UNIVERSITÀ DEGLI STUDI FIRENZE

DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
DISTRIBUTED DATA INTELLIGENCE AND TECHNOLOGIES LAB

https://www.snap4City.org

SNAP4CITY

KM4CITY

NeuroSymbolic Artificial Intelligence at Scale

Marco Fanfani, marco.fanfani@unifi.it

https://www.disit.org/

Parte: P6.2 (2025-26) - Knowledge Graph Completion

The Problem of Incompleteness in KG

- Knowledge Graphs in real-world applications are almost **never complete**.
- They are built from human-generated, extracted, or inferred knowledge, and for this reason they inevitably reflect only a partial view of reality.
- Many facts are simply **not recorded**, others **evolve over time**, and some are implicitly true but **never explicitly represented**.
- This means that even large-scale KGs contain a significant amount of **missing information**, which limits their ability to fully represent real-world knowledge.

Incompleteness is a Critical Issue

- The incompleteness of a KG has direct consequences on downstream applications
- Question answering, recommendation systems, and semantic search depend heavily on the availability of relevant facts in the graph
  - When key relationships are missing, the system may fail to answer queries that are actually valid in the real world
- In other words, the limitation is not in the reasoning process itself, but in the **absence of explicit knowledge**

Incomplete KGQA

- Incomplete KG leads to **Incomplete Knowledge Graph Question Answering** problem, or IKGQA

> *In this setting, a system is asked to answer a question, but the required information may not be directly available in the graph*

- The key difficulty here is that the system must distinguish between two situations:
  - whether a fact is truly false, or
  - whether it is simply missing from the knowledge graph
- This distinction is fundamental and relies on the **Open World Assumption**, where the absence of evidence does not imply evidence of absence

From IKGQA to Knowledge Graph Completion

- To address this limitation, a fundamental task called **Knowledge Graph Completion** (KGC) has been introduced
- The goal of KGC is to **automatically enrich a knowledge graph** by predicting missing entities or relationships
- Rather than assuming the graph is complete, KGC methods attempt to **infer plausible missing facts** based on observed patterns in the existing data
- In this sense, completion becomes a way of compensating for structural incompleteness

Knowledge Graph Completion

- From a more formal perspective, we can think of a knowledge graph as composed of **observed facts** and **unobserved facts**
- The goal of KGC is to infer the missing portion of this structure
- This is typically done by learning a **function that assigns a plausibility score to candidate triples**, allowing the system to estimate which missing facts are most likely to be true given the observed data

Link Prediction

- The most widely studied formulation of KGC is **link prediction**
- In this setting, the system is given a **partial triple**, such as a head entity and a relation, and must predict the missing tail entity
- For example, given *Google is located in ?*, the model is expected to rank possible locations and assign high probability to the correct answer
- This is typically framed as a **ranking problem** over all candidate entities.

Relation Prediction and Semantic Understanding

- A complementary task is **relation prediction**, where the goal is to infer the type of relationship between two given entities
- For instance, given `Steve Jobs` and `Apple Inc.`, the system should infer that the relation is `founded`
- This task is particularly important because
  - it moves beyond entity prediction
  - focuses on understanding the semantics of how entities are connected in the graph

Predicting Heads, Tails, and Relations

- In practice, KGC can be decomposed into three symmetric subtasks:
  - predicting the tail entity,
  - predicting the head entity, and
  - predicting the relation between two entities
- All these tasks can be seen as **classification problems over a large candidate space**, where the model assigns scores to possible completions and selects the most plausible one

Triple Classification as Fact Verification

- Another important formulation is **triple classification**
- Instead of predicting missing elements, the system is given a complete triple and must decide whether it is valid or not
- For example, a triple like *Einstein was born in Germany* can be evaluated as true or false based on the learned representation of the knowledge graph
- This task is closely related to **fact verification** and **consistency checking**

Knowledge Sparsity

- A major challenge in KGC is sparsity
- Real-world knowledge graphs **follow a long-tail distribution:**
  - a small number of entities are highly connected, while most entities have very few relationships
- This imbalance makes it difficult for models to learn reliable patterns for rare entities, which often leads to **poor generalization** in low-data regions of the graph

Embedding-based Intuition

- To address sparsity, many methods rely on **embedding representations**, where entities and relations are mapped into a continuous vector space
  - geometric proximity is used as a proxy for semantic similarity
- The underlying idea is that **structural patterns** in the graph can be captured through **spatial relationships in a latent space**, allowing the model to generalize beyond observed facts

Global Consistency in Knowledge Graphs

- While predicting individual missing facts is useful, it introduces a new challenge:

Ensuring that all predicted facts remain globally consistent!

- A model may produce valid-looking triples individually, but these **may contradict each other** when considered together
- For this reason, consistency across the entire graph becomes an important constraint, especially in applications where logical coherence is required

Closed World vs Open World Assumptions

- A key conceptual distinction in this area is between closed-world and open-world assumptions
  - in a **closed-world** setting, anything not present in the knowledge base is assumed to be false
  - in an **open-world** setting missing information are treated as unknown rather than false
- Knowledge graphs generally operate under the open-world assumption, **which makes completion both necessary and inherently uncertain**

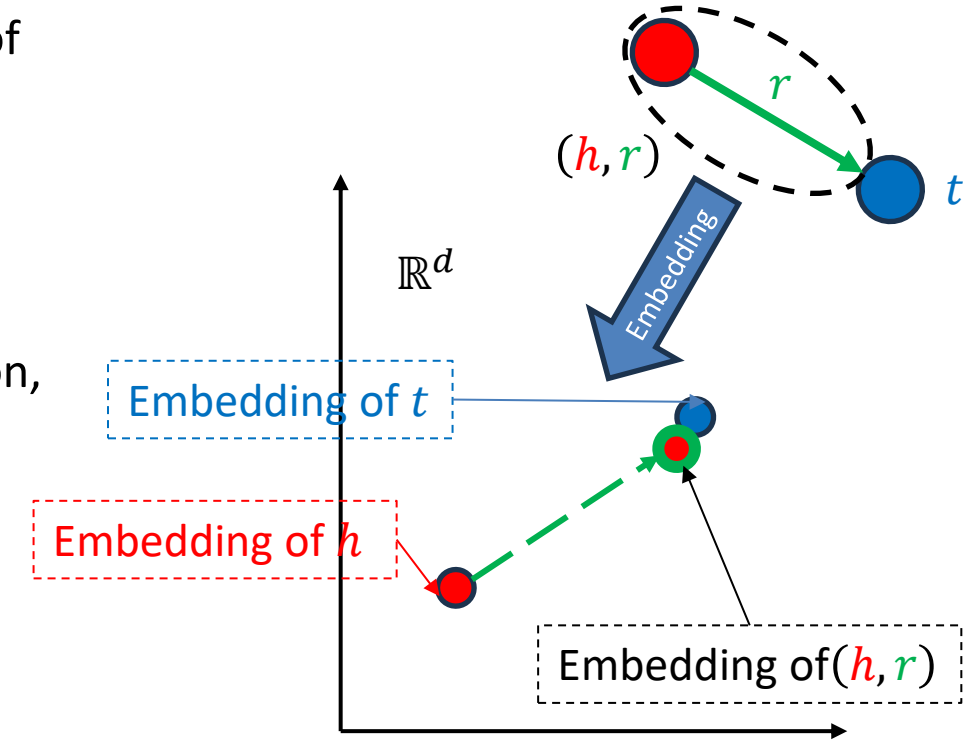
Static vs Dynamic KG and Future Directions

- KGs can be either static or dynamic
  - **Static graphs** represent a fixed snapshot of knowledge
  - **Dynamic or temporal graphs** capture how facts evolve over time
- Recent research is increasingly focused on extending completion methods to handle **evolving** and **multimodal data**
- This moves toward systems that can **continuously update and refine** their knowledge in a more lifelong learning fashion

KG Embedding

Knowledge Graph Embeddings (KGE)

- Knowledge Graph Embedding (KGE), or Knowledge Representation Learning (KRL), is the process of **mapping entities and relations into a continuous, low-dimensional vector space**
- The primary goal is to
  - preserve the inherent semantic and structural information of the graph
  - simplify complex algebraic operations
- In the embedding space
  - entities *h* are represented as points (vectors)
  - relations *r* are modelled as an operation (translation, rotation, or projection)
- This transition from symbolic to distributed representation enables the **calculation of semantic relations using standard optimization algorithms**



The Semantic Matching Objective

- KGE methods rely on a specific evaluation function, known as a **scoring function**, to measure the plausibility of a factual triple (h, r, t)
- The learning process involves
  - maximizing the **global plausibility of observed** facts
  - minimizing the **scores of unobserved or negative** facts
- By positioning similar entities closer together in the vector space, these models facilitate tasks such as **link prediction** and **entity clustering**

Limits of One-Hot Encoding

- Traditional one-hot representations treat every entity as **equally distinct**, failing to capture semantic overlap or proximity
- Embeddings address this limitation by introducing **distributed representations**, where each dimension captures latent semantic features
  - Relationships between entities can be inferred from their **relative positions** in space
- This approach is particularly effective in sparse settings, where explicit connections are limited but **underlying patterns can still be learned**

Families of KGE Models

- Current KGE research is categorized into **three main families** based on their mathematical foundations:
  - translational distance models
  - tensor decomposition models
  - neural matching models
- Each family reflects different design criteria, balancing
  - computational efficiency
  - expressiveness
  - ability to handle specific relational patterns

Translational Models

- Translational distance models interpret relations as **simple geometric translations** in the vector space.
- The fundamental assumption is that if a triple (h, r, t) is valid, the embedding of the head entity h plus the relation vector r should be approximately equal to the tail entity t

$$h + r \approx t$$

- These models minimize a **distance-based scoring function**, typically using Euclidean distance or Manhattan distance

Training and KGC

- The model is trained to **maximize the scoring functions for the triples in the KG** in order to find good embeddings
- Negative examples are created by **altering real triples randomly** changing the head, tail, or relation (Closed World Assumption)
- Graph completion is treated as an **entity sorting problem**:
  - If an item is missing in a triple (e.g., the tail in $(h, r, ?)$), the system calculates the plausibility score for each possible entity in the database using it as a candidate to complete the triple by producing a list of candidate answers sorted by descending score

TransE

- One of the earliest and most influential models in this family is **TransE**
- It models relations as **direct translations** and can be trained efficiently even on very large graphs
- Limitations: TransE **struggles to represent more complex relational patterns**, such as 1-to-N, N-to-1, symmetric relations

Bordes, Antoine, et al. "Translating embeddings for modeling multi-relational data." *Advances in neural information processing systems* 26 (2013).

TransH

- TransH was developed to address TransE's inability to **distinguish between different roles** an entity might play in various relations
- It introduces relation-specific hyperplanes, mapping entities onto these planes before performing the translation
  - For each relation, TransH learns two vectors: a **normal vector** (w_r) that defines the orientation of a hyperplane and a **translation vector** (d_r)
  - To verify the validity of a triple (h, r, t), the vector representations of the head (h) and tail (t) entities are **first projected orthogonally onto the hyperplane** defined by the normal vector of the relation
- This allows a single entity to have **different distributed representations** depending on the relation involved, improving accuracy for many-to-one or one-to-many links

Wang, Zhen, et al. "Knowledge graph embedding by translating on hyperplanes." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 28. No. 1. 2014.

TransR

- TransR posits that entities and **relations are fundamentally different objects** and should not necessarily share the same vector space
- It **builds separate entity and relation spaces** and utilizes a projection matrix M_r to project entities from the entity space into a specific relation space
- While more expressive than TransH, TransR **increases computational complexity** and the number of model parameters

Lin, Yankai, et al. "Learning entity and relation embeddings for knowledge graph completion." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 29. No. 1. 2015.

TransD and TransSparse

- TransD simplifies the TransR approach by replacing matrix-vector multiplications with more efficient **vector-based mapping operations**
  - It utilizes **mappings dynamically computed** from a coordinate and projection vector for each entity-relation pair, avoiding explicit matrix multiplications and making it highly applicable to massive, heterogeneous KGs
- TranSparse further optimizes this by introducing **sparse mapping matrices** to handle the distribution imbalance of entities and relations

Ji, Guoliang, et al. "Knowledge graph embedding via dynamic mapping matrix." *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*. 2015.
Ji, Guoliang, et al. "Knowledge graph completion with adaptive sparse transfer matrix." *Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1*. 2016.

RotatE

- RotatE treats relations as **rotations** in a complex vector space rather than translations in a real space

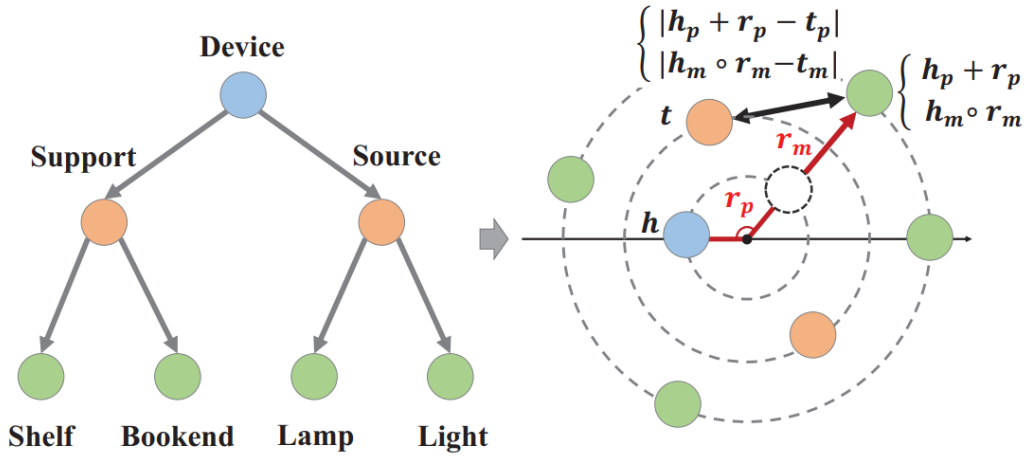
$$h \circ r \approx t$$

- This enables the model to effectively infer **symmetry**, **asymmetry**, in addition to **inversion**, and **composition patterns**
- RotatE significantly improves the ability to represent diverse relational patterns while maintaining computational efficiency
- Limitations:
  - Lack of Neighborhood Context: RotatE processes triples independently, failing to leverage local graph topology
  - Hierarchical Inefficiency: The rotation-based approach is not naturally suited for semantic hierarchies
  - Long-Tail Fragility: Performance significantly degrades for rare entities (long-tail)
  - Black-Box Nature: As a purely neural approach
  - Neglect of External Semantics: ignores valuable external knowledge (entity descriptions, etc.)

Sun, Zhiqing, et al. "Rotate: Knowledge graph embedding by relational rotation in complex space." *arXiv preprint arXiv:1902.10197* (2019).

HAKE (Hierarchy-Aware KGE)

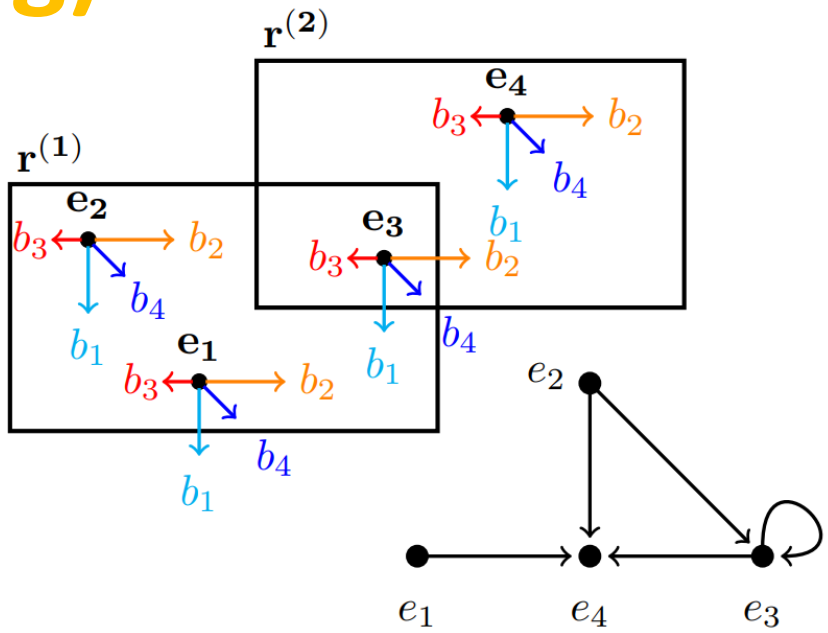
- Designed to model hierarchical entity structure
- HAKE works in a polar space with a two-part embedding:
  - phase: model entities on the same hierarchical level, similar to RotatE, capture symmetric or inverse relations
  - module: model entity level in the hierarchy (root nodes have smaller modules w.r.t. leaf nodes). This allow to distinguish between general and specific cocepts
- Given a triple (h,r,t) the scoring function perform:
  - scaling, with module embedding, $h_m \circ r_m \approx t_m$
  - rotation, with phase embedding, $h_p + r_p \approx t_p$



Zhang, Zhanqiu, et al. "Learning hierarchy-aware knowledge graph embeddings for link prediction." *AAAI* 2020

BoxE (Box Embedding)

- BoxE uses geometric spatial relations
  - entities are modelled as points (vectors) in $\mathbb{R}^n$
  - relations are modelled as two axis-aligned iper-rectangles, one for the head, one for the tail
  - A triple (h, r, t) is considered true if the point representing the head (h) falls into the head box of the relation $r^{(1)}$, and the point of the tail (t) falls into the tail box $r^{(2)}$



- This modelling allows:
  - **relation subsumption**: if r_1 is more specific than r_2 (e.g. *wears* $\sqsubseteq$ *possesses*), then hear box of r_1 is contained contain in that r_2, and the same holds for the tail boxes
  - **intersection**: overlapping boxes correspond to entities that satisfy multiple relational constraints simultaneously
  - **disjunction** and **composition** my be implicitly approximated in the embedding space

Abboud, Ralph, et al. "Boxe: A box embedding model for knowledge base completion." *Advances in neural information processing systems* 33 (2020): 9649-9661.

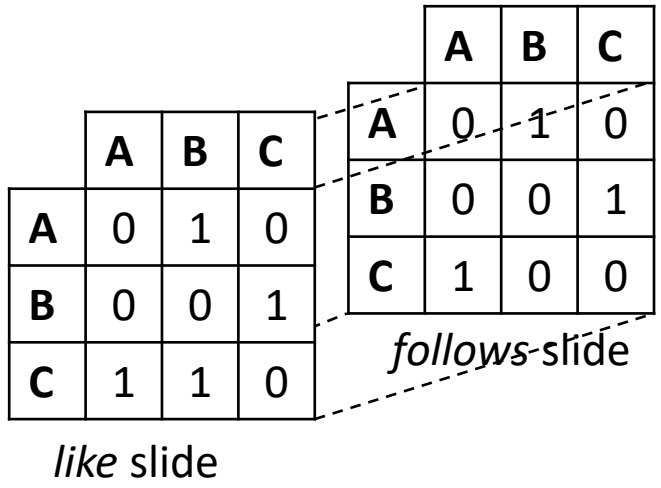
Tensor Decomposition Models

- Tensor decomposition (or factorization) models treat the KG as a large **3D binary tensor** ($E \times E \times R$), where each slice represents an adjacency matrix for a specific relation

KG as 3D tensor

Entities:
- Alice
- Bob
- Carol

Relation:
- likes
- follows



- These models factorize the tensor into **latent factor matrices** to capture rich, high-order interactions between entities and relations

RESCAL

- RESCAL is one of the earliest tensor-based models and captures interactions using **full relation matrices**
  - approximates the relation matrices X_k as product of entity embedding and relation matrices → $X_k = AR_kA^\top$
- This makes it highly expressive, as it can **model complex dependencies** between entity features
- However, this expressiveness comes with a **quadratic cost in the embedding dimension**, making the model difficult to scale

Nickel, Maximilian, Volker Tresp, and Hans-Peter Kriegel. "A three-way model for collective learning on multi-relational data." *Icml*. Vol. 11. No. 10.5555. 2011.

