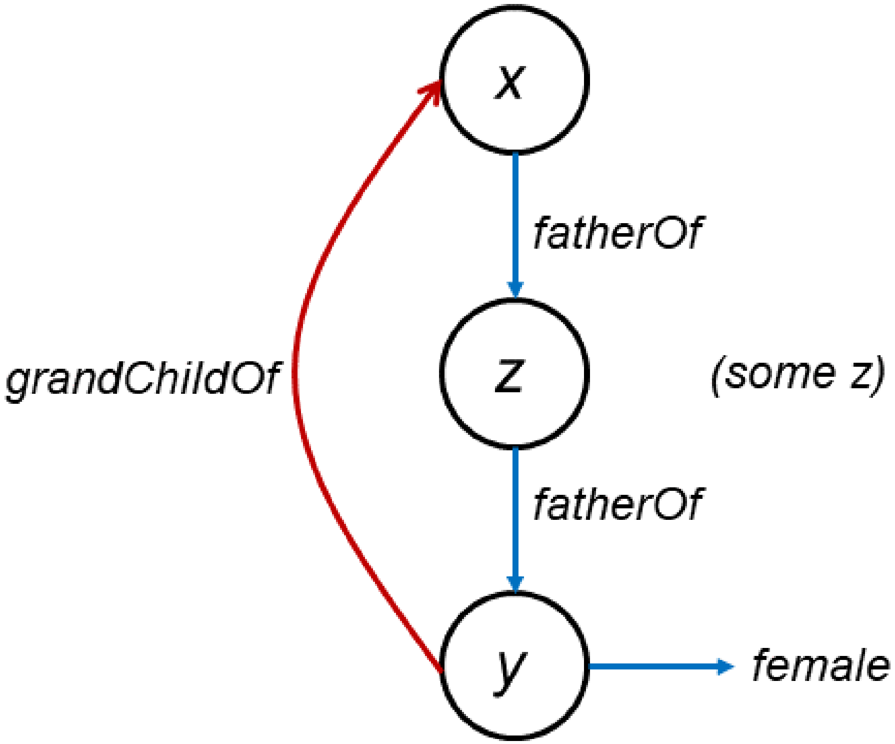
- Inductive logic programming (ILP) concerns the learning of (close to) logical rules from a knowledge base / database
- An important approach is the first-order inductive learner (FOIL), developed in 1990 by Ross Quinlan and nicely described in the Machine Learning book by Tom Mitchell (available online)

FOIL

- FOIL learns (function-free) Horn clauses; a definite Horn clause can be written as an implication (rule)
- Here is an example:

$$\forall x.y.\ \mathit{grandDaughter}(x, y) \leftarrow \exists\ z.\mathit{father}(y, z) \wedge \mathit{father}(z, x) \wedge \mathit{female}(y)$$

- It reads: For all entities x and y it is true that: y is the granddaughter of x if y is female and if there exists another person z. z is the father of y, and x is the father of z
- This rule is sound: if the premise (the right side of the rule, the rule body) is correct (it fires), then the conclusion (rule head) is always correct; the rule has a precision of one


x
fatherOf
z
(some z)
grandChildOf
fatherOf
y
female

FOIL Knowledge Base

- For the application of FOIL, we assume a knowledge base with relations *female*, *father* and *grandDaughter*
- The domain consists of *Sharon*, *Bob*, *Tom* and *Victor*
- The knowledge base contains the instances *female(Sharon)*, *father(Sharon, Bob)*, *father(Tom, Bob)*, *father(Bob, Victor)*, *grandDaughter(Victor, Sharon)*
- The convention is that *father(Sharon, Bob)* means that the *father of Sharon is Bob*, or in other words: *Bob is the father of Sharon*
- We make the closed-world assumption which means that all other ground atoms are false: e.g., *female(Bob)* is false

female
Sharon

father	
Sharon	Bob
Tom	Bob
Bob	Victor

grandDaughter	
Victor	Sharon

FOIL: Start

- FOIL starts with

$$\forall x.y.\ \textit{grandDaughter}(x, y) \leftarrow$$

  which means that each pair of individuals has a *grandChild* relation.

- Consider *precision* $= tp/(tp + fp)$ and *recall* $= tp/(tp + fn)$. With 4 individuals *(Sharon, Bob, Ton, Victor)*, we start with perfect recall 1/1 (we don't miss a grandchild relation), but the precision 1/16 is terrible

FOIL: Step 2

- By adding *father(y, z)* we get

$$\forall x.y.\ \mathit{grandDaughter}(x, y) \leftarrow \exists\ z.\mathit{father}(y, z)$$

  we only consider individuals y with a father z in the KB, i.e., Victor is out as being a grandchild; we still have perfect recall and precision improves to $1/12$;

- Note: Foil adds atoms which contain variables, here *father(y, z*); at least one of the variables must already exist (here y); if a new variable is added (here z) it is associated with an existential quantifier ($\exists$)