RESCAL

- The ultimate goal is Knowledge Graph Completion
- When you multiply the obtained matrices again to reconstruct the original tensor:
  - Cells that were originally 1 should remain close to 1 (confirmation of known facts)
  - Cells that were 0 **may now have a high value** (e.g. 0.85). These new values are interpreted as predictions of missing triple
- The system **discover a probable relationships** based on geometric similarities in the embedding space

DistMult

- DistMult simplifies RESCAL by **restricting relation matrices to be diagonal**, reducing the parameters per relation
- Limit: DistMult is mathematically limited to **symmetric relations**, meaning it cannot distinguish between (h, r, t) and (t, r, h):
  - The scoring function becomes a simplified bilinear product:

$$f_r(h, t) = h^\top diag(r) t$$

  - Since the scoring function is based on the **weighted scalar product** (which has the commutative property), the result of $f_r(h, t)$ will always be identical to that of $f_r(t, h)$

Yang, Bishan, et al. "Embedding entities and relations for learning and inference in knowledge bases." *arXiv preprint arXiv:1412.6575* (2014).

ComplEx

- ComplEx extends DistMult into the **complex vector space** to handle asymmetric and antisymmetric relations
  - Each entity and relationship is represented by a vector whose elements have a real part and an imaginary part
- By using the Hermitian dot product and conjugate transposition, ComplEx can **model directional links** while maintaining high computational efficiency:
  - scoring function is defined as the **real part** of the Hermitian product:

$$f_r(h,t) = Re(\langle h, r, \bar{t} \rangle)$$

    where $\bar{t}$ is the complex conjugate of the queue entity
  - **conjugation operation breaks the symmetry** of the product: by swapping heads and tails, the score changes as the conjugate is applied to a different vector
  - if a relationship is indeed symmetric, the model can simply zero out the imaginary part of the embeddings

Trouillon, Théo, et al. "Complex embeddings for simple link prediction." *International conference on machine learning*. PMLR, 2016.

HolE

- Alternative to ComplEx, HolE (Holographic Embeddings) utilizes circular correlation to combine entity embeddings
  - Circular correlation measures the similarity between two vectors considering all their possible rotations
  - In the scoring function:

$$f(h, r, t) = r^T (h \star t)$$

$h \star t$ captures interactions between all dimensions, and r weighs these interactions

- Circular correlation is not commutative: this allows the **model to distinguish the direction** of the relationship

Nickel, Maximilian, Lorenzo Rosasco, and Tomaso Poggio. "Holographic embeddings of knowledge graphs." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. No. 1. 2016.

Example of circular correlation

Given: $x = [1,2,3]^\top,\ y = [4,5,6]^\top$

Circular correlation is defined as: $(x \star y)_k = \sum_{i=o}^{n} x_i \cdot y_{i+k\ mod 3}$

Step 1 — k=0

$$(x \star y)_0 = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 4 + 10 + 18 = 32$$

Step 2 — k=1, now let's "rotate" y: $y_{(i+1)mod3} = [5,6,4]\text{^}\top$

$$(x \star y)_1 = 1 \cdot 5 + 2 \cdot 6 + 3 \cdot 4 = 5 + 12 + 12 = 29$$

Step 3 — k=2, next rotation: $y_{(i+2)mod3} = [6,4,5]^\top$

$$(x \star y)\_2 = 1 \cdot 6 + 2 \cdot 4 + 3 \cdot 5 = 6 + 8 + 15 = 29$$

Final result: $x \star y = [32, 29, 29]^\top$

TuckER

- TuckER is based on the **Tucker decomposition** of the binary adjacency tensor of the graph:
  - estimates a **shared core tensor** ($W \in \mathbb{R}^{d \times k \times d}$) and embedding vectors for relations and entities
  - scoring is evaluated with

$$f(h, r, t) = \sum_{i,j,l} W_{ijl} \cdot h_i \cdot r_j \cdot t_l = h^\top W_r t$$

where $W_r = \sum_j r_j W_{:,j,:}$

- The core tensor
  - capture interactions between latent dimensions
  - is shared among all relationships
  - allows parameter sharing
- TuckER is able to be **highly expressive** in modelling complex interactions between entities and relationships, while maintaining a **more compact and parameter-efficient representation**

h = [1, 2], r = [3, 4], t = [5, 6]

W slices:
W[:,0,:] = [[1,0],[0,1]]
W[:,1,:] = [[0,1],[1,0]]

Step 1: W_r = 3*[[1,0],[0,1]] + 4*[[0,1],[1,0]] = [[3,4],[4,3]]

Step 2: W_r t = [[3,4],[4,3]] [5,6] = [39, 38]

Step 3: score - f(h,r,t) = [1,2] · [39,38] = 115

Balažević, et al.. "Tucker: Tensor factorization for knowledge graph completion." EMNLP-IJCNLP. 2019.

Neural Matching Models

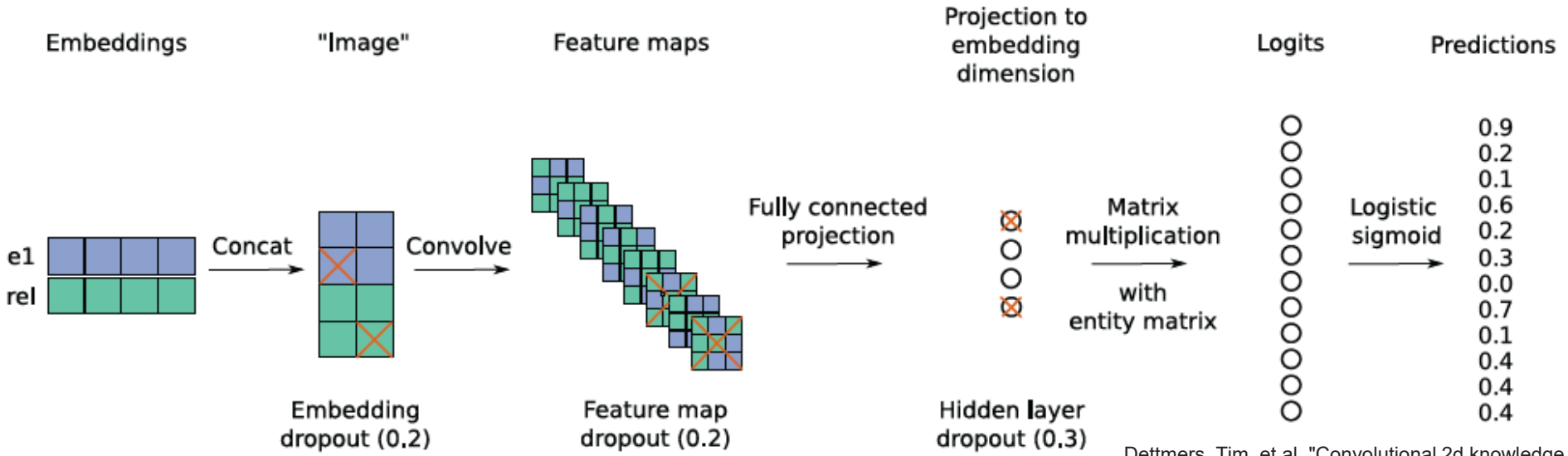
- Neural Matching Models extend KGE approaches beyond simple geometric or algebraic scoring
- These methods include
  - Semantic Matching Energy (SME)
  - Neural Tensor Networks (NTN)
- Respect to previous solutions, embeddings are still learned from data, but the learning is less structured:
  - scoring function is **fully parameterized neural architecture that** learns both feature interactions and non-linear transformations
- This allow to achieve **higher expressiveness**, and **less inductive bias**
- Limits:
  - require more data and careful tuning
  - poor scalability to large Knowledge Graphs
  - high computational complexity during training

Convolutional KGE

- NMM do not explicitly exploit the internal structure of embedding vectors
- In Convolutional Knowledge Graph Embeddings, embeddings are no longer processed as isolated vectors, but are **reshaped and treated as structured feature maps**
- Models such as **ConvE**, **ConvKB**, and **InteractE** introduce convolutional operations to:
  - capture local and global feature interactions
  - model higher-order dependencies between embedding dimensions
  - improve parameter efficiency while maintaining high expressiveness
- Relational reasoning is now learned through structured feature extraction mechanisms inspired by **convolutional neural networks**

ConvE

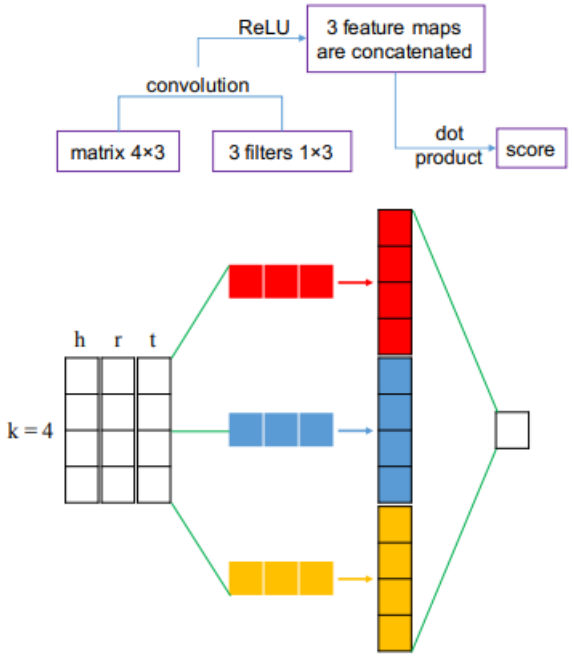
- **ConvE** was the first model to utilize **2D convolutional layers** to extract features from reshaped entity and relation embeddings
  - first encodes (h, r) into a query vector via convolutional feature extraction
  - then ranks all candidate tail entities by similarity (dot product) with this query representation.

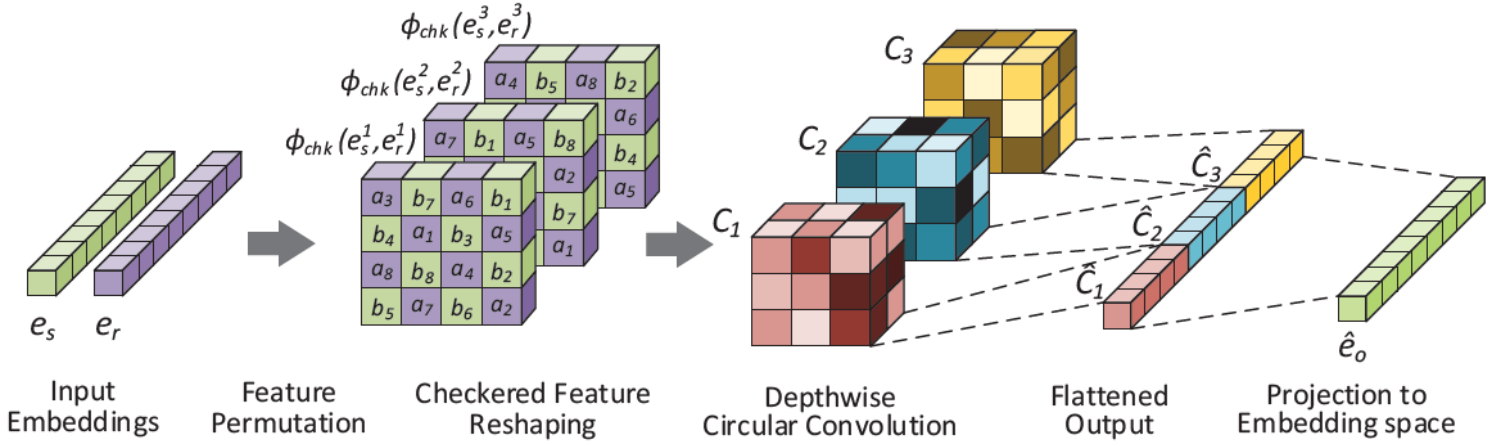


Dettmers, Tim, et al. "Convolutional 2d knowledge graph embeddings." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. No. 1. 2018.

ConvKB - InteractE

- **ConvKB** uses a 1D convolution to capture global relationships and translational characteristics without the reshaping operations of ConvE
- **InteractE** improves upon ConvE by increasing feature interactions through permutations and circular convolutions.

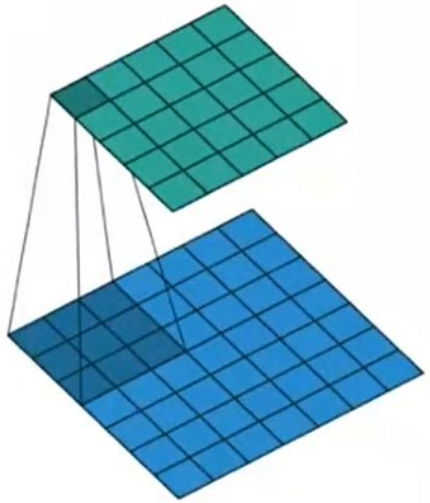


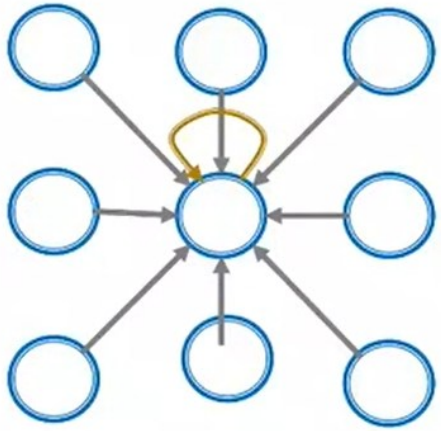


Nguyen et al. "A novel embedding model for knowledge base completion based on convolutional neural network." 2018.
Vashishth, Shikhar, et al. "Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions." 2020.

Graph Neural Networks

- GNNs are a family of **neural architectures** specifically designed to process data represented as collections of **nodes** and **edges**
- They generalize deep learning from Euclidean data (like images or text) to **non-Euclidean graph structures**
- The fundamental shift is from learning static embeddings to an **iterative process of information aggregation and propagation**

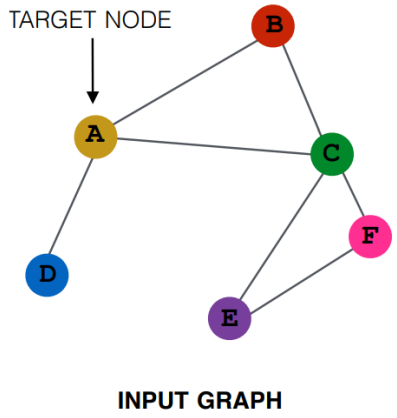




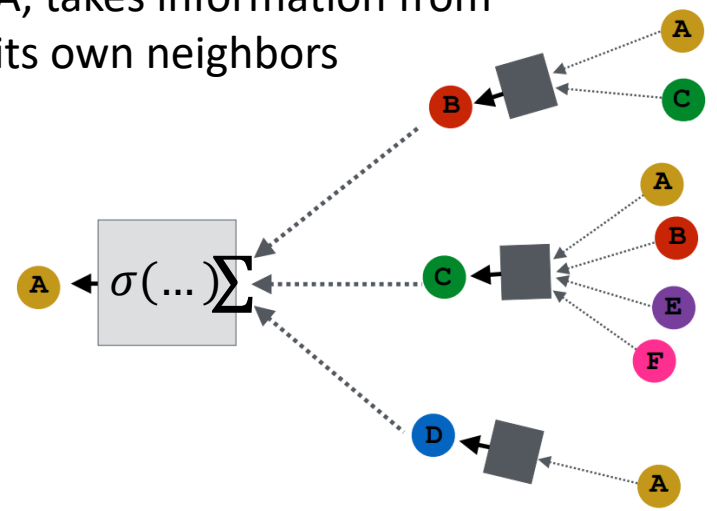
Message Passing Paradigm

- The core mechanism of a GNN is Message Passing, where each node **updates its representation by aggregating features from its immediate neighbours**.
- At each layer l, a
  - node u receives messages from its neighbours $N(u)$
  - combines them with its current state
  - passes the result through a non-linear activation function

The target node A takes information from its neighbors B, C, and D



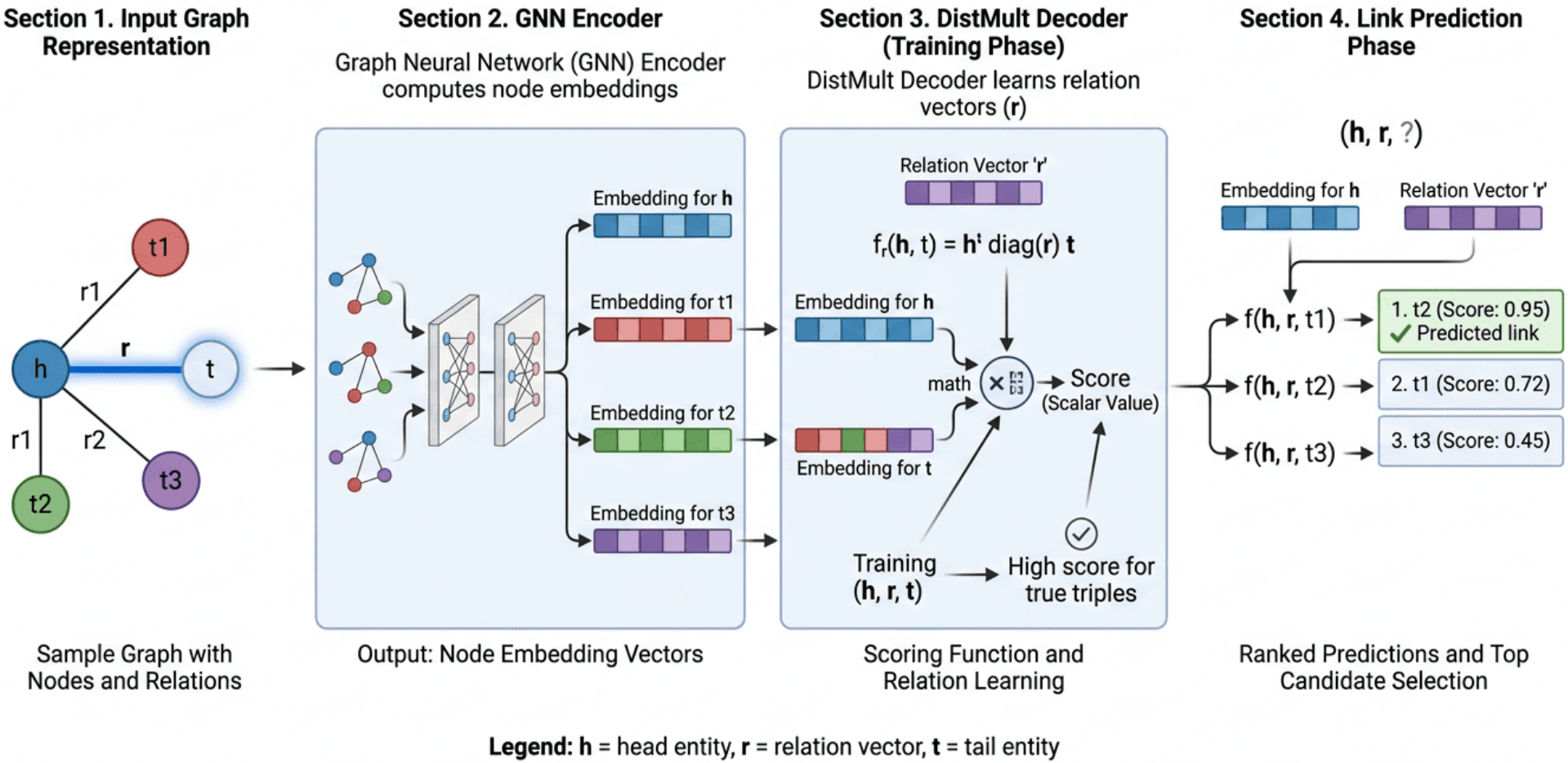
Similarly, each neighbor of A, takes information from its own neighbors



GNNs as Encoders and Decoders

- GNN-based KGC typically follows an **encoder-decoder** framework
- The **Encoder** (the GNN) maps each entity to a context-aware vector that incorporates neighbourhood information
- The **Decoder** (often a scoring function like DistMult, $f_r(h,t) = h^\top diag(r)t$) then reconstructs the edges of the graph based on these enriched representations
  - In practice, given the node embeddings, the decoder estimate the relation representation in order to reconstruct the input graph