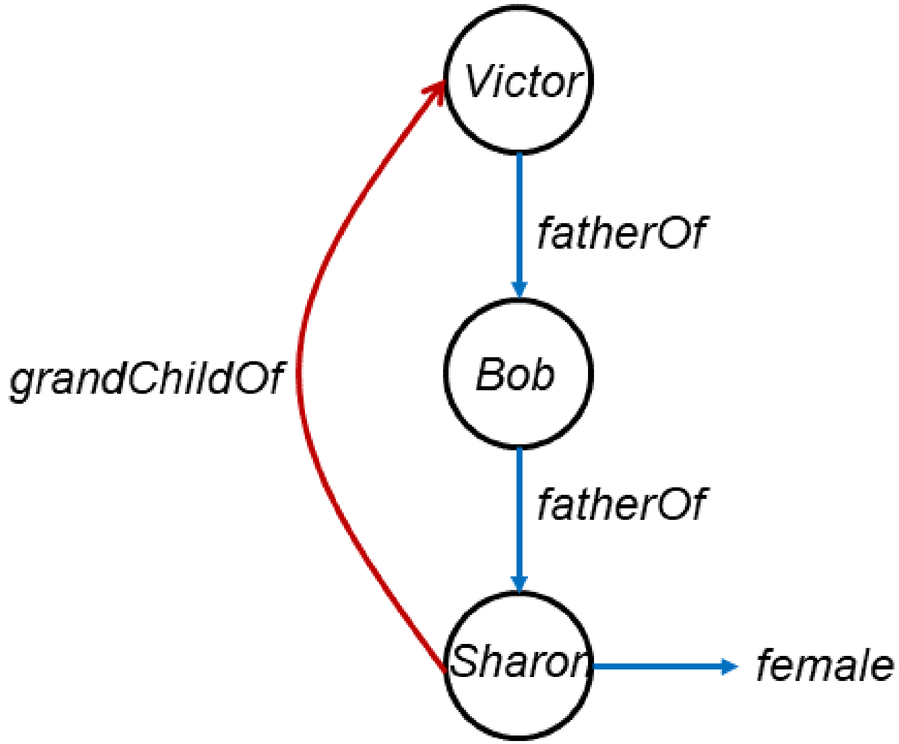
FOIL: Steps 3 and 4

- By adding *father(z, x)* we get

$$\forall x.y.\ \textit{grandDaughter}(x, y) \leftarrow \exists z.\ \textit{father}(y, z) \wedge \textit{father}(z, x)$$

  we only consider individuals x who are the father of z; we are left with the options *grandDaughter(Victor, Sharon)* and *grandDaughter(Victor, Tom)*; we still have perfect recall and precision improves to $1/2$;

- Now we add *female(y)* and we get perfect recall and precision


Victor
fatherOf
grandChildOf
Bob
fatherOf
Sharon
female

FOIL: Adding Literals and FOIL Gain

- In addition to adding atoms, FOIL might add in a step their negation (example: $\neg$*father*(y, z)), or might set two already existing variables to be equal (example: *equal*(y, z))

- In each step, FOIL adds the new literal L (ground atom, negated ground atom, equal) with the best FOIL gain defined as

$$t\,(\log_2 \mathit{precision}(R + L) - \log_2 \mathit{precision}(R))$$

  The equation says that precision should go up; t is the number of true positive cases of R that remain covered under $R + L$

Recursive Rule Sets

- FOIL permits the learning of recursive rule sets
- Example:

$$\forall x.y.\ ancestor(x, y) \leftarrow parent(x, y)$$

$$\forall x.y.\ ancestor(x, y) \leftarrow \exists z.\ parent(x, z) \wedge ancestor(z, y)$$

Comments

- FOIL is one of the most popular examples of an ILP approach and many extensions have been developed
- Note the power from the existential quantifier: it searches for certain paths in the KB
- ILP works best in domains where dependencies are either deterministic or close to deterministic

Learning of Probabilistic Rules

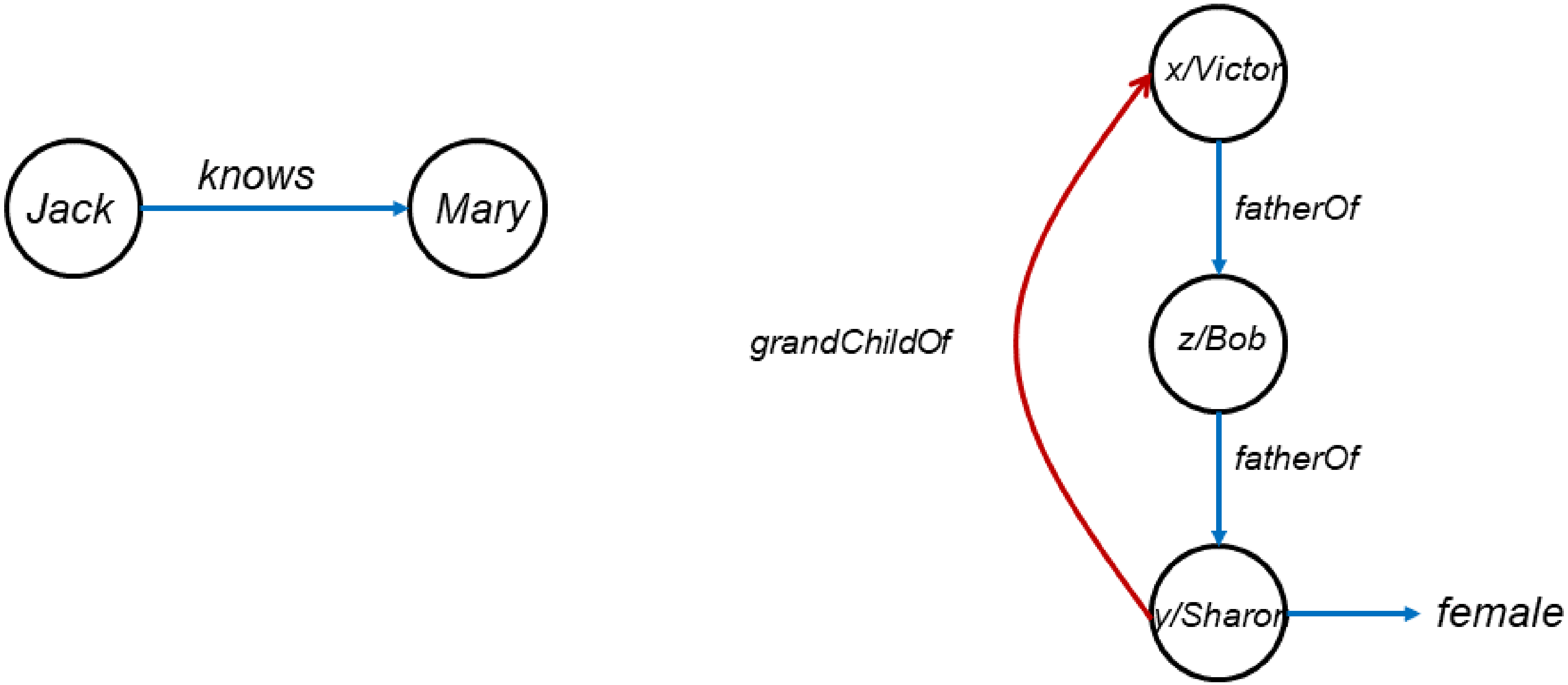
- Statistical Relational Learning (SRL) has been employed when dependencies are more statistical in nature
- Important examples are the probabilistic relational models (PRMs) which are relational extensions of Bayesian networks
- Markov Logic Networks (MLNs) implements soft rule constraints into relational learning
- Bayesian Logic Programming (BLP) is a probabilistic extension to ILP
- Although effective in some domains, these approaches have not been as successful in application, as it was hoped for