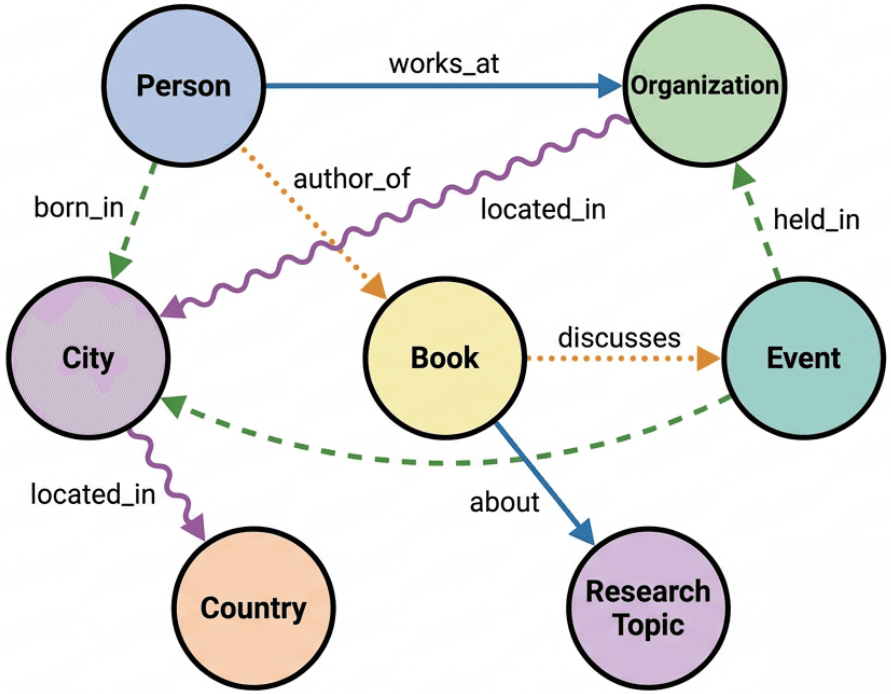
GNNs as Encoders and Decoders



Legend: **h** = head entity, **r** = relation vector, **t** = tail entity

Relational Graph Convolutional Networks (R-GCN)

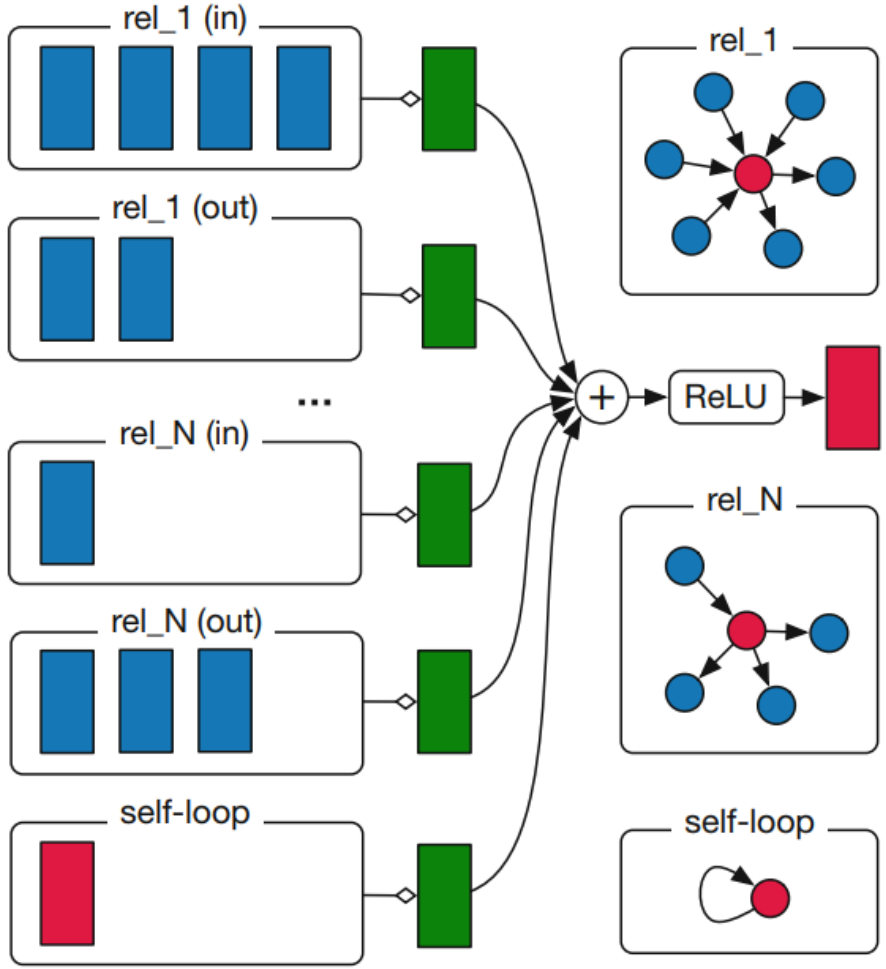
- Standard GCNs are limited to homogeneous graphs
- R-GCN was developed to handle the **multi-relational nature** of KGs.
  - A multi-relational graph is a graph in which **two nodes can be connected by multiple types of different edges** (relations), each with its own meaning
  - Unlike a simple graph — where there is at most one edge between two nodes — in a multi-relational graph the same node can have **completely different relationships with the other nodes**



Schlichtkrull, Michael, et al. "Modeling relational data with graph convolutional networks." *European semantic web conference*. Cham: Springer International Publishing, 2018.

Relational Graph Convolutional Networks (R-GCN)

- R-GCN **aggregates relation-specific transformation** matrices for each neighbour
- This ensures that different types of edges (e.g., *bornIn* vs. *founded*) contribute differently to the update
- Obtained the embeddings, DistMul can be used as decoder



Schlichtkrull, Michael, et al. "Modeling relational data with graph convolutional networks." *European semantic web conference*. Cham: Springer International Publishing, 2018.

CompGCN

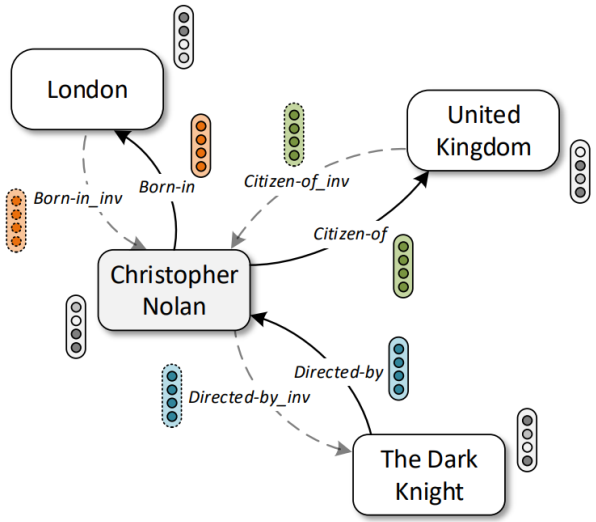
- CompGCN advances the relational convolution by **jointly learning representations for both nodes and relations**
- Node embedding update:

$$\boldsymbol{h}_v^{k+1} = f\left(\sum_{(u,r)\in\mathcal{N}(v)} \boldsymbol{W}_{\lambda(r)}^k \phi(\boldsymbol{h}_u^k, \boldsymbol{h}_r^k)\right).$$

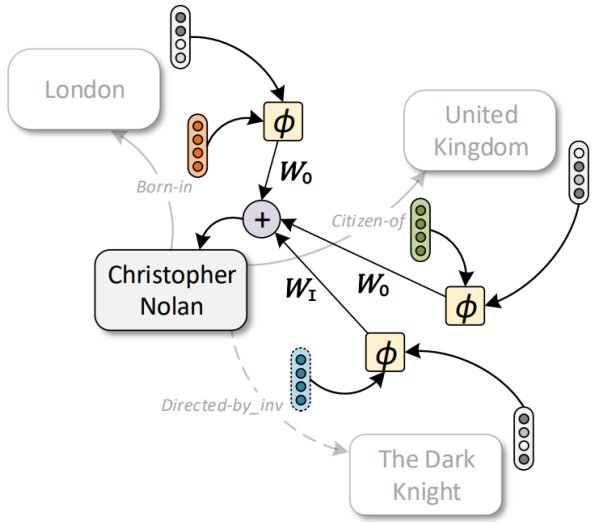
- Relation embedding update:

$$\boldsymbol{h}_r^{k+1} = \boldsymbol{W}_{\text{rel}}^k \, \boldsymbol{h}_r^k.$$

- This approach
  - preserves relational information across layers
  - provides native support for inverse relations



Relational Graph with Embeddings



CompGCN Update

Vashishth, Shikhar, et al. "Composition-based multi-relational graph convolutional networks." *arXiv preprint arXiv:1911.03082* (2019).

CompGCN

- The ϕ function is used to combine the embedding of a **neighbouring entity** with the **relation embedding** that connects it to the central node (hv) during the message update phase

$$\boldsymbol{h}_v^{k+1} = f\left(\sum_{(u,r)\in\mathcal{N}(v)} \boldsymbol{W}_{\lambda(r)}^k \phi(\boldsymbol{h}_u^k, \boldsymbol{h}_r^k)\right).$$

- CompGCN is flexible and allows you to choose ϕ from several:
  - **Subtraction** ($h_u - h_r$), inspired by TransE, it models the relationship as a translation
  - **Element-by-element multiplication** ($h_u \circ h_r$), inspired by DistMult
  - **Circular correlation** ($h_u \star h_r$), inspired by HolE

- ϕ allows CompGCN to integrate **the power of algebraic models** within the **GNN message-passing** architecture

Vashishth, Shikhar, et al. "Composition-based multi-relational graph convolutional networks." *arXiv preprint arXiv:1911.03082* (2019).

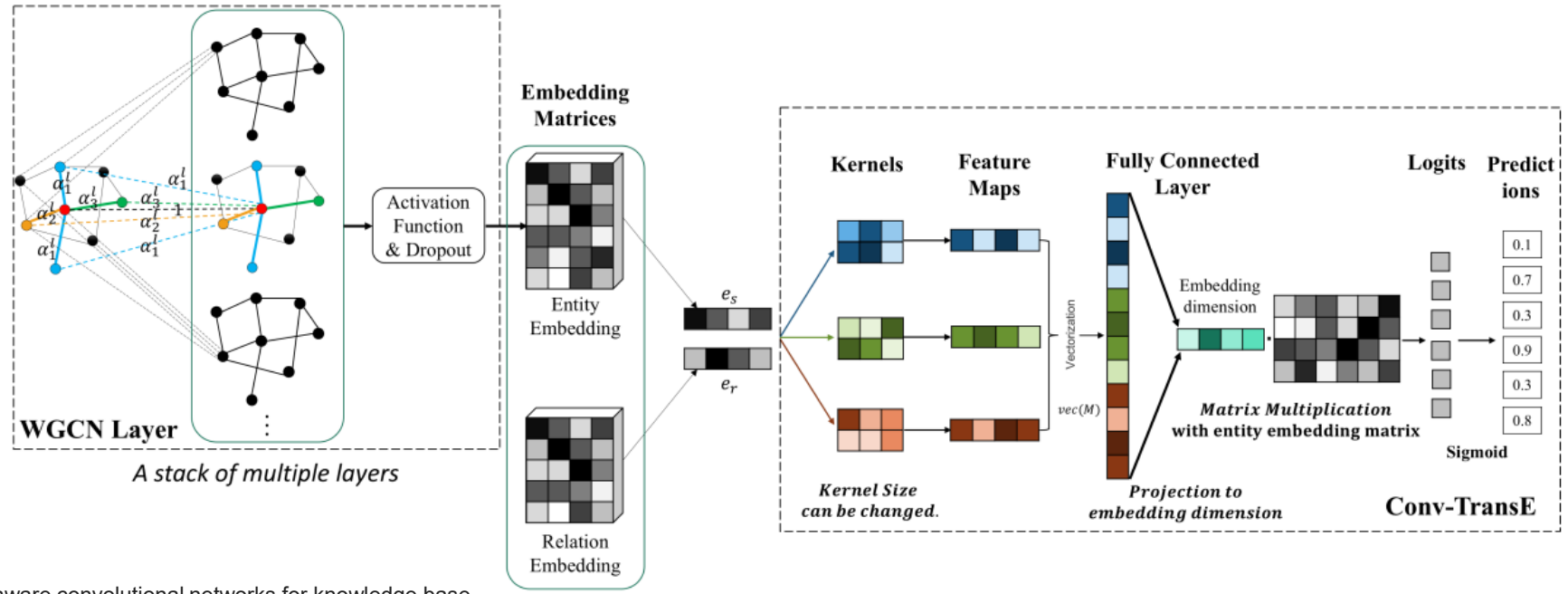
CompGCN

- CompGCN advantages:
  - **Direction-awareness**: Since many of the operations chosen for ϕ are not commutative, the model natively distinguishes between head and tail direction
  - **Support for inverse relationships**: ϕ allows you to correctly manage inverse relationships (r−1) by integrating their semantics into the aggregation
  - **Joint Learning**: Allows you to learn representations of nodes and relationships jointly, preserving relational information across all layers of the network, rather than limiting it to the decoder alone
  - **Efficiency**: It helps to mitigate the problem of over-parameterization typical of R-GCN, since it allows to share embeddings of the relationships between the various layers or to use decompositions into bases

Vashishth, Shikhar, et al. "Composition-based multi-relational graph convolutional networks." *arXiv preprint arXiv:1911.03082* (2019).

Structure-Aware Convolutional Networks (SACN)

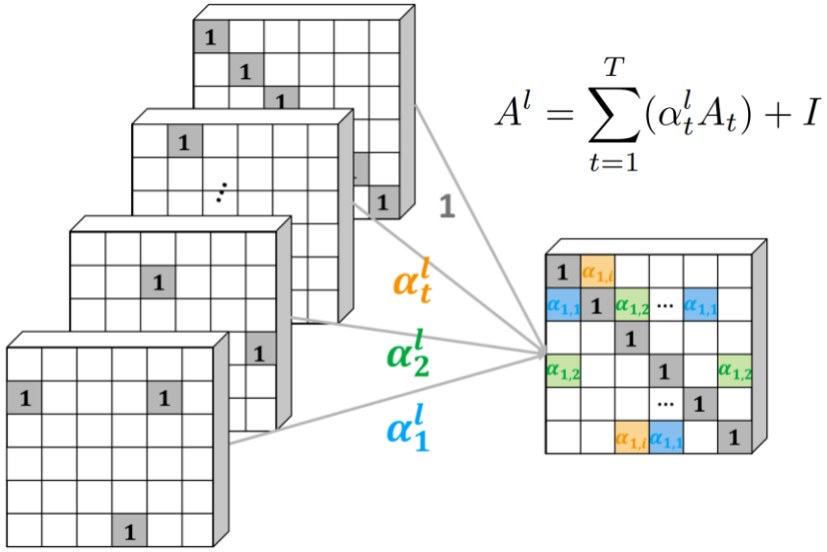
- SACN combines the strengths of GNNs (WGCN) and convolutional KGE models (ConvE)
- It uses a Weighted GCN as an encoder to aggregate structural information and entity attributes, followed by Conv-TransE as a decoder



Shang, Chao, et al. "End-to-end structure-aware convolutional networks for knowledge base completion." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. No. 01. 2019.

WGCN in SACN

- WGCN is an evolution of standard GCNs:
  - weighs differently different types of relations when aggregating
  - weights are adaptively learned during the training of the network
- WGCN treats a multi-relational KB graph as **multiple single-relational subgraphs** where each subgraph entails a specific type of relations



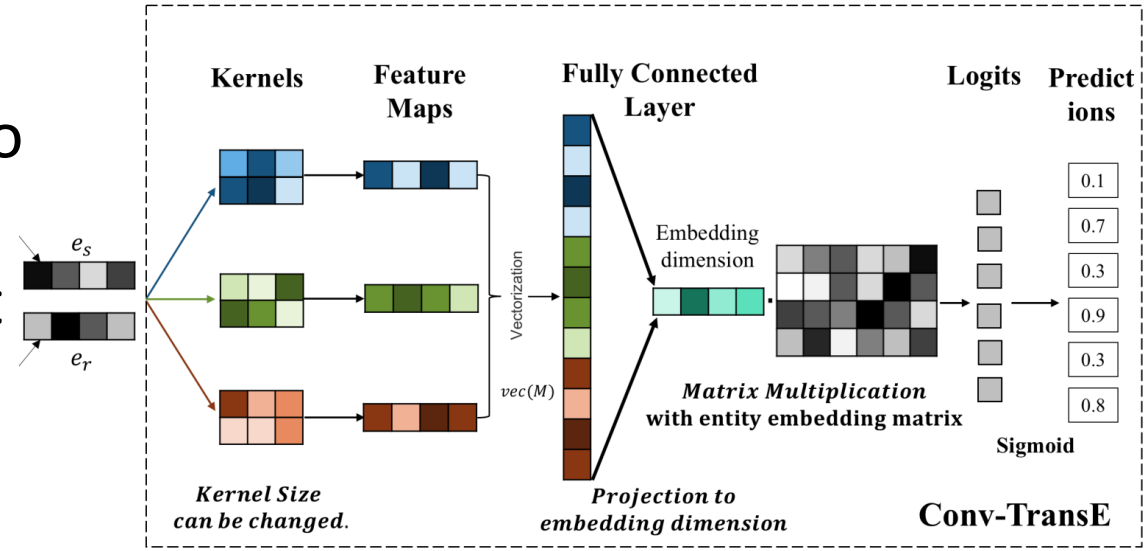
$$h_i^{l+1} = \sigma\left(\sum_{j \in \mathbf{N_i}} \alpha_t^l g(h_i^l, h_j^l)\right)$$

α_t^l is the weight for relation t in layer l

Shang, Chao, et al. "End-to-end structure-aware convolutional networks for knowledge base completion." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. No. 01. 2019.

SACN Decoder

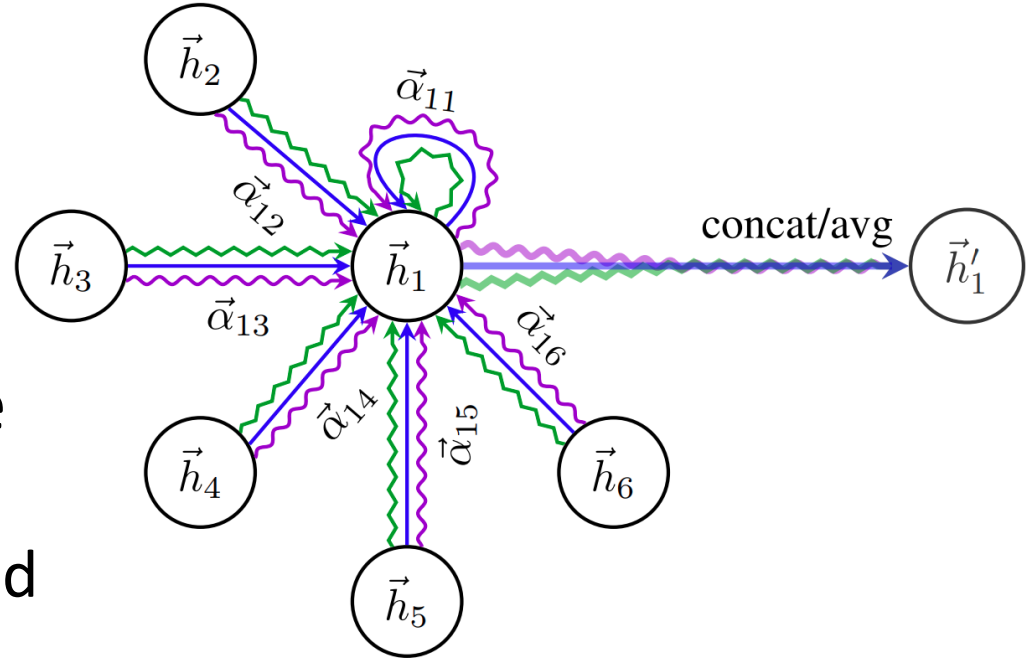
- **Convolutional integration**: Conv-TransE combines head and relation embedding vectors and processes them through **2D convolutional layers** (similar to ConvE) to extract feature maps
- Preservation of translational properties: Unlike ConvE, Conv-TransE is designed to preserve translational characteristics (inherited from TransE), making it easier to predict missing links:
  - In ConvE, the head (h) and relation (r) vectors are "folded" over multiple rows to form a 2D matrix before being concatenated breaking natural alignment
  - Conv-TransE avoids this complex reshaping



Shang, Chao, et al. "End-to-end structure-aware convolutional networks for knowledge base completion." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. No. 01. 2019.

Graph Attention Networks in KGs

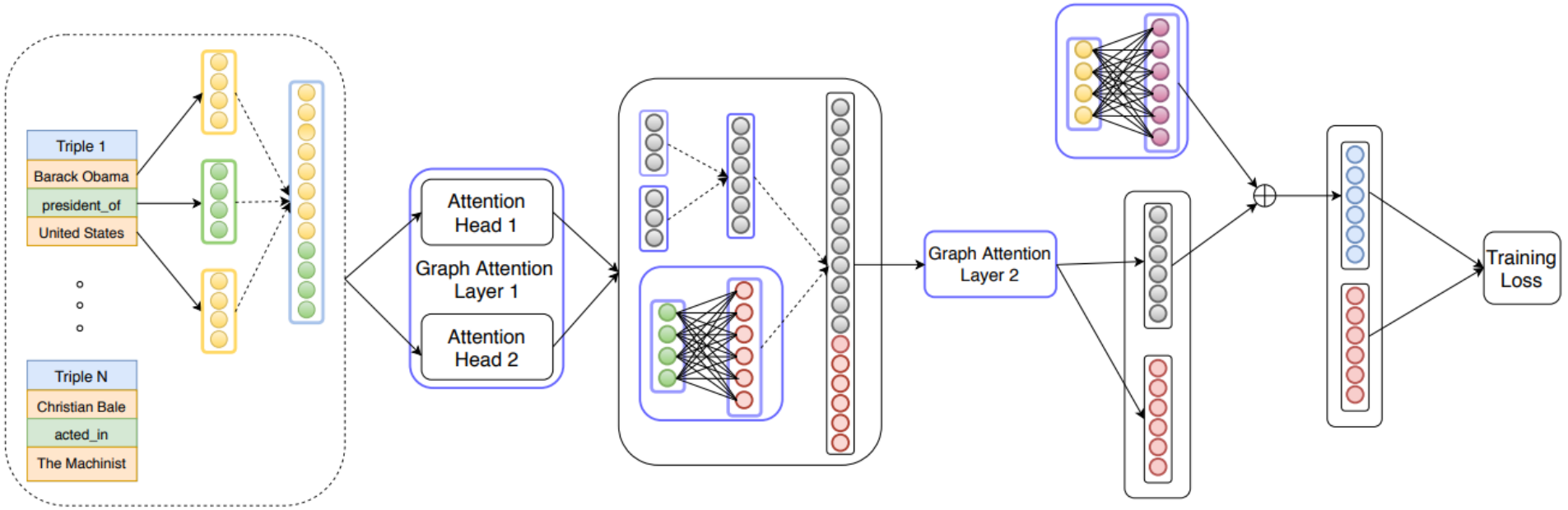
- Unlike GCNs, where all neighbours contribute equally, GATs assign **different levels of importance** (weights) to each neighbour using an attention mechanism
- Attention coefficients α_{ij} allow the model to **focus on the most relevant features** in a neighbourhood
- Standard GATs are node-centric, but relational variants (R-GAT) parameterize attention based on relation types
  - α_{ij} depends on node i and j and the related edge e_{ij}



https://commons.wikimedia.org/wiki/File:Graph_attention_network.png

KBGAT

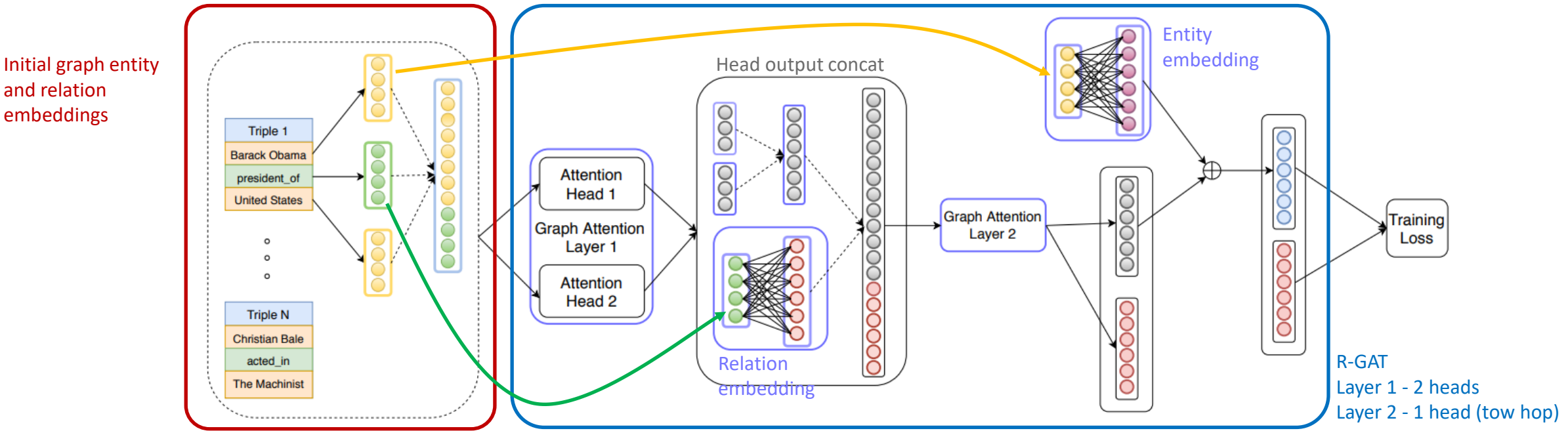
- KBGAT (Knowledge Base Graph Attention Network) proposes a multi-layer attentional network to encode the importance of neighbours relative to a target triple
- It specifically targets the multi-hop neighbourhood, concatenating entity and relation embeddings to calculate precise attention values
- This results in high-fidelity representations of both entity and relationship semantics



Nathani, Deepak, et al. "Learning attention-based embeddings for relation prediction in knowledge graphs." *Proceedings of the 57th annual meeting of the association for computational linguistics*. 2019.

KBGAT

- KBGAT (Knowledge Base Graph Attention Network) proposes a multi-layer attentional network to encode the importance of neighbours relative to a target triple
- It specifically targets the **multi-hop neighbourhood**, concatenating entity and relation embeddings to calculate precise attention values
- This results in high-fidelity representations of both entity and relationship semantics



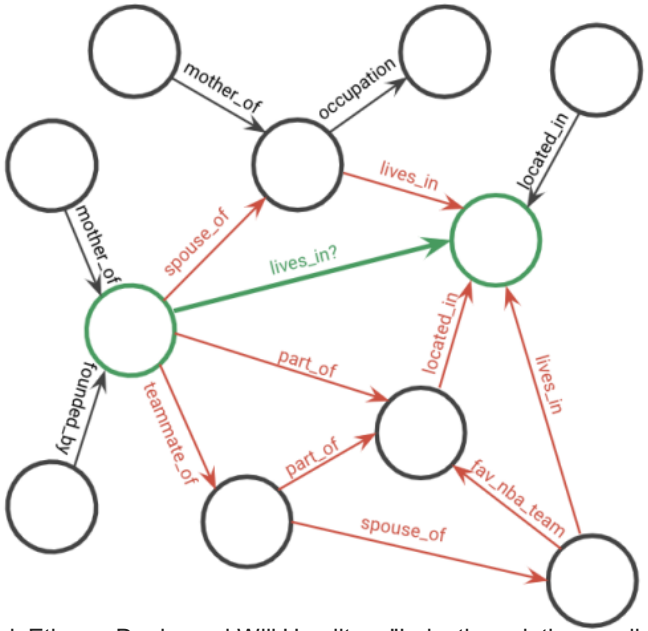
Nathani, Deepak, et al. "Learning attention-based embeddings for relation prediction in knowledge graphs." *Proceedings of the 57th annual meeting of the association for computational linguistics*. 2019.

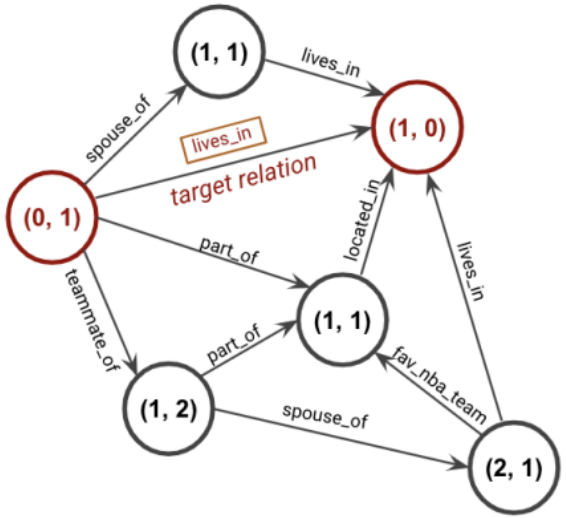
Inductive vs. Transductive GNNs

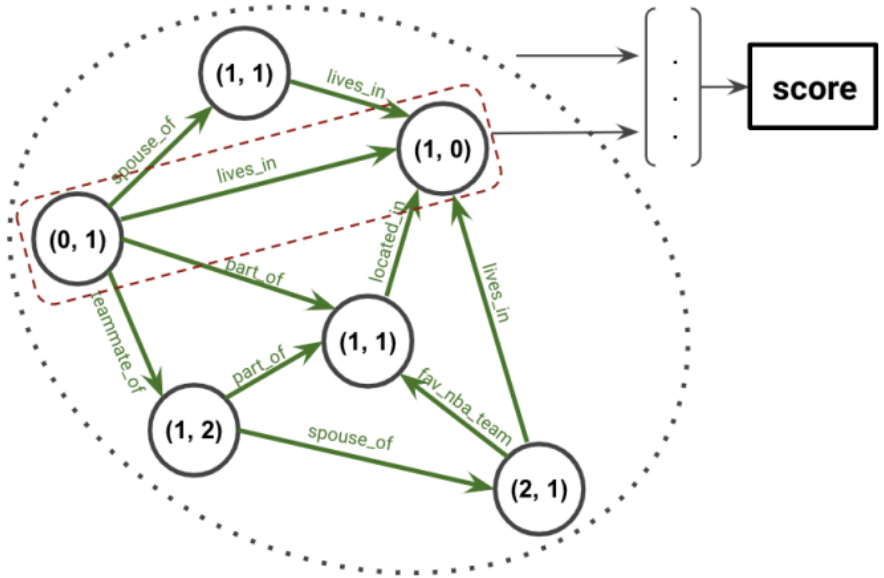
- Most KGE models are **transductive**, meaning they can only reason about **entities seen** during training
  - in KG entities are usually represented with discrete IDs without intrinsics semantic features
  - usually, to compute embeddings, IDs are associate to the nodes, thus their embedding depends on the node ID
- To solve this, **inductive** GNNs like GraIL sample local neighborhoods to generate embeddings for **previously unseen entities**, without using ID-based embeddings
- This is critical for open-world KGs where new entities are frequently added without the possibility of full retraining

GraIL

- GraIL (Graph Inductive Link Prediction) reasons by **extracting a subgraph** around a candidate edge rather than learning global node IDs
- It assigns structural labels to nodes based on their distance from target entities, allowing the GNN to learn **generalizable relational rules**
- This enables the model to predict links in entirely new graphs with different entity sets



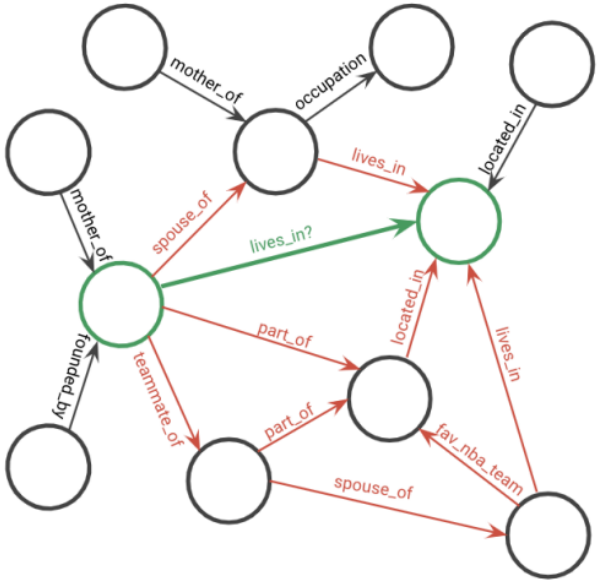


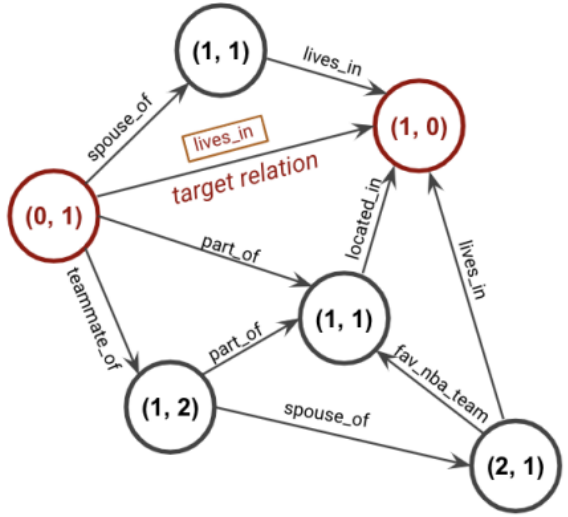


Teru, Komal, Etienne Denis, and Will Hamilton. "Inductive relation prediction by subgraph reasoning." *International conference on machine learning*. PMLR, 2020.

GraIL

1. Dataset: For every positive triple (h,r,t) GraIL constructs the induced subgraph (nodes within k-hop of h and t)
2. Sub-graph node distance encodings
3. Sub-graph embedding with GNN
4. Score: $score(h, r, t) = f(GNN(subgraph), r)$



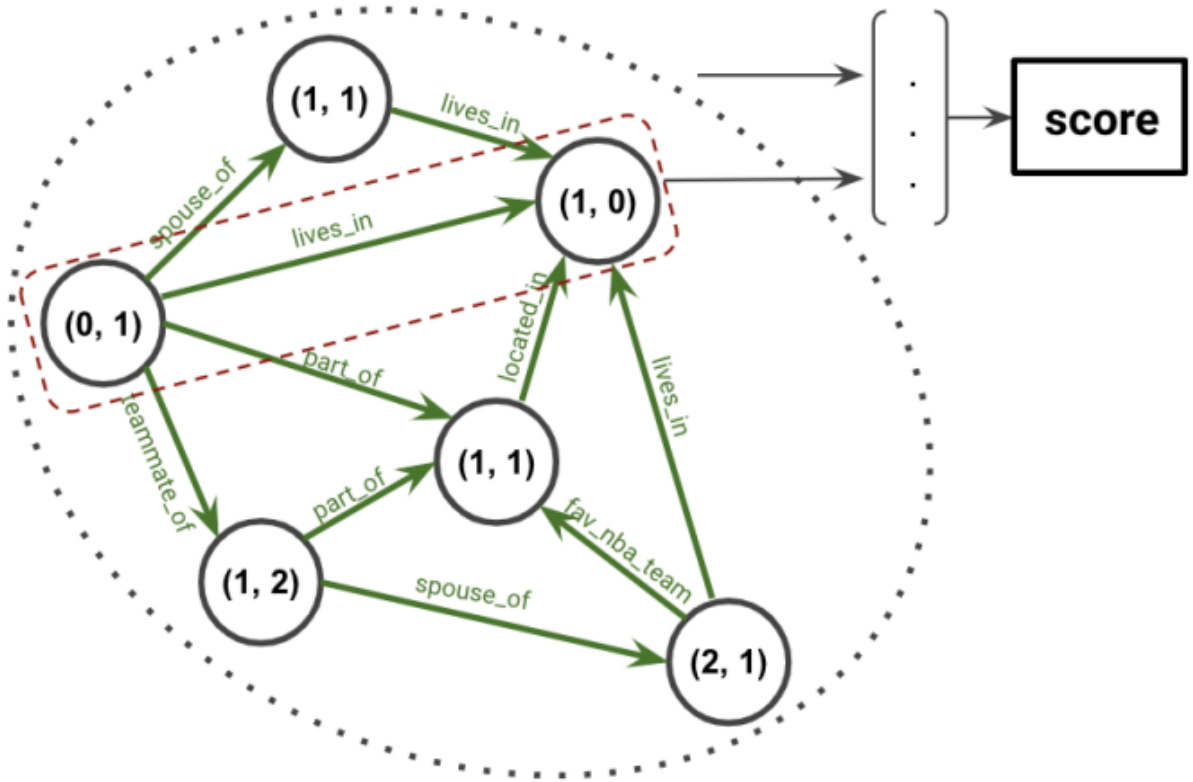


Teru, Komal, Etienne Denis, and Will Hamilton. "Inductive relation prediction by subgraph reasoning." *International conference on machine learning*. PMLR, 2020.

GraIL

For a query $(h, r, ?)$:

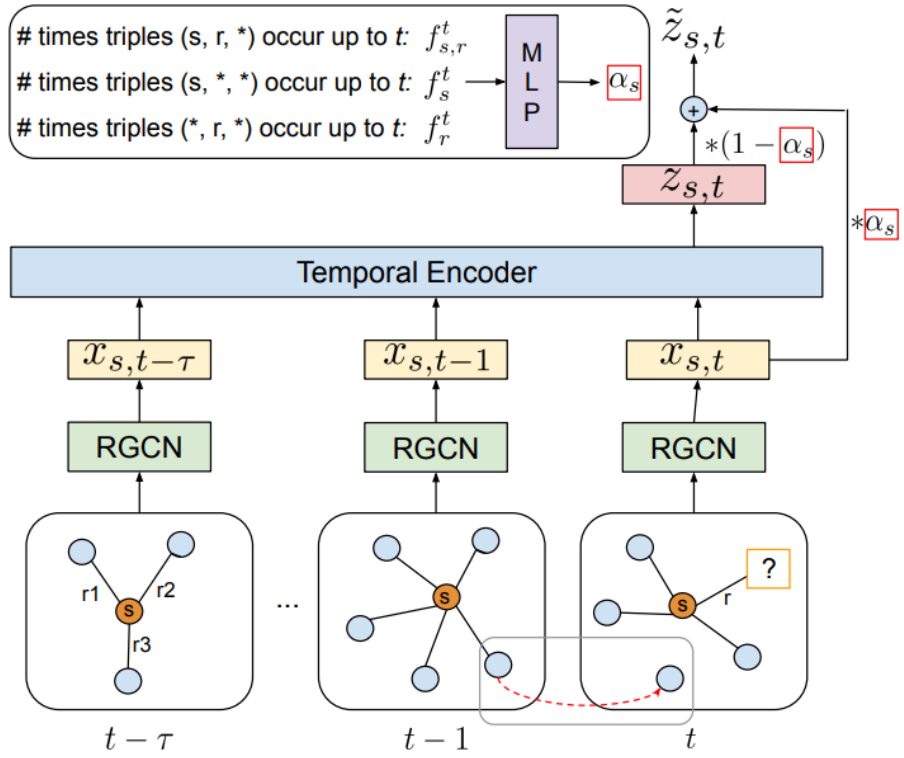
- for each candidate t:
  - constructs subgraph around (h, t)
  - passes into GNN
  - gets scores
  - sort candidates



Teru, Komal, Etienne Denis, and Will Hamilton. "Inductive relation prediction by subgraph reasoning." *International conference on machine learning*. PMLR, 2020.

Temporal GNNs for Dynamic KGs

- Real-world knowledge evolves
- Temporal GNNs (e.g., TeMP) condition neighbourhood aggregation on time information
- TeMP uses
  - Structural enc: R-GCN
  - Temporal enc:
    - TeMP-GRU: GRU + temporal decay, or
    - TeMP-SelfAttention: transformer-like
  - Imputation of inactive entities
  - Frequency-based gating mechanisms to weight temporal and structural embeddings



Query: $(s, r, ?, t)$

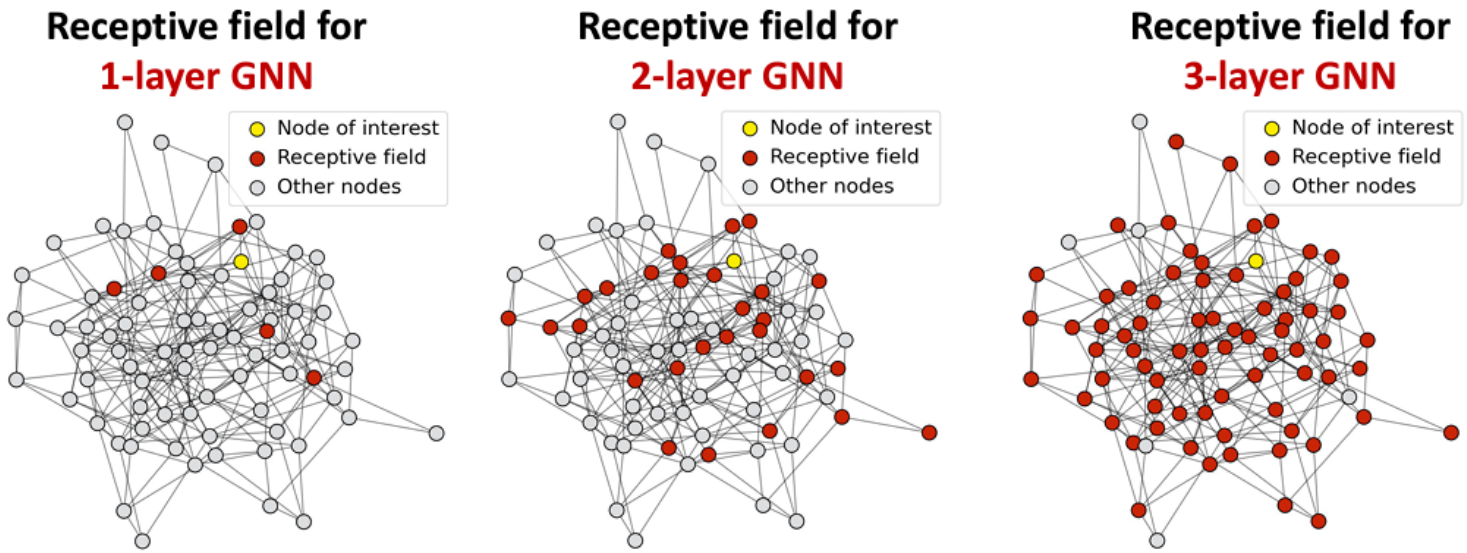
Wu, Jiapeng, et al. "Temp: Temporal message passing for temporal knowledge graph completion." *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2020.