RESCAL: Learning Knowledge Graphs using Latent Representations

Knowledge Graphs

- Consider the tuple *knows(Jack, Mary)*
- We now write this as the triple *(Jack, knows, Mary)* where *Jack* is the subject, *knows* the predicate and *Mary* the *object*
- A database of binary relations can be transformed into a knowledge graph, where entities *(Jack, Mary)* are the nodes and predicates become labelled directed links pointing from subject to object

Knowledge Graphs



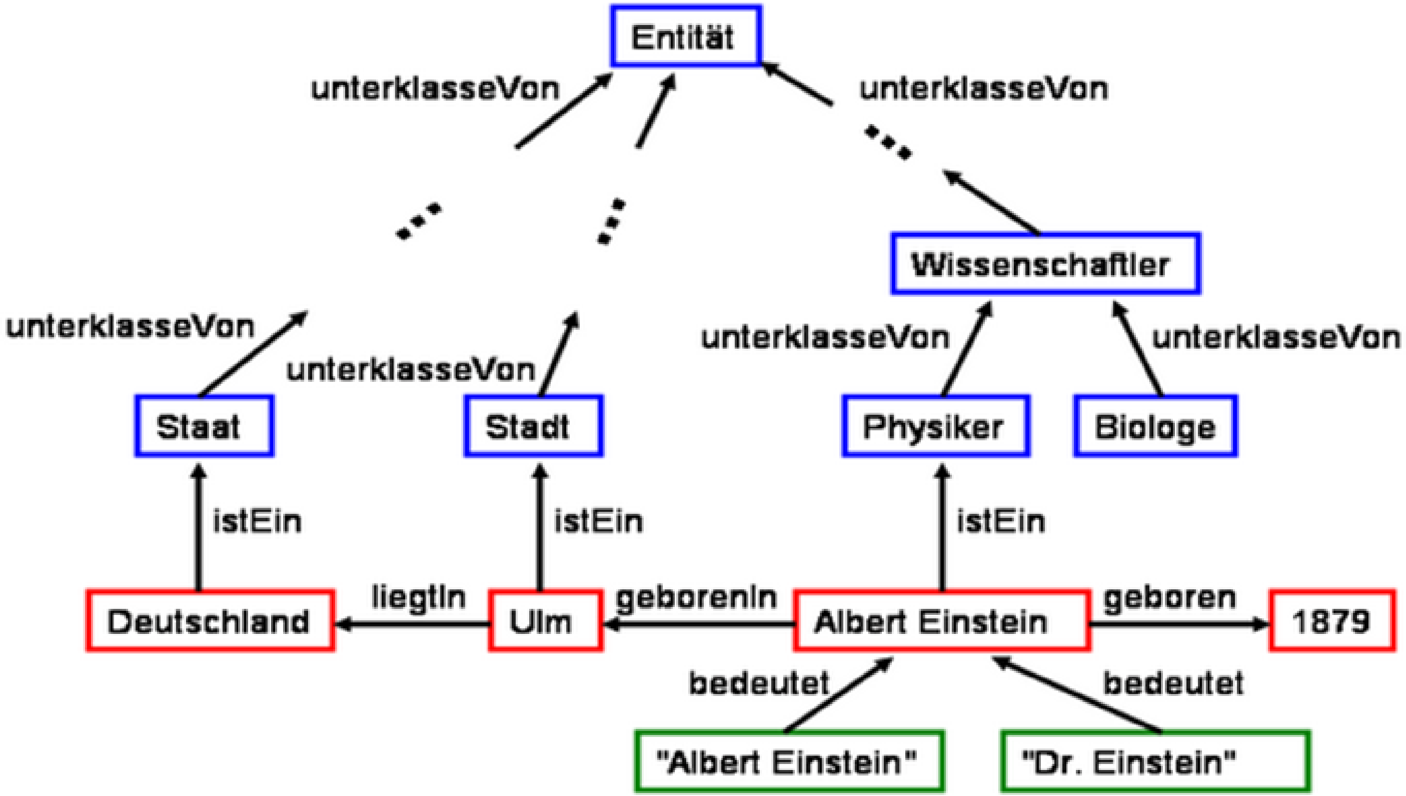
Knowledge Graphs (cont'd)

- KGs are easy to understand and for many users more accessible than, e.g., relational databases
- KGs can be represented in graph databases with efficient implementation of relevant algorithms (e.g., querying)
- Essentially, a KG is a set of triples, so a KG database is sometimes called a triple store which can be represented as one big table with three columns and where each row is a triple
- KGs can be used in conjunction with RDF-ontologies popular in the Semantic Web community; RDF stands for *Resource Description Framework*
- KGs are the basis for linked open data (LOD), i.e., the effort to connect open databases
- Most importantly: commercial KGs, like the Google KG, have been developed which are scalable, where the quality issues have been solved, and which have been made useful for search, text understanding and Q&A