High-Dimensional Scaling and GNN Challenges

- As graphs scale to billions of nodes, memory and communication overheads become bottlenecks
- The **neighbour explosion problem** happens when staking layers on densely connected graphs:
  - **Neighbour sampling** avoids the explosion by selecting only a subset of neighbours
  - Neighbourhood-based **mini-batching** to create batches that consist of a set of target nodes along with their sampled subgraphs
- Despite these efforts, the high computational complexity of GNNs remains a hurdle for industrial applications

Oversmoothing in GNNs

- **Oversmoothing**: As GNN layers increase, node representations tend to converge to the same value, making them indistinguishable
- This structural bottleneck limits the **receptive field** and the **expressiveness** of GNNs, often restricting peak performance to only two or three layers



Advanced Neuro-Symbolic Reasoning

The Need for Synergy

- Pure **symbolic reasoning** is interpretable but fails to scale and cannot handle the noisy, incomplete nature of real-world data
- **Representation learning** for KGE is more robust but is typically less interpretable
- The bridge to **Advanced Neuro-Symbolic Reasoning** is found in architectures that integrate the **learning power of neural networks** with the **formal structure of logic**

Advanced Neuro-Symbolic Reasoning

- **Grounding Logic into Tensors**: Frameworks like Logic Tensor Networks (LTN) bridge this gap by grounding first-order logic symbols into neural computational graphs
  - transform **discrete logical constraints** into a **continuous loss function** that guides the neural model to learn representations that are globally consistent with symbolic knowledge
- **Neural Architectures as Logic Programs**: Systems like Logical Neural Networks (LNN) go a step further by creating a direct 1-to-1 correspondence between neurons and logical formula elements
- **From Data to Interpretable Rules**: with Rule Learning, instead of just predicting links, the system discovers generalizable Horn clauses (e.g., $h(R) \leftarrow b_1 \wedge \cdots \wedge b_n$)
  - This allows the system to not only complete the graph but to reveal the explicit "logic" of the domain, providing both a predictive engine and a human-readable explanation

Logic Tensor Network & Real Logic

- **Real Logic** serves as the foundation for **Logic Tensor Networks** (LTN), providing a **fully differentiable first-order logic language**
  - It allows for the specification of relational knowledge using constants, variables, predicates, and functional symbols
- Unlike classical logic, Real Logic adopts **infinitely many truth values** in the interval, enabling the handling of fuzzy concepts
- Moreover, Real Logic is **fully differentiable**, which enables the elements of its signature to be grounded onto data using neural computational graphs

Real Logic

- Given a triple (h, r, t)
  - with classical logic is *true* (1) or *false* (0)
  - with real logic truth $\in [0, 1]$:
    - $1.0 \rightarrow$ absolutely true
    - $0.8 \rightarrow$ likely true
    - $0.2 \rightarrow$ unlikely

- Example:

$$(Obama, bornIn, USA) \rightarrow bornIn(Obama, USA) = 0.95$$
$$(Obama, livesIn, Florence) \rightarrow livesIn(Obama, Florence) = 0.2$$

Formal Syntax and Typing System

- Real Logic is defined on a first-order language $\mathcal{L}$ with a signature containing a set of
  - **constant** (objects) like $Obama, USA$
  - **function** like $fatherOf(x) \rightarrow$ return an object
  - **relation** (predicates) like $livesIn(x, y), worksAt(x, y) \rightarrow$ return a truth value
  - **variable** symbols, like $x, y, \ldots$
- Objects, functions and predicates can belong to **different types**, defined by a non-empty set of domains D.
  - e.g., $D = \{Person, City, Company\} \rightarrow Obama \in Person, Rome \in City, \ldots$
  - function and predicates are associated to types according to their input, output or arguments

Definitions

- Three **Domain Mapping** functions are defined:
  - $D : X \cup C \rightarrow D$ returns the domain of a **variable** x or a **constant** c
  - $D_{in} : F \cup P \rightarrow D^*$ returns the domains of the arguments of a **function** f or a **predicate** p
  - $D_{out} : F \rightarrow D$ returns the range of a **function** f

- Mappings ensures that expressions like $lives\_in(Alice, Rome)$ are **well-formed** only if the arguments match the predicate's defined input domains

Definitions

- A **term** is defined as
  - an object or constant t in domain D(t)
  - a sequence (tuple) of terms
  - the results (application) of a function
- **Formulas**:
  - atomic:
    - equivalence,
    - application of a predicate
  - composite:
    - logic connection ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$)
    - quantification ($\forall$, $\exists$)

Let ***Town*** denote the domain of towns in the world and ***People*** denote the domain of living people.

Suppose that $\mathcal{L}$ contains the **constant symbols**
- *Alice*, *Bob* and *Charlie* of domain *People*,
- *Rome* and *Seoul* of domain *Town*

Let
- *x* be a variable of domain *People*
- *u* be a variable of domain *Town*.

The **term** *x*,*u* (i.e. the sequence *x* followed by *u*) has domain *People*,*Town* which denotes the Cartesian product between *People* and *Town* (*People*×*Town*).
- *Alice,Rome* is interpreted as an element of the domain *People,Town*.

Let *lives_in* be a ***predicate*** with input domain D_{in}(*lives_in*) = *People,Town*.
- *lives_in(Alice,Rome)* is a **well-formed expression**, whereas *lives_in(Bob, Charlie)* is not.

Badreddine, Samy, et al. "Logic tensor networks." *Artificial Intelligence*, 2022

Grounding in Real Logic

- Connects the **symbolic level** of logic with the **numerical level** (vectors, neural networks) to obtain
  - Continuous representations
  - Learning from data
  - Noise tolerance
- Formally, **grounding is a function $G(\cdot)$** that maps
  - terms into tensors: $G(t) \in \mathbb{R}^n$
  - predicates to functions: $G(p) \in (\mathbb{R}^n)^k \rightarrow [0,1]$
  - formulas to numbers: $G(\varphi) \in [0,1]$
  - variables to sequences: $G(x) = \{d_1, \ldots, d_k\}, d_i \in \mathbb{R}^n$

Grounding in Real Logic

- Connects the symbolic level of logic with the numerical level (vectors, neural networks) to obtain
  - Continuous representations
  - Learning from data
  - Noise tolerance
- Formally, grounding is a function $G(\cdot)$ that maps
  - terms into tensors: $G(t) \in \mathbb{R}^n$
  - predicates to functions: $G(p) \in (\mathbb{R}^n)^k \rightarrow [0,1]$
  - formulas to numbers: $G(\varphi) \in [0,1]$
  - **variables to sequences**: $G(x) = \{d_1, \ldots, d_k\}, d_i \in \mathbb{R}^n$

- Assigns variables to **multiple examples** simultaneously
- This allow the system to compute logical formulas over **entire batches of data** using efficient tensor operations

Grounding in Real Logic

- Formula:

$$\varphi\colon\ livesIn(Obama, USA)$$

- Grounding:
  - $G(Obama) = \mathbf{v}_O$
  - $G(USA) = \mathbf{v}_U$
  - $G(livesIn) = f_{NN}$
- Evaluation:

$$G(\varphi) = f_{NN}(\mathbf{v}_O, \mathbf{v}_U) = 0.92$$

Grounding logical operator

Logic connectives are interpreted using **fuzzy semantics** like

- Negation:

$$\neg A = 1 - A$$

- Conjunction:

$$A \wedge B = A \cdot B$$

- Disjunction:

$$A \vee B = A + B - AB$$

- Logic becomes **differentiable**!

Grounding of quantifiers

Quantifiers are interpreted as **aggregator**

- Universal:

$$\forall x\ \varphi(x) \approx \text{aggregation (e. g. mean)}$$

- Existential:

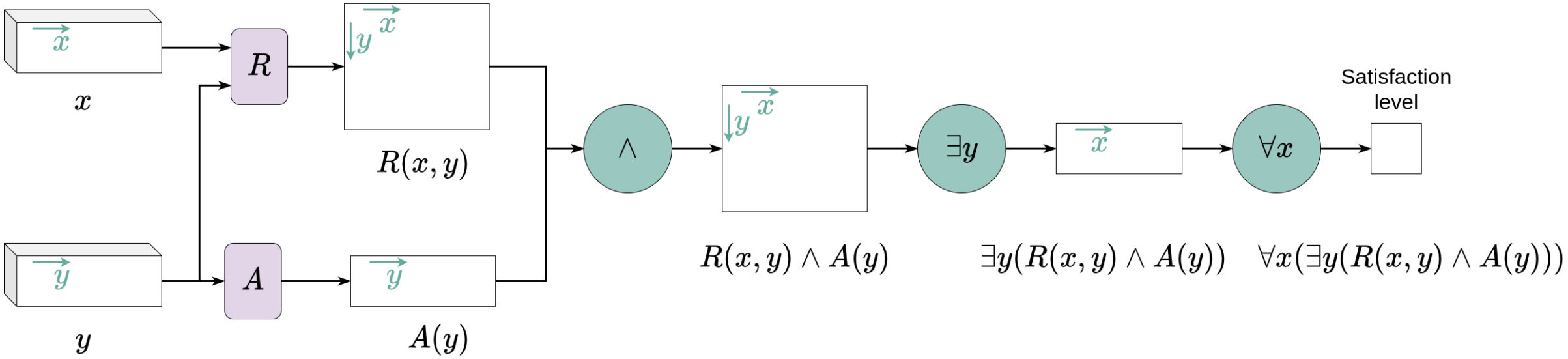
$$\exists x\ \varphi(x) \approx \max$$

Grounding in practice

- Constants are mapped to **point embeddings** in a latent space, representing specific entities like "*Dante*" or "*Firenze*".
- Domains are interpreted as **subsets of real-valued vector spaces**, such as the unit hypercube or n-simplex.
- Predicates are grounded as **neural networks** that map input vectors to a truth value in [0, 1]
- Functions are grounded as **linear transformations or MLPs** that map entities between domains
- This parametric definition allows the system to learn the meaning of symbols through **standard backpropagation**:
  - Fuzzy semantics transform discrete logical evaluations into a continuous surface suitable for gradient-based optimization.

LTN Example

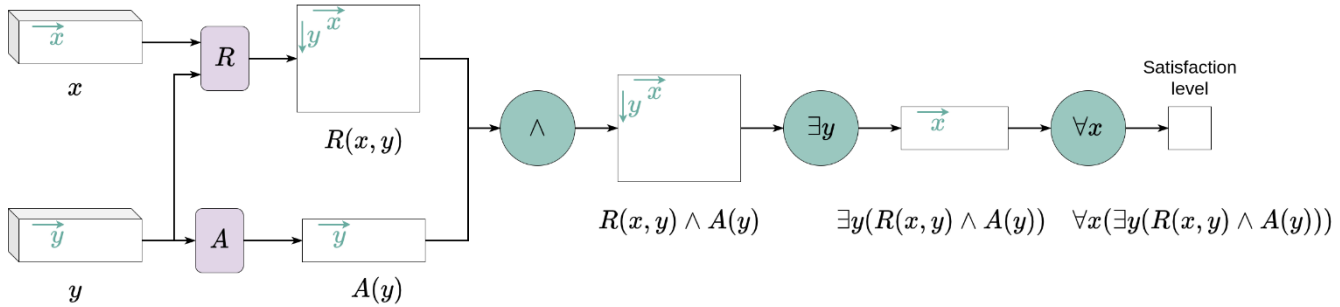
- Example: Everybody has a friend that is Italian
  - R is predicate *friendsOf()*
  - A is predicate *isItalian()*

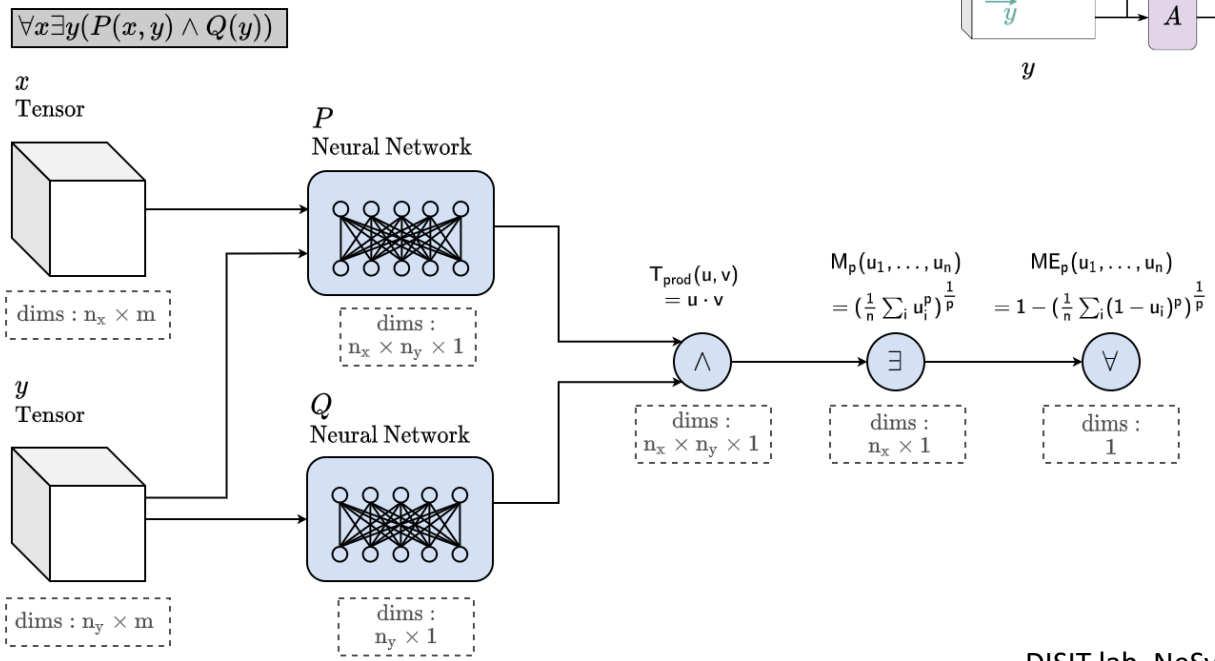


https://github.com/logictensornetworks/logictensornetworks

LTN Example

- Example: Everybody has a friend that is Italian
  - R is predicate *friendsOf()*
  - A is predicate *isItalian()*

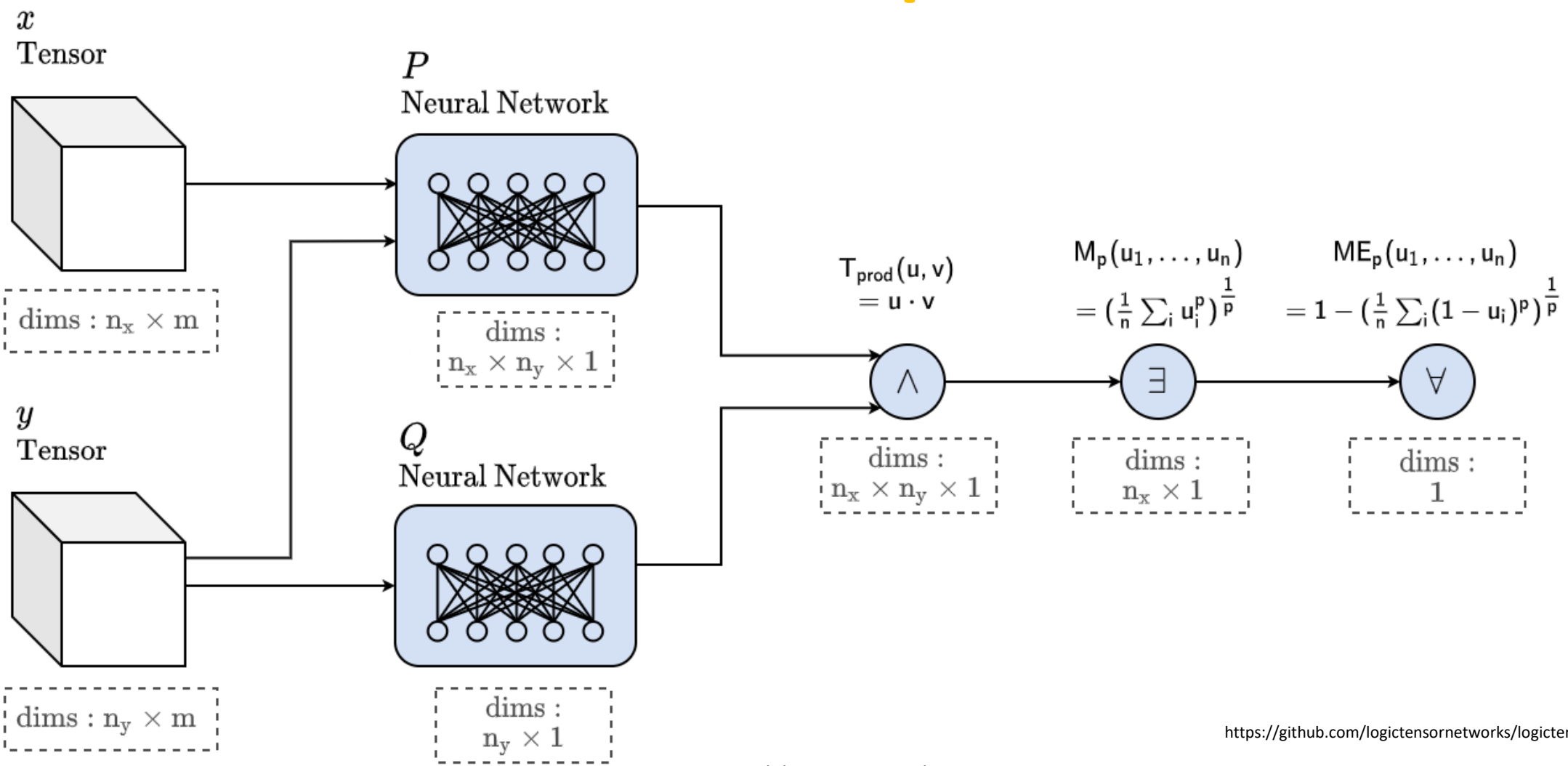




https://github.com/logictensornetworks/logictensornetworks

LTN Example

$\forall x \exists y (P(x,y) \wedge Q(y))$



https://github.com/logictensornetworks/logictensornetworks

Integrating Logic into the Loss Function

- Logical knowledge is integrated as a **set of constraints** that must be satisfied during training
- The satisfiability of a theory is aggregated into a single scalar value that the optimizer seeks to maximize
- This transforms logical reasoning into a **differentiable objective**, where the model learns to **align its neural predictions with symbolic rules**

Fuzzy Loss

- Fuzzy loss measures the degree to which a neural network **violates logical constraints and facts** provided by an expert or get from a KG
  - given **facts** (e.g. KG triples) and **rules** (e.g. ontological constraints), the loss function penalizes the model if it fails to make the facts "true" or if it violates the rules
  - It serves as a regularizer that pushes the network towards predictions that are **globally consistent** across the entire Knowledge Graph
- This approach is particularly effective when labelled data is scarce, as symbolic rules provide a **strong inductive bias**

Enforcing Subsumption Axioms

- Subsumption is modelled as a logical implication
  - e.g. *every dog is an animal*

$$\forall\, x\, \big(Dog(x) \rightarrow\ Animal(x)\big)$$

- Fuzzy loss penalizes instances where the model predicts a high truth value for the subclass but a low value for the parent class.
- This ensures that the learned embedding space respects the hierarchical structure of the domain ontology.

Enforcing Subsumption Axioms

- Subsumption is modelled as a logic
  - e.g. *every dog is an animal*

$$\forall x \left(Dog(x) \rightarrow Ai\right.$$

- Fuzzy loss penalizes instances whe truth value for the subclass but a l
- This ensures that the learned emb hierarchical structure of the domai

- In logic implication is modelled as
$$A \rightarrow B \equiv \neg A \vee B$$
- In fuzzy logic implication becomes a function
$$I(A,B) \in [0,1]$$
- There are several possible implementations, e.g.:
$$I(A,B) = \min(1,1 - A + B)$$

A	B	I(A,B)
0	0	1
0	1	1
1	1	1
1	0	0

Disjointness Constraints

- Disjointness rules are implemented via axioms
  - e.g. *nothing can be both a cat and a dog*

$$\forall\, x \neg (Cat(x) \wedge Dog(x))$$

- Enforcing disjointness helps stabilize training and improves the precision of multi-class classification tasks.

Relational Learning through Axioms

- Logic allows for relating multiple objects through complex formulas, such as symmetry

$$\forall\, x, y\, \left(isFriend(x, y) \rightarrow\ isFriend(y, x)\right)$$

- The model learns to predict relationships not just from individual features, but from the **structural dependencies between entities**
  - This is a cornerstone of KGC, where the goal is to infer new facts from existing relational patterns.

LTN for Knowledge Graph Completion

- By using **Real Logic**, LTN transforms the task of predicting missing links into a **differentiable optimization problem** where logical formulas are grounded into neural computational graphs
- While traditional embedding models rely on simple scoring functions, LTN evaluates the **plausibility of facts** based on their **global consistency** with a set of logical axioms and background knowledge

KG Elements to LTN Signature

- In the LTN framework, KG **entities** are represented as **constant symbols**.
  - These are grounded into real-valued tensors, such as **trainable embeddings** or fixed feature vectors.
- KG relations are modeled as **predicate symbols**.
  - These are grounded as functions—typically **Multi-Layer Perceptrons (MLPs)**—that project input tensors into a fuzzy truth value in the interval [0,1].
- Every fact in the graph, represented as a **triple** , is interpreted as an **atomic formula**
  - The output of the grounded predicate represents the system's degree of confidence in the truth of that specific link

Embedding Learning through Grounding

- LTN facilitates **embedding learning** by treating the grounding of constants as a set of **learnable parameters**
- Unlike shallow KGE models (e.g., TransE), LTN optimizes entity vectors, so they satisfy not just the observed facts, but also the **complex relational patterns** defined in the logic

Ontological Knowledge via Axioms

- LTN allows users to explicitly encode **structural properties** of relations, such as *symmetry*, *transitivity*, or *inverses*, using formulas
- **Ontological constraints** derived from schemas—such as class *subsumption* or *disjointness* are integrated into the training process as logical axioms
- These axioms provide a **strong inductive bias** that guides the model to learn representations that are semantically valid and logically consistent across the entire KG

Ontological Knowledge via Axioms

OWL Construct (OWL)	**Meaning**	**LTN Translation**	**Differentiable?**
`A ⊑ B` (SubClassOf)	A is subset of B	$\forall x \ (A(x) \rightarrow B(x))$	✓ Yes
`A ≡ B` (Equivalence)	A and B equivalent	$\forall x \ (A(x) \leftrightarrow B(x))$	✓ Yes
`A ⊓ B ⊑ C`	Intersection implies C	$\forall x \ \big(A(x) \wedge B(x)\big) \rightarrow C(x)))$	✓ Yes
`A ⊓ B = ⊥`	Disjoint classes	$\forall x \ \neg(A(x) \wedge B(x))$	✓ Yes
`A ⊑ ¬B`	A disjoint from B	$(\backslash\text{forall } x \ (A(x)\backslash\text{rightarrow } \backslash\text{neg } B(x)))$	✓ Yes
`R(x,y)` (Assertion)	Relation holds	$(G(R)(x,y))$	✓ Yes
`R ⊑ S`	Subproperty	$(\backslash\text{forall } x,y \ (R(x,y)\backslash\text{rightarrow } S(x,y)))$	✓ Yes
`R symmetric`	Symmetry	$(\backslash\text{forall } x,y \ (R(x,y)\backslash\text{rightarrow } R(y,x)))$	✓ Yes
`R transitive`	Transitivity	$(\backslash\text{forall } x,y,z$ $((R(x,y)\backslash\text{land } R(y,z))\backslash\text{rightarrow } R(x,z)))$	✓ Yes (approx.)
`domain(R)=A`	Domain restriction	$(\backslash\text{forall } x,y \ (R(x,y)\backslash\text{rightarrow } A(x)))$	✓ Yes
`range(R)=B`	Range restriction	$(\backslash\text{forall } x,y \ (R(x,y)\backslash\text{rightarrow } B(y)))$	✓ Yes
`∃R.A` (existential restriction)	Some relation to A	$(\backslash\text{forall } x \ (\backslash\text{exists } y \ (R(x,y)\backslash\text{land } A(y))))$	⚠ Approx.
`≤ n R.A` (cardinality ≤ n)	At most n relations	$soft\ constraint\ (penalty\ on\ counts)$	❌ Not exact
`= n R.A`	Exactly n relations	$soft\ counting\ loss$	❌ Not exact
`∀R.A` (universal restriction)	all R-values in A	$(\backslash\text{forall } x,y \ (R(x,y)\backslash\text{rightarrow } A(y)))$	✓ Yes

Satisfiability as Loss

- The core of LTN learning is the maximization of the **theory's satisfiability**, which is the aggregation of truth values for all factual and generalized formulas in the KG
- Discrepancies between neural predictions and logical rules are measured via a **Fuzzy Loss** (calculated as $1 - Satisfiability$), acting as a regularizer that penalizes contradictory predictions
- The system uses standard backpropagation to adjust both the **weights of the neural predicates** and the **entity embeddings**

Link Prediction

- Once trained, the LTN becomes a **grounded theory** that can be queried for the truth value of any formula:
  - Every fact in a KG is treated as an atomic formula r(h,t).
  - When querying a missing tail (h,r,?) the system evaluates a truthiness score representing the system's degree of confidence that the link is valid.
- The Ranking Process:
  - the system iterates across the entire domain of candidate entities.
  - It performs a truth query for each candidate triple (e.g., r(h,e1),r(h,e2),…)
  - computes their respective truth values and rank the candidates

Example: Consider the query (*Google*, *LocatedIn*, *?*). The LTN uses its grounded predicate for *LocatedIn* to score various candidate entities. It might assign a truth value of 0.98 to *Mountain View*, 0.45 to *New York*, and 0.02 to *Rome*. The entity with the highest likelihood (*Mountain View*) is then predicted as the missing link.

Generalization to Unseen Data

- LTNs can perform **open-world knowledge graph completion:**
  - LTN can evaluate relationships involving **unseen entities** using **feature-based representations**

- How LTNs Handle New Entities:
  - Let suppose entity is represented via **features** (text, attributes, images)

    features(Alice) = "works at Google"
  - An **encoder** (i.e., the learned grounding function) maps features to embeddings:

    $$\mathbf{v}_{Alice} = f_\theta(features)$$
  - Predicates are trained **neural functions** and can be used to evaluate the likelihood for new relations:

    $$G(worksAt)(\mathbf{v}_{Alice}, \mathbf{v}_{Google}) = 0.95$$
- Logical rules can be used to infer additional relations:

  $$worksAt(x, y) \wedge locatedIn(y, z) \rightarrow livesIn(x, z)$$

- New entities can be used **immediately**, even if not seen during training, and new triple can be added to the KG:
  - *(Alice, worksAt, Google)*
  - *(Alice, livesIn, USA)*

Generalization to Unseen Data

- LTNs can perform **open-world knowledge graph completion** improving over solution like TransE or R-GCN
  - TransE: memorizes graph structure
  - R-GNN: learns from neighbours
  - LTN: learns functions + rules

Model	Entity Representation	New Entities	Role of Graph	Logic
TransE	Lookup embeddings (IDs)	❌ Not supported	Central	❌
R-GCN	Message passing on graph	⚠️ Only if connected	Essential	❌
LTN	Feature-based embeddings	✅ Require features	Optional	✅ Explicit

Logic Neural Network

- LNNs are neural networks **that are logical formulas**

- Each neuron maps logical element:
  - atoms = predicates
  - operators = AND, OR, NOT

- Network **model the syntactic structure** of the logic formula
  - interpretable by design

Riegel, Ryan, et al. "Logical neural networks." *arXiv preprint arXiv:2006.13155* (2020).

Truth Value Representation

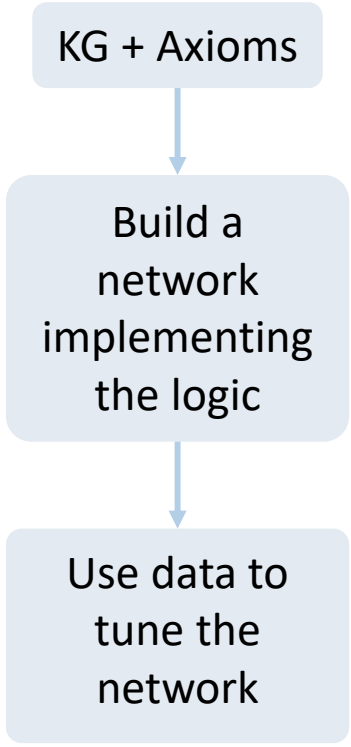
- In LNNs, each formula returns an interval:

$$[L, U] \; s.t. \; 0 \leq L \leq U \leq 1$$

- Interpretation:
  - [1,1]: true
  - [0,0]: false
  - [0,1]: unknown
  - L>U: contradiction

LNN for KGC

- Input:
  - known triples in the KG
  - logical rules (e.g., ontology axioms)
- Network construction:
  - grounding logical formulas
  - building corresponding neurons
- Training:
  - optimization of logical parameters
- Inference:
  - compute bounds for unseen triples

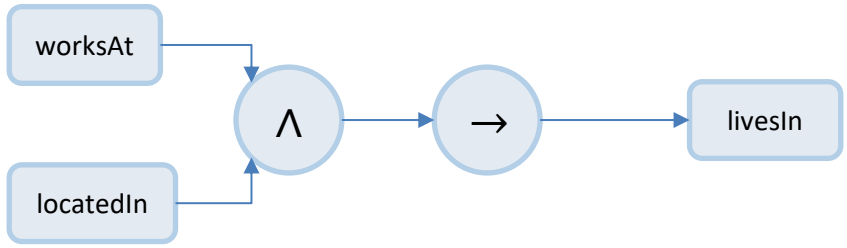


LNN for KGC

- A triple like $worksAt(x, y)$ becomes an **atomic neuron**
- A rule like

$$worksAt(x, y) \wedge locatedIn(y, z) \rightarrow livesIn(x, z)$$

become a **set of neurons and logic connectors** arranged to model the logic tree of the formula



Example KG

- *Entities*: Alice, Bob, Apple, Cupertino
- *Relations*: $worksAt(x, y)$, $locatedIn(y, z)$, $livesIn(x, z)$
- *Facts*:
  - $(Alice, worksAt, Apple)$
  - $(Apple, locatedIn, Cupertino)$
- *Rules*:
  $worksAt(x, y) \wedge locatedIn(y, z) \rightarrow livesIn(x, z)$
- *Grounding*:

worksAt

x	y	value
Alice	Apple	[1,1]
Bob	Apple	[0,1]

locatedIn

y	z	value
Apple	Cupertino	[1,1]

livesIn

x	z	value
Alice	Cupertino	[0,1]
Bob	Cupertino	[0,1]

Unknow facts

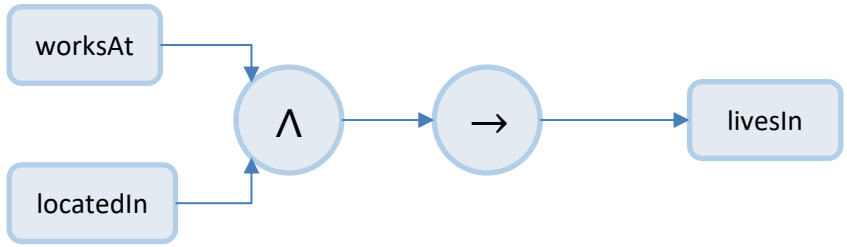
LNN for KGC

Upward inference:

- The rule modelled in the network allow to infer a new fact:

$$livesIn(Alice, Cupertino) = [1,1]$$

- Link prediction: a new relation is created in the KG
- Differently not much can be said about Bob



Example KG

- *Entities*: Alice, Bob, Apple, Cupertino
- *Relations*: $worksAt(x, y)$, $locatedIn(y, z)$, $livesIn(x, z)$
- *Facts*:
  - $(Alice, worksAt, Apple)$
  - $(Apple, locatedIn, Cupertino)$
- *Rules*:
  $worksAt(x, y) \wedge locatedIn(y, z) \rightarrow livesIn(x, z)$
- *Grounding*:

worksAt

x	y	value
Alice	Apple	[1,1]
Bob	Apple	[0,1]

locatedIn

y	z	value
Apple	Cupertino	[1,1]

livesIn

x	z	value
Alice	**Cupertino**	**[0,1]**
Bob	Cupertino	[0,1]

Unknow facts

LNN for KGC

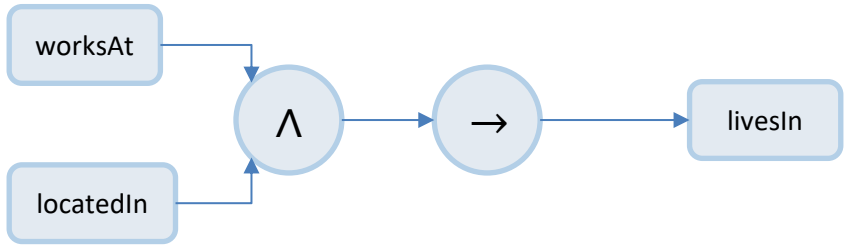
Downward inference:

- A new fact is added: *Bob does not live in Cupertino*

$$livesIn(Bob, Cupertino) = [0,0]$$

- Given the rule, since the conclusion is false, the antecedent is probably false:

$$worksAt(Bob, Apple) = [0,0.2]$$



Example KG

- *Entities*: Alice, Bob, Apple, Cupertino
- *Relations*: $worksAt(x, y)$, $locatedIn(y, z)$, $livesIn(x, z)$
- *Facts*:
  - $(Alice, worksAt, Apple)$
  - $(Apple, locatedIn, Cupertino)$
  - *(Bob, livesIn, Cupertino)* = FALSE
- *Rules*:
  $worksAt(x, y) \wedge locatedIn(y, z) \rightarrow livesIn(x, z)$
- *Grounding*:

worksAt

x	y	value
Alice	Apple	[1,1]
Bob	**Apple**	**[0,0]**

locatedIn

y	z	value
Apple	Cupertino	[1,1]

livesIn

x	z	value
Alice	Cupertino	[0,1]
Bob	**Cupertino**	**[0,0]**

LNN for KGC

- LNN are particularly powerful in case of **noisy knowledge**:
  - worksAt(Alice, Apple) = [0.9, 1]
  - locatedIn(Apple, Cupertino) = [1, 1]
  - livesIn(Alice, Cupertino) = [0.5, 0.7]
- The network learn to adapt the **weigh of the rule** and alter the **behavior of AND and IMPLIES** to
  - reduce contradiction
  - keep consistency

LNN can naturally **incorporate probabilistic data** when available, handling both certain facts and uncertain evidence within the same logical framework.

In LNN, learning does not operate over entity representations, but over the **parameters governing logical operators** and **rule strengths**, enabling the model to adapt the rigidity of logical reasoning to noisy and incomplete data.

This is particularly useful when knowledge is obtained from automatically extracted, incomplete, and potentially noisy data.

LNN Coding

- A real LNN is not a neural network with a bit of logic but a **differentiable logic inference engine**
- The loss penalizes
  - contradictions
  - logic violations
- See also:
  - https://github.com/IBM/LNN
  - https://ibm.github.io/LNN/introduction.html

```
from lnn import *

# predicates
WorksAt = Predicate("WorksAt")
LocatedIn = Predicate("LocatedIn")
LivesIn = Predicate("LivesIn")

# variables
x, y, z = Variables("x", "y", "z")

# rule
rule = Implies(
    And(
        WorksAt(x, y),
        LocatedIn(y, z)
    ),
    LivesIn(x, z)
)

# model
model = Model()

model.add_knowledge(rule)

# facts
model.add_data({
    WorksAt: {("Alice", "Apple"): TRUE},
    LocatedIn: {("Apple", "Cupertino"): TRUE}
})

# reasoning
model.infer()

# inspect result
print(LivesIn.state(("Alice", "Cupertino")))
```

Scaling Neuro-Symbolic Reasoning Beyond LNNs

- LNNs provide:
  - interpretability
  - explicit logical reasoning
  - contradiction handling
- … but:
  - grounding becomes expensive
  - difficult to scale to industrial-scale KGs
- New frameworks aim to **preserve logical reasoning** while scaling to **millions of triples**

DiffLogic

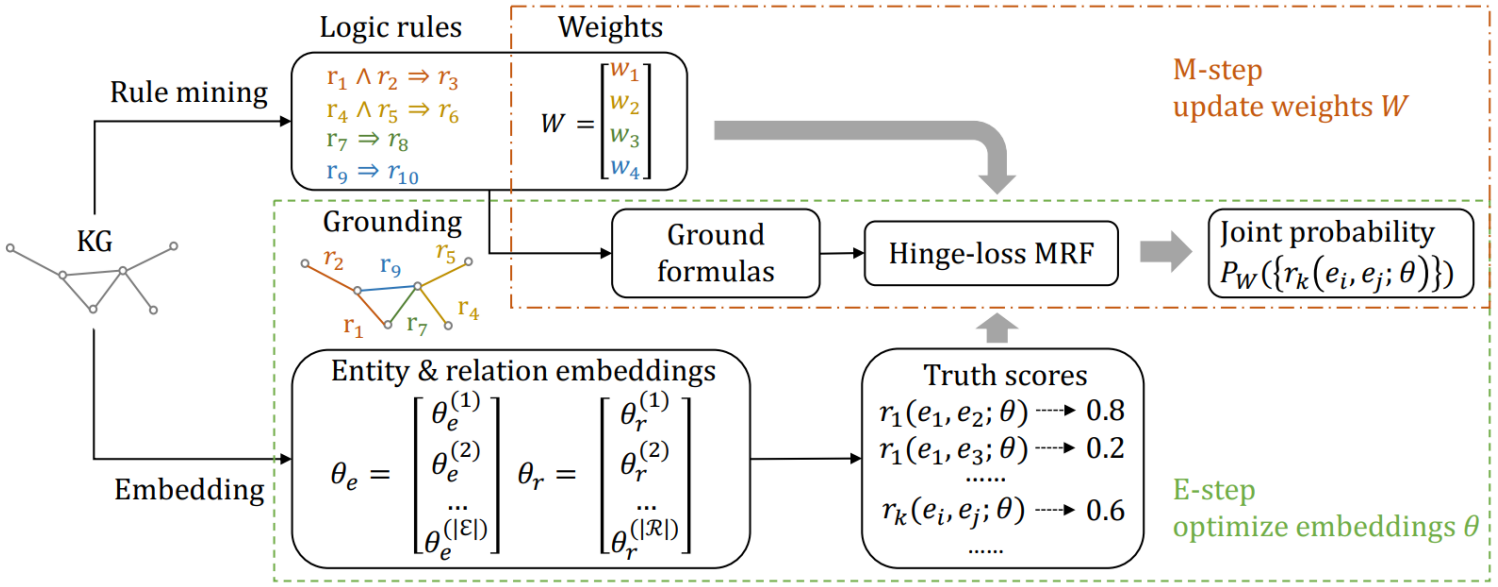
- DiffLogic combines into a unified framework:
  - KG embeddings
  - probabilistic logic rules
  - differentiable optimization
- Core Components
  - entity/relation embeddings
  - logical rules
  - Probabilistic Soft Logic (PSL)
  - adaptive grounding filters

Model	Main Representation
LTN	neural predicates
LNN	logical computation graph
DiffLogic	embeddings + differentiable logic constraints