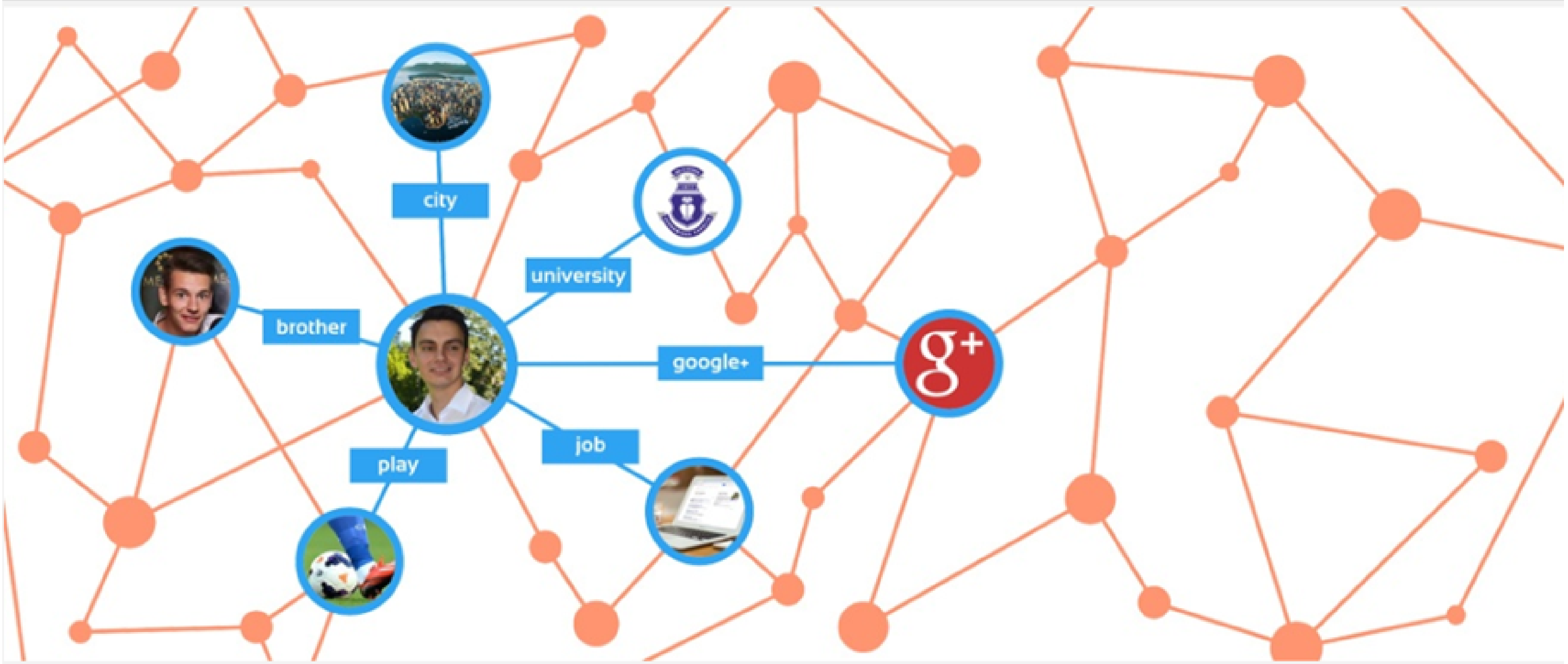
Knowledge Graphs (cont'd)

- There is no generally accepted theory behind KGs; here are some of their common properties
- An entity is typically linked to one or several class nodes via the type relation: *(Sparky, type, dog)* would indicate that Sparky is a dog and *(Sparky, type, mammal)* would indicate that Sparky is a mammal
- Depending on the application one should think of a class as a simple attribute of the entity or as an entity representing the set of its class members
- A class can be a subclass of another class, which is expressed as *(dog, subclass, mammal)*
- One might then reason that if Sparky is a dog, it is also a mammal
- Type constraints can be used to model, e.g., that only humans can be legally married

Yago Ontology



Suchanek, Kasneci, Weikum: 2007



Knowledge Graph

Comments

- *(Labelled-)Property Graphs* are KGs developed in the database community (Neo4j)
- Higher-order relations can be transformed into a KG by using so called blank nodes
- Unary relations can be treated in several different ways, e.g., introducing attribute nodes *(Jack, hasAttribute, Tall)*

Relational Learning in KGs

- ILP methods and kernel methods have been extended to be applicable to KGs; note that for KGs one often makes an open-world assumption in which some ILP approaches are difficult to apply
- More successful are approaches using *latent representations* of entities and predicates
- Early approaches are *statistical block models* and the *infinite (hidden) relational model* (IRM, IHRM) which generalize statistical mixture models (soft clustering) to relational domains
- The SUNS approach applied singular value decomposition (SVD) to a matrix generated from the KG by a matricification of the adjacency tensor
- A certain breakthrough was the RESCAL model, described next

RESCAL

- The RESCAL model estimates the probability that a triple exists, given all the available information in the KG
- The RESCAL model is

$$P((s, p, o)|KG) = \mathit{sig}\left(\sum_{m=1}^{r}\sum_{n=1}^{r} a_{s,m}a_{o,n}R_{m,n,p}\right)$$

- This can be written as

$$P((s, p, o)|KG) = \mathit{sig}(\mathbf{a}_s^T R_p \mathbf{a}_o)$$

  Here $\mathbf{a}_s = (a_{s,1}, \ldots, a_{s,r})^T$ is the latent representation of the subject, $\mathbf{a}_o = (a_{o,1}, \ldots, a_{o,r})^T$ is the latent representation of the object, and R_p is a matrix with dimensions $r \times r$ and is the latent representation of the predicate