Shengyuan, Chen, et al. "Differentiable neuro-symbolic reasoning on large-scale knowledge graphs." *NeurIPS* 2023

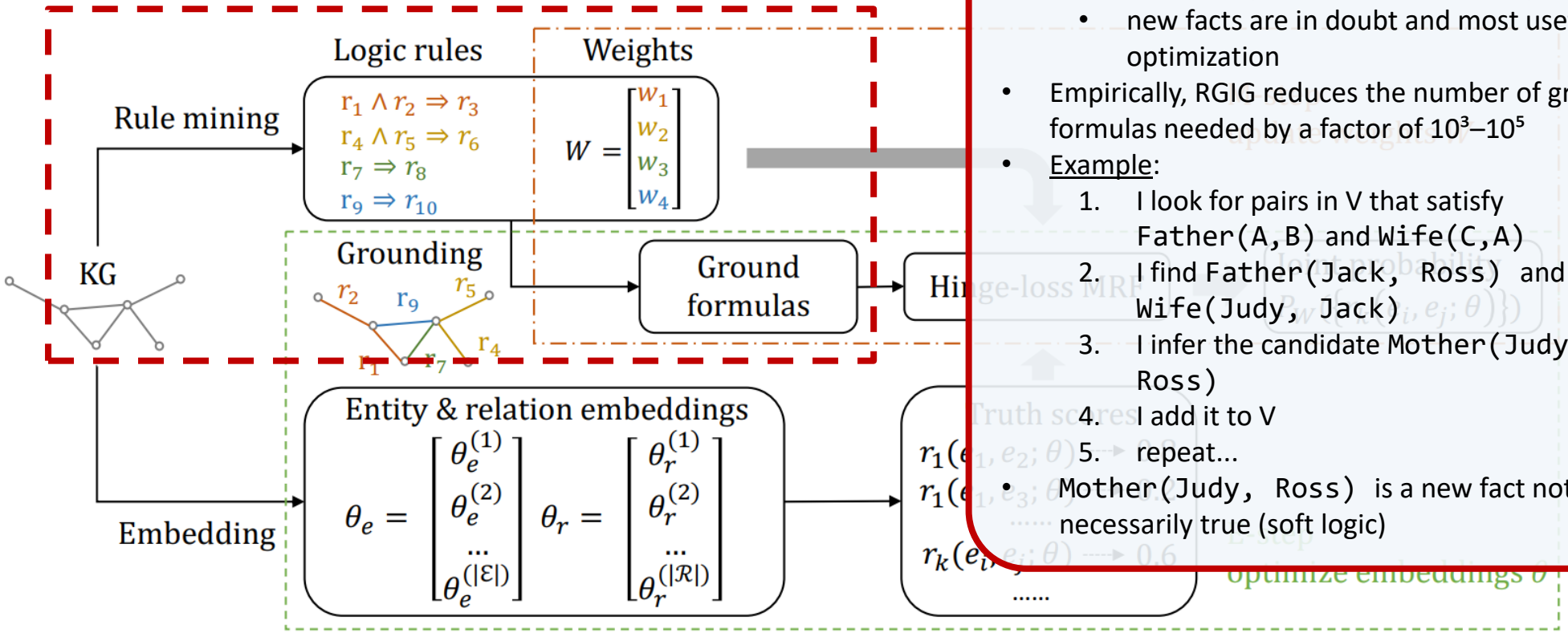
How DiffLogic Works

- Input
  - KG triples
  - logical rules
- Learn vector representations for entities and relations
- Proposes a differentiable framework that combines **embedding** and **rule-based reasoning**, using a continuous logical network called Probabilistic Soft Logic
- Assume soft logic and incomplete KG



Shengyuan, Chen, et al. "Differentiable neuro-symbolic reasoning on large-scale knowledge graphs." *NeurIPS* 2023

How DiffLogic Works

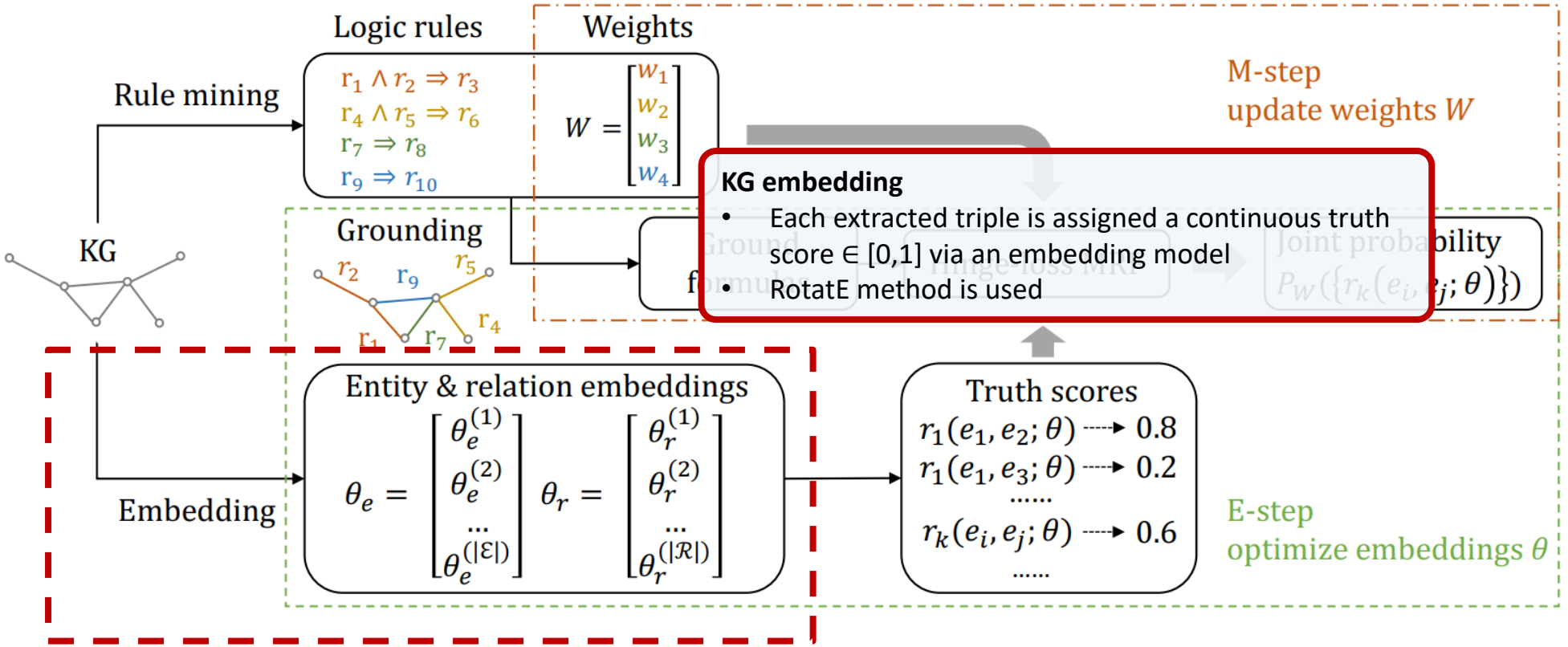


Rule-Guided Iterative Grounding

- RGIG iteratively ground formulas adding new facts
  - new facts are in doubt and most useful for optimization
- Empirically, RGIG reduces the number of ground formulas needed by a factor of 10^3–10^5
- Example:
  1. I look for pairs in V that satisfy `Father(A,B)` and `Wife(C,A)`
  2. I find `Father(Jack, Ross)` and `Wife(Judy, Jack)`
  3. I infer the candidate `Mother(Judy, Ross)`
  4. I add it to V
  5. repeat...
- `Mother(Judy, Ross)` is a new fact not necessarily true (soft logic)

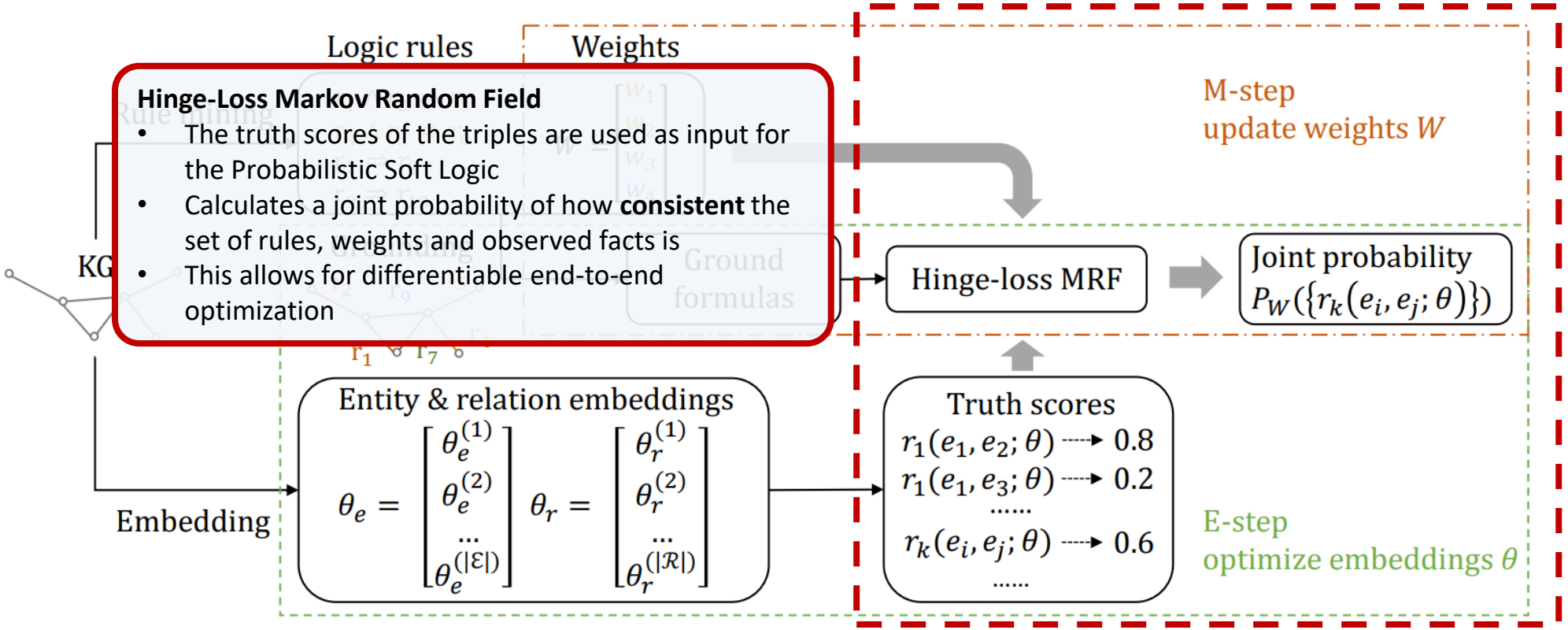
Shengyuan, Chen, et al. "Differentiable neuro-symbolic reasoning on large-scale knowledge graphs." *NeurIPS* 2023

How DiffLogic Works



Shengyuan, Chen, et al. "Differentiable neuro-symbolic reasoning on large-scale knowledge graphs." *NeurIPS* 2023

How DiffLogic Works

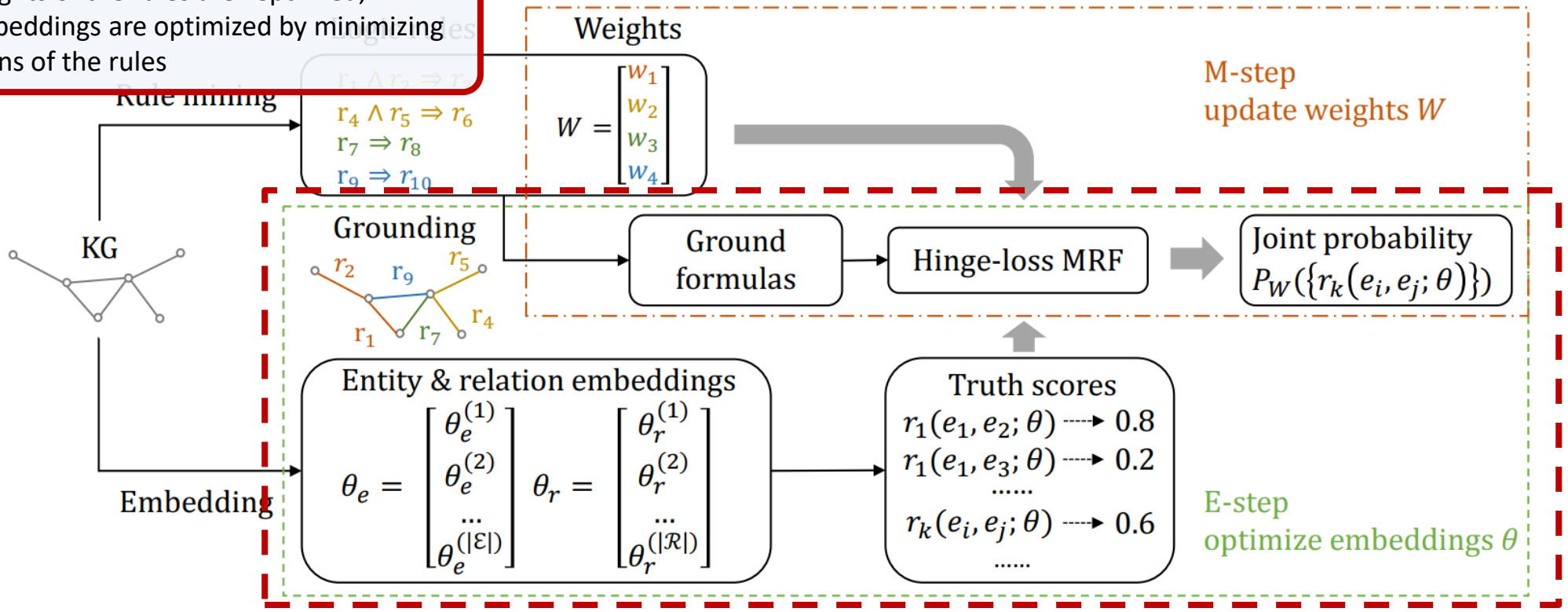


Shengyuan, Chen, et al. "Differentiable neuro-symbolic reasoning on large-scale knowledge graphs." *NeurIPS* 2023

How DiffLogic Works

Training

- **E-step**
  - the weights of the rules are kept fixed,
  - the embeddings are optimized by minimizing violations of the rules

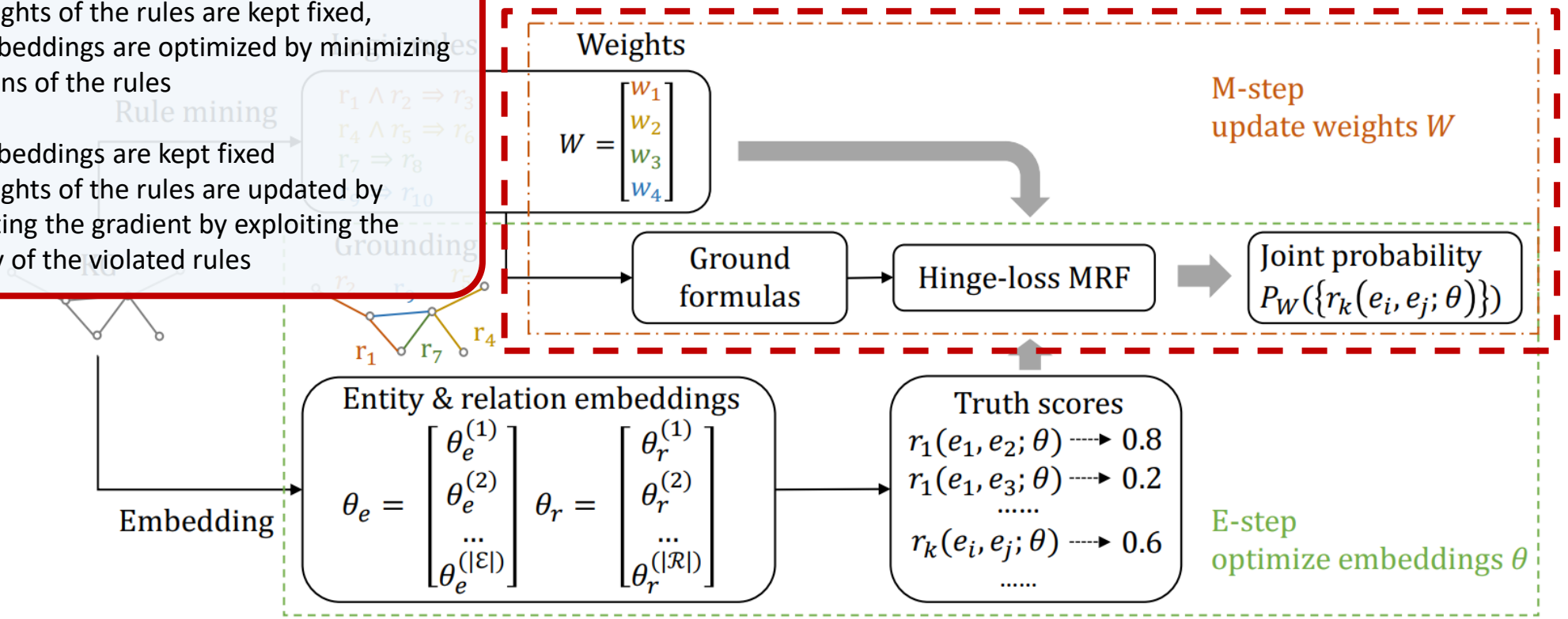


Shengyuan, Chen, et al. "Differentiable neuro-symbolic reasoning on large-scale knowledge graphs." *NeurIPS* 2023

How DiffLogic Works

Training

- **E-step**
  - the weights of the rules are kept fixed,
  - the embeddings are optimized by minimizing violations of the rules
- **M-step**
  - the embeddings are kept fixed
  - the weights of the rules are updated by estimating the gradient by exploiting the sparsity of the violated rules



Shengyuan, Chen, et al. "Differentiable neuro-symbolic reasoning on large-scale knowledge graphs." *NeurIPS* 2023

LTN vs LNN vs DiffLogic

Aspect	LTN	LNN	DiffLogic
Main idea	Neural predicates	Logic as network	Embeddings + soft logic
Learnable parameters	predicate NNs	logical operators	embeddings + rule weights
Truth values	scalar [0,1]	intervals [L,U]	scalar [0,1]
Inference	forward fuzzy evaluation	upward/downward reasoning	probabilistic optimization
Scalability	medium	limited	high
Interpretability	partial	very high	moderate
Best use case	soft logical regularization	explainable reasoning	large-scale KGC

Rule Learning for Knowledge Graph Completion

- Rule learning for KGC encompasses the automatic extraction, ranking, and application of **logical rules** (Horn Clause) from knowledge graphs to infer missing links
- A first-order Horn rule takes the form $b_1 \wedge \ldots \wedge b_n \rightarrow h$
  - Intuitively, the rule reads: if $b_1, \ldots, b_n$ all hold, then h holds too

```
bornInCity(X, Z) ∧ cityInState(Z, W) ∧ stateInCountry(W, Y) → hasNationality(X, Y)
```

Symmetry

```
marriedTo(X, Y) → marriedTo(Y, X)
```

Transitivity

```
hasChild(X, Y) ∧ hasChild(Y, Z) → hasGrandchild(X, Z)
```

Classical ILP Pipeline

- Goal: *learn symbolic rules from observed facts*
- Inductive Logic Programming learns symbolic rules using:
  - positive examples
  - negative examples
  - background knowledge
- Then, the system
  - generate candidate rules
  - test them against the KG
  - keeps rules that explain positives while avoiding negatives

Limitation

Rule search is:

- discrete
- combinatorial
- non-differentiable

Difficult to scale to large noisy KGs!

Statistical Path Generalisation

AnyBURL

- Sample random paths in the KG, then abstract entities into variables
- Efficient, anytime algorithm — can stop early with decent rules
- Matches embedding models in prediction quality + adds explanations

AnyBURL in Action

- Sample a path:

`(Alice)-bornInCity-> (Rome)-cityInCountry-> (Italy)`

- Generalize: replace entities with variables:

$$\texttt{bornInCity(X, Z)} \wedge \texttt{cityInCountry(Z, Y)} \rightarrow \texttt{hasNationality(X, Y)}$$

- Measure confidence on the full KG
- Apply rule to predict: `hasNationality(Bob, ?)` if we know `bornInCity(Bob, Milan)` and `cityInCountry(Milan, Italy)`

Meilicke, Christian, et al. "Anytime Bottom-Up Rule Learning for Knowledge Graph Completion." *IJCAI*. 2019.