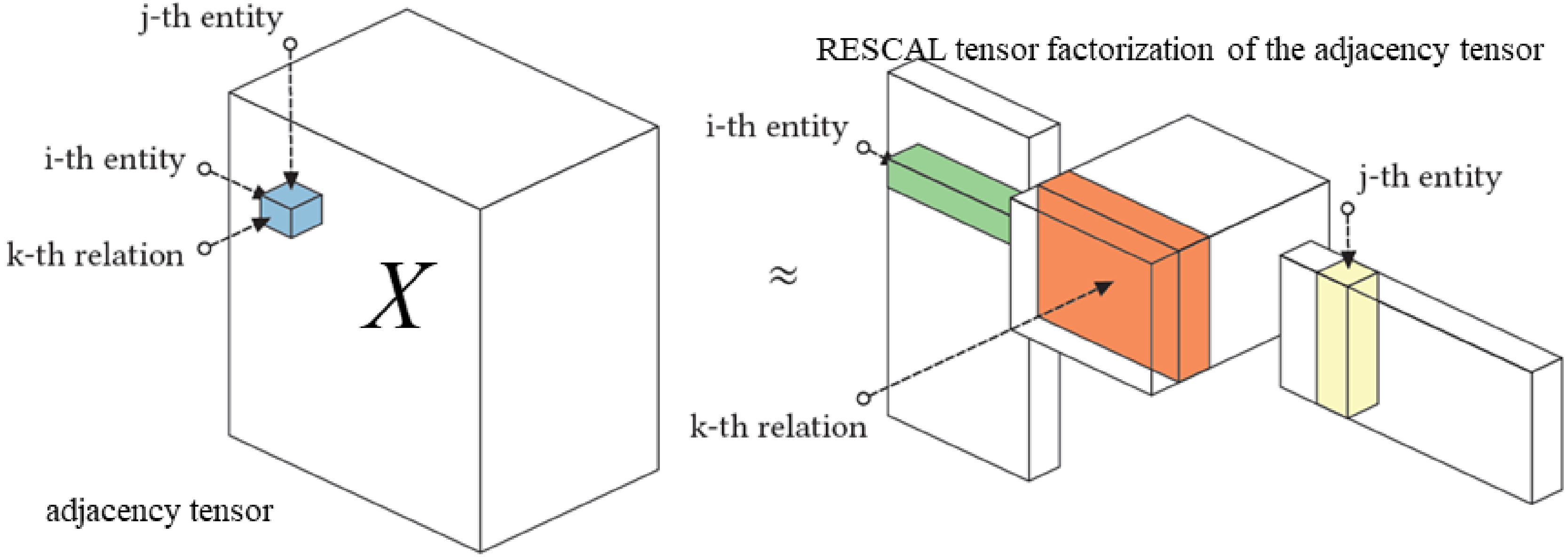
RESCAL as Tensor Factorization

- Let $\mathcal{X}$ be the adjacency tensor of the KG: $x_{i,j,k} = 1$ if $(s = i, p = k, j = o)$ is known to exist and 0 otherwise.
- Consider a Tucker2 tensor decomposition of the form (using the n-mode product $\times_n$)

$$\mathcal{X} \approx \mathcal{R} \times_1 A_s \times_2 A_o$$

- If we ignore the sigmoidal transfer function, the RESCAL model describes a Tucker2 decomposition with the constrain that $A_s = A_o$, which means that entities have unique representation, independent if they act as subject or object; $\mathcal{R}$ is the core tensor with R_p as slices

RESCAL as Tensor Factorization



Training Data:

$x_{s,p,o} = 1$ If *(s,p,o)* is known to be true

$x_{s,p,o} = 0$ otherwise

After factorization (RESCAL2: constr. Tucker2):

$$P((s,p,o)) = \mathrm{sig}(\theta_{s,p,o})$$

$$\theta_{s,p,o} = \sum_{r_1} \sum_{r_3} a_{e_s,r_1} \, a_{e_o,r_3} \, g(r_1, p, r_3)$$

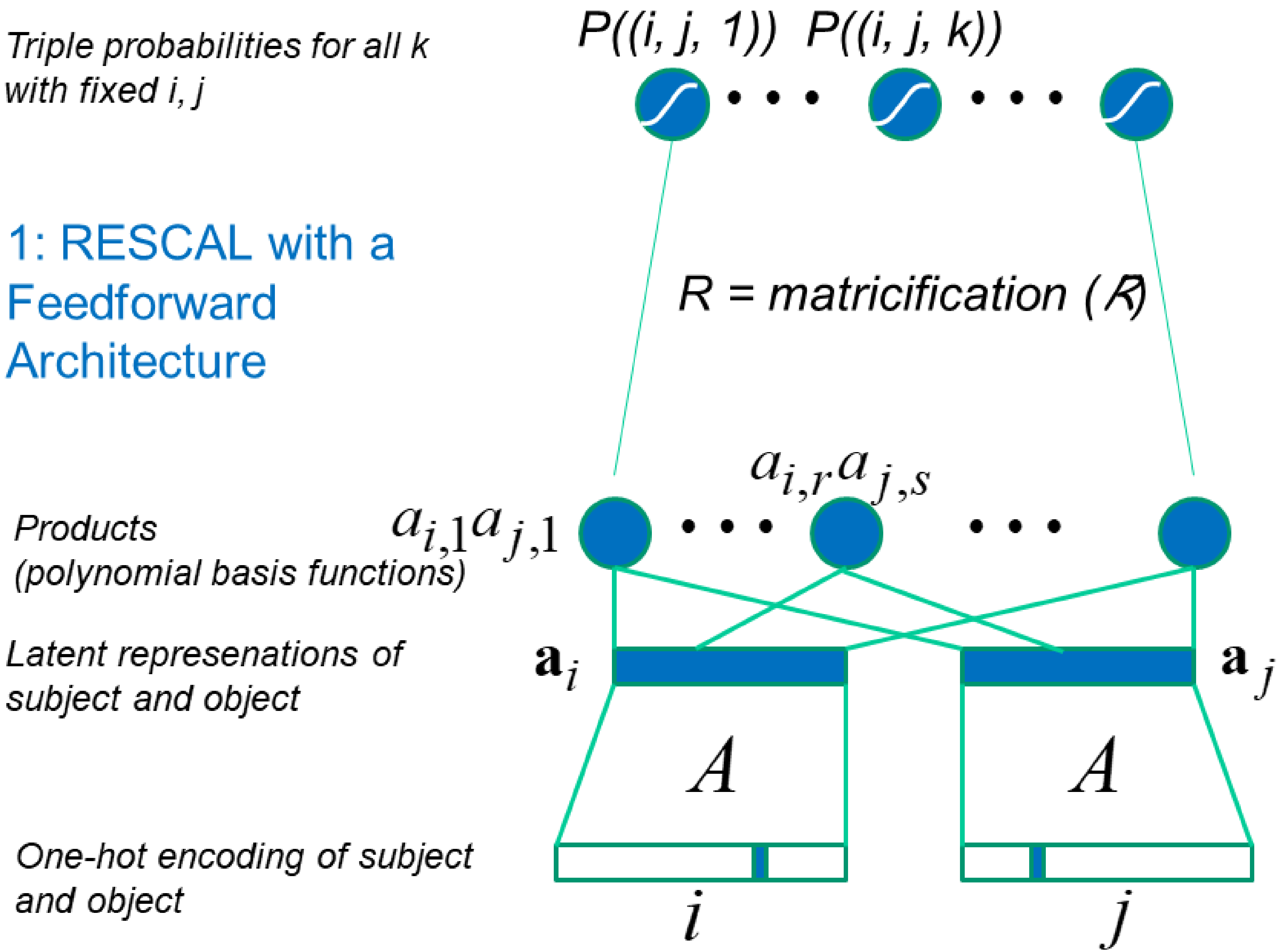
$$\Theta = G \times_1 A \times_2 A$$

Nickel, Tresp, Kriegel. A Three-Way Model for Collective Learning on Multi-Relational Data. ICML 2011

Feedforward Architectures

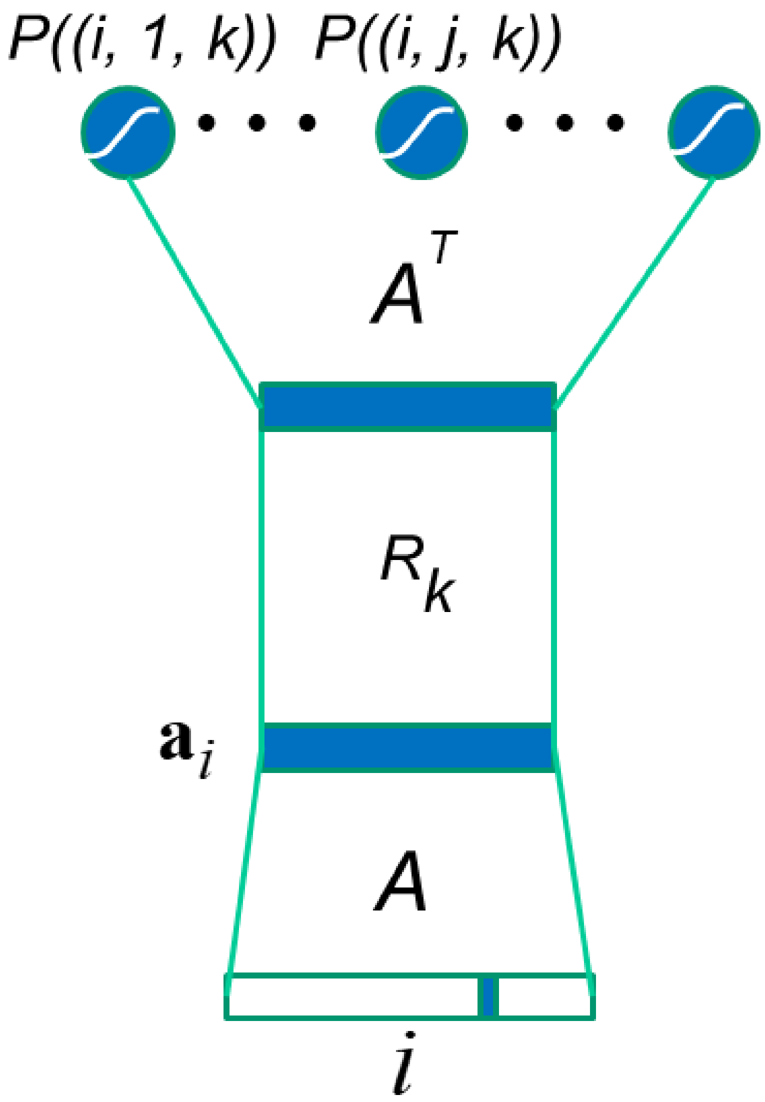
- The next slides (1, 2) show two feedforward architectures for RESCAL and a variant (3) which replaces the core tensor by a neural network
- Note that if the latent representations would be known features of the entities (i.e., they are not latent) then RESCAL is a polynomial classifier with quadratic polynomials; in RESCAL the features are latent and are learned in the optimization process (i.e., during training)

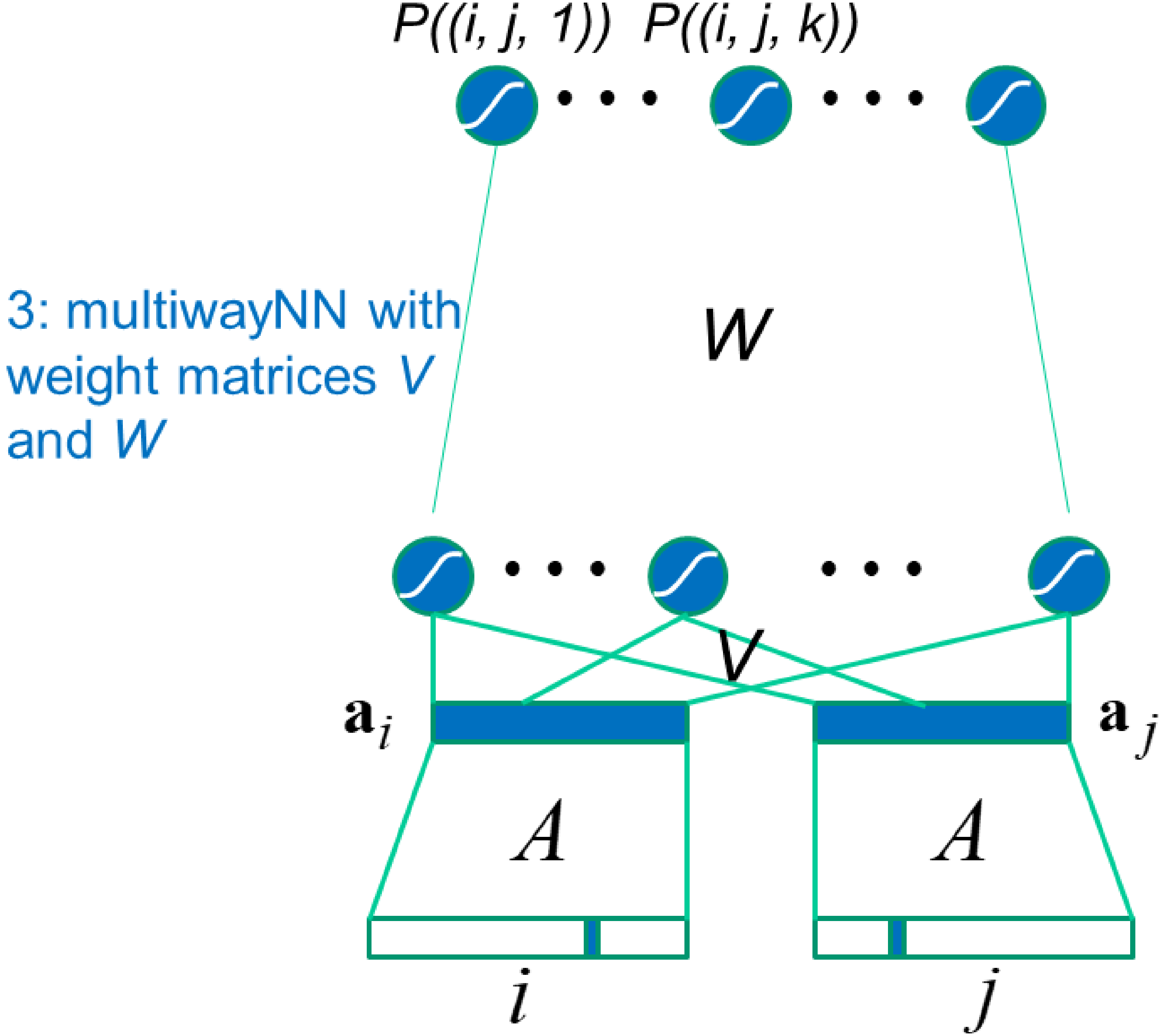
1: RESCAL with a Feedforward Architecture