Neuro-Symbolic Rule Learning

- Instead of explicitly searching for rules (like ILP) or sampling paths (like AnyBURL), neuro-symbolic methods encode rules as **differentiable operations** and let gradient descent figure out which rules are useful
- Rules and their weights are learned simultaneously.

Yang, Fan, Zhilin Yang, and William W. Cohen. "Differentiable learning of logical rules for knowledge base reasoning." *NeurIPS* 2017

Neuro-Symbolic Rule Learning

KG

(Alice, bornInCity, Rome)
(Rome, cityInCountry, Italy)
(Bob, bornInCity, Milan)
(Milan, cityInCountry, Italy)
(Alice, hasNationality, Italy)
(Bob, hasNationality, ?)

Step 1 — Represent relations as matrices

- Each relation becomes a **matrix** where entry [i,j] = 1 if triple (entity_i, relation, entity_j) exists.
- This is just a graph adjacency matrix per relation.

Step 2 — Chain matrices = chain relations

- Multiplying two matrices chains two relations together:
  M_bornInCity × M_cityInCountry
- This product gives, for each entity X, all the countries reachable via `X-bornInCity->Z-cityInCountry->Y`.
- It's the matrix encoding of the rule:

$$bornInCity(X, Z) \wedge cityInCountry(Z, Y) \rightarrow hasNationality(X, Y)$$

Step 3 — Learn weights

- Combine *many* such chain products with **learned scalar weights**:

$$P(hasNationality(X, Y)) \approx w_1 \cdot [M\_bornInCity \times M\_cityInCountry] + w_2 \cdot [M\_citizenOf] + w_3 \cdot [M\_livesIn \times M\_locatedIn] + \ldots$$

- During training, the model sees known `hasNationality` facts and adjusts the weights via **backpropagation**
- Rules that explain the data well get high weights; useless ones get weights near zero.

Neuro-Symbolic Rule Learning

Computational cost

- For a KG with N relations:
  - 1-hop chains: N combinations
  - 2-hop chains: N × N = N² combinations
  - 3-hop chains: N × N × N = N³ combinations
- On real KG N can be in the order of hundreds or thousands of relations so N³ quickly becomes prohibitive

Search space reduction

- **Sparsity pruning**: If the product $M_{r1} \times M_{r2}$ is an all-zero matrix (no such path exists in the graph), that chain is discarded
- **Maximum fixed length** typically k=2 or k=3
- Only "compatible" relationships:
  - r1 goes to entities of type City
  - r2 starts from entities of type Person
  - their product doesn't make sense semantically

Hybrid methods use embeddings to pre-filter which chains are worth considering. The embedding suggests *these two relationships are semantically related*, drastically reducing the number of products to be calculated

Omran, et. al "Scalable rule learning via learning representation." *IJCAI* 2018.

Language Models for Knowledge Graph Completion

Language Models for KGC

- Methods based on **embedding** (TransE, RotatE, DistMult) and **neuro-symbolic logic** (Logic Tensor Networks, Neural LP) have shown that graph structure is a powerful signal source for KG completion.
- However, they share a fundamental limitation: they operate exclusively on the topology of triples, **without accessing the linguistic meaning of entity and relationship** names
  - Example: case of a low-frequency entity:
    - An entity with very few triples involving it led to a poorly informative vector representation.
    - A language model can exploit the textual description of the entity to infer plausible relationships even in the absence of abundant structural data
- Additionally, relational schemas are rigid:
  - traditional methods operate on a fixed and pre-defined set of relationship types
  - when new relationships emerge or you switch to a different domain, the model must be re-trained from scratch on new annotated data
  - pre-trained language models are a natural response to these limitations: their exposure to huge textual corpora equips them with generalizable semantic knowledge, capable of supporting relational reasoning in open, low-resource scenarios

Integrating Textual Information

- Prior to the advent of deep language models, several works explored ways to **incorporate textual information into embedding models**, with promising but limited results.
- A pioneering approach represented entities as the **average of token word embeddings** in their names or descriptions.
  - For example, the entity "New York City" might be represented as the average of the vectors of "New", "York", "City" — a simple but context-insensitive encoding.
- Another direction **aligned Wikipedia entities and anchors** in a shared vector space:
  - if the Wikipedia text mentions "the Eiffel Tower" as a link, the model learns to place the corresponding entity near the word representations of the sentence describing it
- Subsequent methods have adopted convolutional neural networks (CNNs) to **encode sequences of words in entity descriptions**, capturing local patterns that the vector mean cannot represent.

Despite advances, these techniques **remain task-specific models trained on fixed corpora**: they do not possess the ability to generalize to new domains or unseen relationships, nor to treat language as a means of relational reasoning

KG-BERT: Triples as Text Sequences

- KG-BERT is the foundational work that redefined KG completion as a problem of natural language understanding, exploiting the contextual representations produced by BERT
- The central idea is simple but powerful:

> instead of mapping entities and relationships in vector spaces using geometric scoring functions, you **concatenate the textual descriptions** of the three components of a triple into a single input sequence for BERT

- Example: given *(Marie Curie, nobelPrize, Physics)*, the input to KG-BERT could be constructed as:

```
[CLS] Marie Curie, a Polish-French physicist and chemist [SEP] was awarded Nobel
Prize in [SEP] Physics, the natural science [SEP]
```

  - BERT processes this sequence bidirectionally, capturing dependencies between all tokens.
  - The special token [CLS] accumulates the contextual representation of the entire sequence and is used as input for a classification head that assigns a plausibility score to the triple.

Yao, Liang, Chengsheng Mao, and Yuan Luo. "KG-BERT: BERT for knowledge graph completion." *arXiv* (2019).

KG-BERT: Task Formulations and Input Design

- For **triple classification**, the model receives the complete sequence (h, r, t) and produces a binary label (valid/invalid).
  - Negative samples are generated by corrupting valid triples — randomly replacing the head, tail, or relationship — and the model is trained to discriminate them from positives.
- For **relation prediction**, given a couple of entities (h, t), the model scores all possible triples (h, r_i, t) for each relationship r_i in the vocabulary and selects the one with the highest score.
- A critical design element is the use of **entity and relationship descriptions** (rather than just surface names):
  - for "*bornIn*", the description "*the place where the subject entity was born*" is provided, significantly increasing the semantic signal available to the model and mitigating the problem of rare entities.

From Language Models to LARGE Language Models

- KG-BERT and its successors (KG-RoBERTa, MTL-KGC, StAR) use moderately sized encoder-only models
  - These models excel in ranking and ranking, but they don't generate text openly.
- The emergence of LLMs introduces a qualitative shift:
  - at sufficiently large scales, models exhibit emergent capabilities not present in smaller models, including **in-context learning**, **instruction following**, **multi-step reasoning**, and **open generation**.
  - **In-context learning:** given the prompt "The relation between Berlin and Germany is 'capitalOf'. What is the relation between Paris and France?", an LLM responds "capitalOf" without any update of weights — learning the pattern from the in-context demonstration alone.
  - **Instruction following**: An LLM can correctly respond to instructions such as "Given the triple (Picasso, nationality, ?), identify the missing entity from the following list: [Spain, France, Italy]" without having been trained specifically on this format.

These capabilities open up **new possibilities for the KGC**: LLMs can generate plausible completions, explain their predictions, and manage relationships and entities not seen during training.

At the same time, they introduce risks of **hallucination** and difficulties in **complying with rigid structural constraints**.

KG-LLM: Instruction Tuning for KGC

- KG-LLM extends the KG-BERT paradigm to the LLM era by adopting an instruction tuning strategy:
  - open-source decoder-based models (LLaMA-7B and ChatGLM-6B) are optimized for KGC tasks through **instruction-response pairs** built from benchmark datasets
  - the basic intuition is unchanged from KG-BERT — treating entities, relationships, and triples as textual sequences
  - the architecture changes dramatically: the entire task is formulated as sequence-to-sequence
  - LLM generates the response as free text

Triple classification:
Q: "Context: The triple (Einstein, receivedAward, Nobel Prize in Physics) is given. Is this triple correct? Answer yes or no."
A: "yes"

Entity prediction:
Q: "Given that (Marie Curie, bornIn, ?), select the correct answer from: [Warsaw, Berlin, Paris, Rome]."
A: "Warsaw"

The instruction tuning process builds hundreds of thousands of such pairs from the KGC benchmark training data

Yao, Liang, et al. "Exploring large language models for knowledge graph completion." *ICASSP 2025*

KG-LLM: Three Task Formulations and Prompt Design

- Triple Classification: (h, r, t) True or False?
  - The expected "yes/no" is a **generated token**, not a classification label → **the model to leverages the full distribution of its vocabulary to express uncertainty**
- Relation Prediction: Given h, t find r (among selected candidates)
  - the prompt can present the candidate relations in list form, forcing the output into the valid space; or operate in open mode, letting the model generate a relational label and then mapping it to the closest in the KG vocabulary
- Entity (Link) Prediction: $(h, r, ?)$ with candidate entities (selected via first-stage score embedding), asking the model to identify the most plausible one
  - This hybrid scheme — **embedding for candidate selection, LLM for re-ranking** — reduces the search space

- The design of the **prompt template is crucial**: the arrangement of the entity descriptions, the relationship name, the task statement, and the examples in context drastically affects performance
  - Small variations in the formulation can produce differences in accuracy of several percentage points.

KG-LLM Finding

- The most surprising empirical result of KG-LLM is that LLaMA-7B and ChatGLM-6B, **after instruction tuning**, outperform ChatGPT and GPT-4 on triple classification and relation prediction, despite the scale difference of one or two orders of magnitude in the parameters
- This result challenges the assumption that greater capabilities automatically imply better performance on any task:
  - in frontier LLMs, KGC capabilities are distributed among billions of parameters optimized for general objectives,
  - a smaller but specialized model **focuses its computational** resources on the specific task

Schema in Open-Domain KGC

- A structural obstacle of LLM-based methods for KGC in real-world scenarios involves **schema management**
- To produce valid triples, many approaches **include the schema in the model prompt**

- This works **becomes impractical for real KGs**
  - including many relationship types in the prompt of a model with a limited context leaves little context for the input text and demonstration examples
  - LLMs tend to ignore or confuse elements in distant positions in context ("lost in the middle" problem).
- Moreover, often a **pre-defined scheme does not exist** (e.g. company that wants a KG from its internal documents)

These problems motivate the design of approaches capable of operating both with large schemes (through selective retrieval) and without schemes (through automatic canonicalization)

Open Information Extraction with LLMs

- Open Information Extraction (OIE) is the paradigm that relaxes the constraint of the schema
- OIE systems extract arbitrary **relational statements from the text** in the form (subject, predicate, object), without the predicate having to belong to a pre-defined vocabulary
- LLMs have proven to be excellent OIE extractors in few-shot mode providing a few examples noted in the prompt

Text: "*Marie Curie, born in Warsaw in 1867, was the first woman to win a Nobel Prize and the only person to win Nobel Prizes in two different sciences*"

Extracted triples:

```
[Marie Curie, bornIn, Warsaw]
[Marie Curie, bornIn, 1867]
[Marie Curie, isThe, first woman to win a Nobel Prize]
[Marie Curie,  wonNobel, Physics]
[Marie Curie, wonNobel, Chemistry]
```

- Note that relationships such as "`isThe`" or "`wonNobel`" do not correspond to standardized types in any existing KG.
- This illustrates both the strength of the OIE (high semantic coverage) and its main limitation: the **output is non-canonicalized and not directly usable in a structured KG**.

Canonicalization Problem

The problem of canonicalization emerges when triples extracted from different texts describe the same fact with different relational formulations

- Without a standardization step, the resulting KG suffers from **redundancy** and **semantic ambiguity** that compromise its usefulness.
- Example:
  - "*Einstein worked as a professor at Princeton*" → [Einstein, workedAs, professor]
  - "*Einstein's job at Princeton was that of a researcher*" → [Einstein, job, researcher]
  - "*Einstein held a position at the Institute for Advanced Study*" → [Einstein, heldPosition, IAS]
- Traditional canonicalization methods rely on **clustering** of predicate embeddings
  - However, clustering is prone to over-generalization: semantically distinct relationships such as "is brother of" and "is main villain of" can be grouped in the same class
- The desired solution is a process of canonicalization that is **contextually sensitive**!

EDC Framework

- EDC is a structured three-step framework for constructing KGs from text
  - addresses the problem of canonicalization in a contextual way, exploiting the generative and explanatory capabilities of LLM
- **Extract**: extraction through few-shot prompting
- **Define**: Generation of a natural language definition for each component of the schema induced by the extracted triples
- **Canonicalize**: standardization of triples by comparison of vectorized definitions and LLM verification LLM
- EDC is designed to be flexible with respect to the availability of the schema:
  - Target Alignment mode: a target schema (potentially very large) is provided, and the extracted triples must be aligned to it
  - Self-Canonicalization mode: there is no pre-defined schema and EDC builds one automatically, consolidating semantically equivalent relationships

Zhang & Soh. "Extract, define, canonicalize: An llm-based framework for knowledge graph construction." *EMNPL* 2024

EDC+R for Large-Schema Settings

- For targeted modality in case of huge KG including the entire schema in the prompt **would be impractical**
- The refinement phase (EDC+R) addresses the problem of scalability by introducing a dedicated **Schema Retriever**:
  - a fine-tuned embedding model that, given an input text, efficiently retrieves the relationships of the target schema most likely to be instantiated in that text
- The mechanism is inspired by RAG:
  - instead of including the entire schema in the prompt only the relevant subset is retrieved and presented to the LLM during extraction.
- The Schema Retriever is a fine-tuned version of E5-Mistral-7b-instruct, trained with InfoNCE loss on positive pairs (input text, relationship instantiated in that text).

KICGPT

- KICGPT addresses a specific challenge of **long-tail entities** in KGC

- Example: The entity "*Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch*" (a Welsh village) rarely appears in any KG
  - A TransE model does not have enough data to build a meaningful embedding of it
  - A non-fine-tuned LLM can instead exploit their implicit knowledge of the geography of Wales to infer plausible relationships

- KICGPT proposes a **two-stage hybrid architecture**
  - Stage 1 — Structural Retriever: An embedding-based model efficiently identifies entity candidates for triple completion, exploiting the topological structure of the graph
  - Stage 2 — LLM re-ranker: The LLM receives first-stage candidates and re-ranks them based on semantic reasoning, guided by an in-context learning strategy called Knowledge Prompt

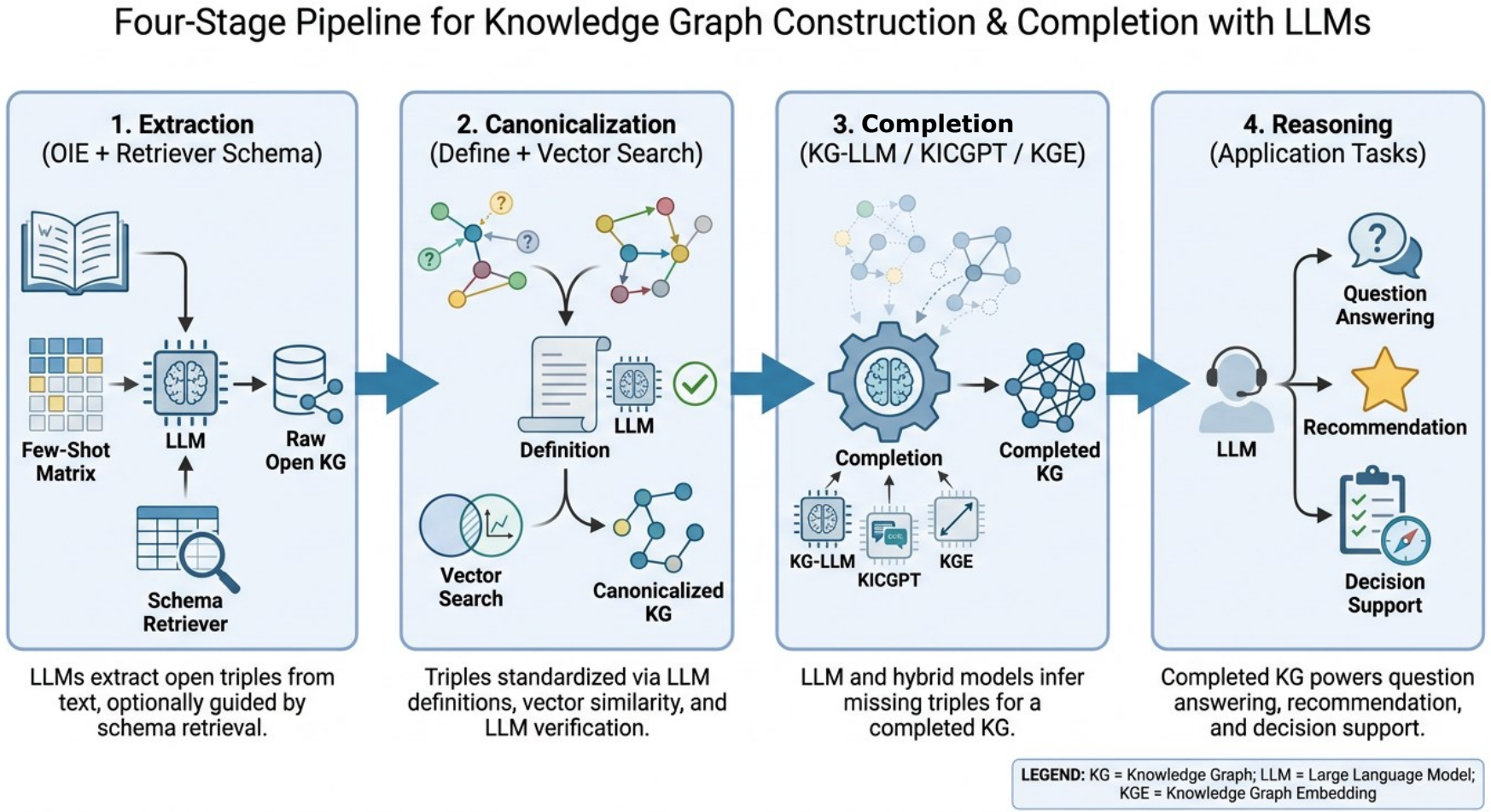
Wei et al. "KICGPTv2: Large Language Model with Knowledge in Context for Knowledge Graph Completion." *IEEE T-KDE* (2026).

The Knowledge Prompt Strategy

- The Knowledge Prompt is an in-context learning strategy that **encodes the structural knowledge of the KG** as natural language proofs in the context of the LLM
- This guides the model towards predictions consistent with the factual structure of the graph
- Example: given the task "*Predict the missing entity in (Paris, capitalOf, ?)*"
  - the system retrieves structurally analogous triples from the KG: (Berlin, capitalOf, Germany), (Rome, capitalOf, Italy), (Madrid, capitalOf, Spain)
  - includes them in the prompt as proofs
  - the LLM sees the relational pattern and applies it to complete the triple with "France"
- The key advantage is that the proofs provide **localized structural context around the target entity**, compensating for the poor graph coverage for long-tail entities
  - even if "Llanfairpwll..." has few triples, triples from nearby cities or entities of the same type (Welsh settlements) can effectively drive inference.
- KICGPT does not require any fine-tuning of the LLM

The division of labor between specialized components (the structural retriever for efficiency) and LLM (for semantic reasoning) produces more robust systems than the exclusive use of either paradigm.

From Raw Text to Completed KG



Open Challenges

- Hallucination and factual reliability: LLMs tend to produce semantically plausible but **fatally incorrect triples** — a particularly insidious problem because the hallucinations of a KGC system propagate silently in the graph, polluting downstream applications.
- Scalability to large-scale schemas and graphs: even with the Schema Retriever, EDC was evaluated on schemas with a maximum of 200 relationships. **Industrial KGs pose challenges of scale** that are still open (e.g. Wikidata: 11,000+ property types) .
- Long-tail entities: Despite the progress of KICGPT, the problem of rare entities remains partially unsolved. **Knowledge prompt quality degrades when there are no structurally similar entities** to use as proof in the graph.
- Evaluation of Self-Canonicalization: When there is no target schema, standard token-matching-based metrics are inapplicable. There is a **lack of consolidated automatic protocols** to evaluate the semantic quality of a self-built KG.
- LLM model dependency: The performance of EDC and KICGPT depends on the quality of the underlying LLM. The use of **closed-source models** (GPT-4) introduces dependence on external APIs, inference costs and risks related to model drift over time; **Open-source models** guarantee reproducibility but with lower performance.