All triple probabilities for a specific predicate k

2: RESCAL with a Feedforward Architecture




P((i, j, 1)) P((i, j, k))
3: multiwayNN with weight matrices V and W
W
V
$\mathbf{a}_i$
$\mathbf{a}_j$
A
A
i
j

RESCAL Cost Function

- One can use the usual Bernoulli cost function

$$\sum_{i,j,k} (x_{i,j,k} \log \mathit{sig}(\mathbf{a}_i^T R_k \mathbf{a}_j) + (1 - x_{i,j,k}) \log(1 - \mathit{sig}(\mathbf{a}_i^T R_k \mathbf{a}_j))$$

$$+\lambda_1 \sum_{i,m} a_{i,m}^2 + \lambda_2 \sum_{m,n,k} R_{m,n,k}^2$$

- In large KGs, the 0's dominate the cost function so one typically performs SGD where positive and negative examples are balanced (local closed-world assumption)

Applications and Variants

- Our team has developed a part-recommendation for industrial solutions; it was shown that the RESCAL model was much more powerful than the typical recommendation systems based on matrix factorization
- Since RESCAL has been introduced, many variants have been developed: RESCAL, DistMult, HolE, ComplEx, ConvE, ...

Configuration Support System

Historical data

Contains information about 35,888 previously configured (anonymized) solutions containing 6,865 different items.

Technical features

Contains information about technical features of the items, such as voltage, size, weight, material, etc.

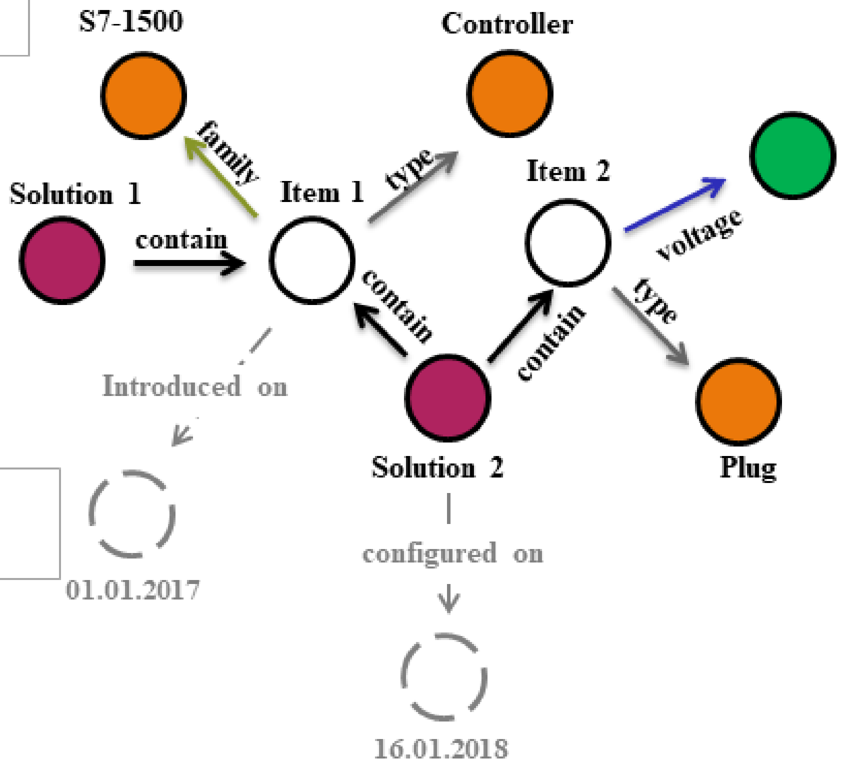
While most of the features are **numerical**, they belong to different scales: nominal, ordinal, interval, ratio.

Catalog data

Contains the information for categorization of the product.

Temporal data

Contains information about when a given solution was configured and when a given item was first introduced to the TIA Portal.



Hildebrandt, Shyam, Mogoreanu, Thon, Tresp, Runkler. Configuration of Industrial Automation Solutions Using Multi-relational Recommender Systems. ECML, 2018.

Graph Convolution: Learning Knowledge Graphs with Deep Learning

A Regular Neural Network

- Consider a normal neural network with D inputs and F outputs and

$$h_j(l+1) = \textit{sig}\left(\sum_k w_{j,k}(l) h_k(l)\right)$$

- Neuron $h_j(l+1)$ is the j-th neuron in layer $l+1$ and $w_{j,k}(l)$ is the weight from neuron $h_k(l)$ to neuron $h_j(l+1)$

- This equation can be applied to data point or entity i, and we write

$$h_{i,j}(l+1) = \textit{sig}\left(\sum_k w_{j,k}(l) h_{i,k}(l)\right)$$

Entities with a Graph Structure

- Now we assume that a data point defines an object (e.g., a person) and there is some neighborhood relation (e.g., knows) between the objects
- Then, maybe we should average the activations over the direct neighborhood of the node, and we write

$$h_{i,j}(l+1) = \mathit{sig}\left(\sum_k w_{j,k}(l)\bar{h}_{i,k}(l)\right)$$

- This is called a graph convolutional network (GCN)
- If there are N entities, we now have one neural network with ND inputs and NF outputs, but with only one training data point!
- Often, some of the outputs are unknown (semisupervised learning)
- Generalization: After training, we can add additional entities, without retraining the weights!

Calculating the Average

- In the simplest case

$$\bar{h}_{i,k}(l) = \frac{1}{N_i} \sum_{i' \in nb(i)} h_{i',k}(l)$$

Here $nb(i)$ are the neighbors of i, including i, and N_i is the number of neighbors of i, including i

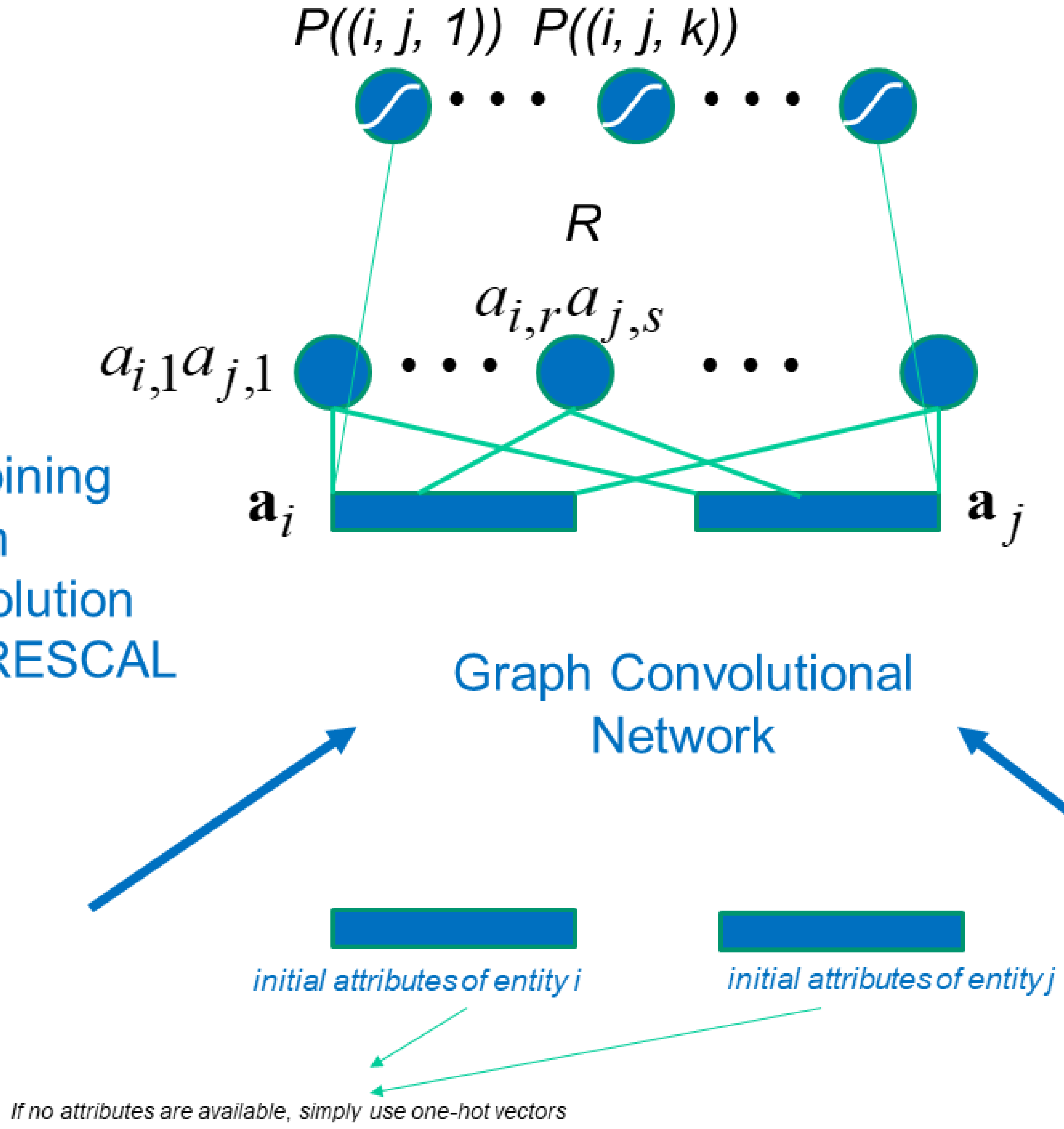
- In the literature, one often sees the operation described as

$$H(l+1) = sig(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H(l)W(l))$$

Here $\tilde{A} = A + I$ where A is the adjacency matrix, I is the identity matrix and $\tilde{D}$ is the diagonal node degree matrix of $\tilde{A}$. Thus $d_{i,i}$ is thenumber of neighbors of node i, plus 1.

Application to KG Learning

- GCNs can be combined, e.g., with RESCAL as shown in the next slide, to predict relationships
- The GCN is an encoder and calculates the latent representations for the entity, and the RESCAL model is the decoder
- The node neighborhood is determined by the known triples; different variants are used here; some approaches introduce predicate specific weight parameters



P((i, j, 1))
P((i, j, k))
R
$a_{i,r}a_{j,s}$
$a_{i,1}a_{j,1}$
Combining Graph Convolution with RESCAL
$\mathbf{a}_i$
$\mathbf{a}_j$
Graph Convolutional Network
initial attributes of entity i
initial attributes of entity j
If no attributes are available, simply use one-hot vectors

Conclusion

- There is rapidly growing interest in learning with KGs and other relational structures
- In industry, a lot of data is available as structured data (e.g., in databases) but there is an abundance of unstructured data (text, images, videos,sensor data) as well
- KG learning can be combined with unstructured data in many interesting ways: for example, to get a deep description of a visual scene
- Another interesting research direction analyses if KGs might be useful as models for human semantic and episodic memory

Appendix: More on FOL

First-Order Logic

- If a sentence φ evaluates to true under a given interpretation M, one says that M satisfies φ; this is denoted $M \models \varphi$; A first-order model that satisfies all sentences in a given theory is said to be a *model of the theory*. In other words: each sentence in the book is correct in this model (world)
- Deduction: In reality, we only have the theory and cannot query the model to get more information. Is it still possible to infer additional statements? A deductive system is used to demonstrate, on a purely syntactic basis, that one formula is a logical consequence of another formula (logical inference)

First-Order Logic (cont'd)

- A formula φ is a *semantic consequence or semantic entailment* of a set of statements Γ if and only if there is no model M in which all members of Γ are true and φ is false. Or, in other words, the set of the interpretations that make all members of Γ true is a subset of the set of the interpretations that make φ true. One writes,

$$\Gamma \models \varphi$$

- A formula φ is a *syntactic consequence or syntactic entailment* of a set Γ of formulas if there is a formal proof (under proof system S) and one writes,

$$\Gamma \vdash_S \varphi$$

  Syntactic consequence does not depend on any interpretation of the formal system

- If $(\Gamma \models \varphi) \leftarrow (\Gamma \vdash_S \varphi)$, then S is sound

- If $(\Gamma \vdash_S \varphi) \leftarrow (\Gamma \models \varphi)$, then S is complete

- Inferred sentences are called *theorems*