Knowledge Graph Embeddings 101

(AKA Neural Link Predictors)

Pasquale Minervini
p.minervini@ed.ac.uk

Main References

Very good introductions to the field:

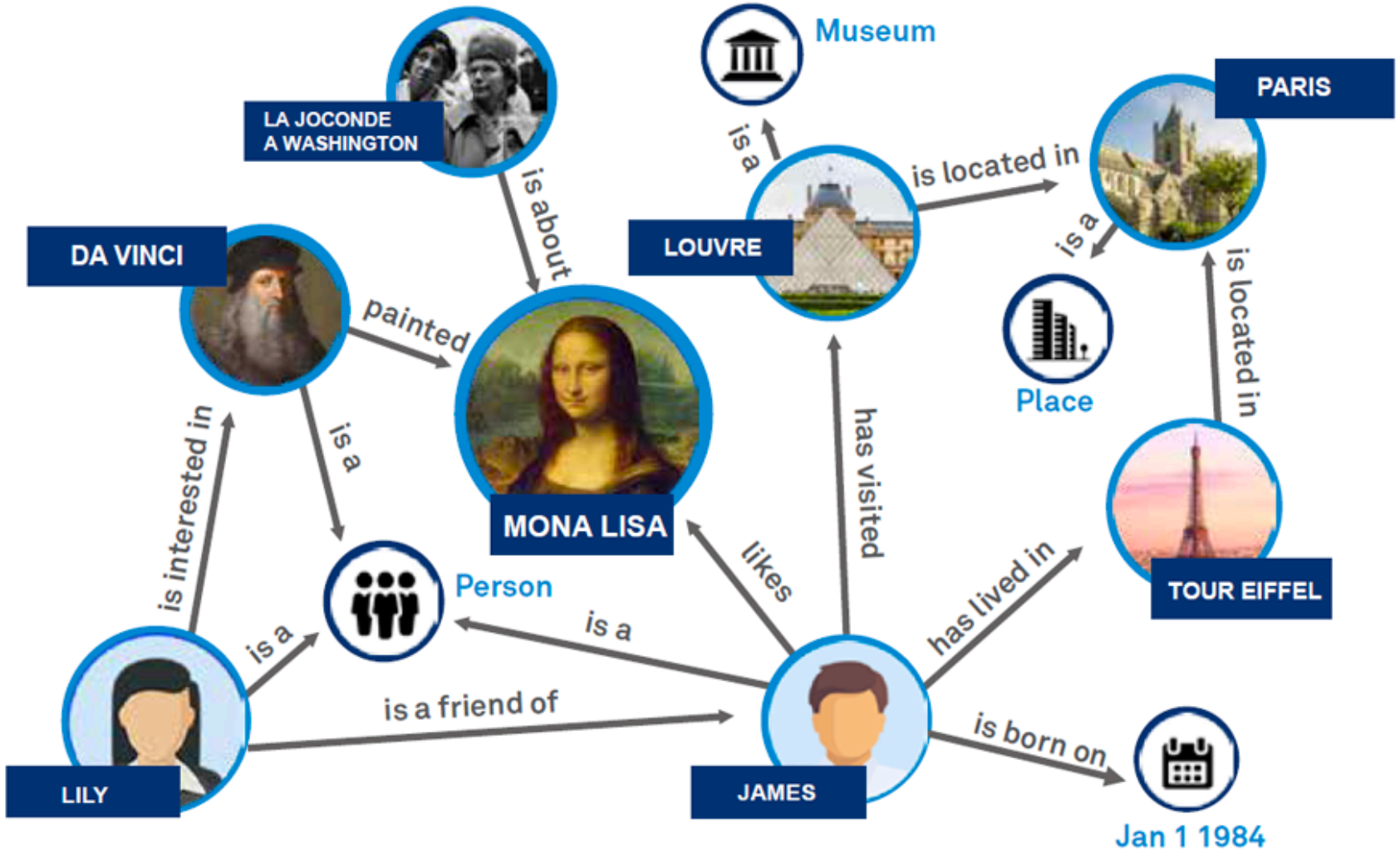
1. Nickel et al., A Review of Relational Machine Learning for Knowledge Graphs. https://arxiv.org/abs/1503.00759 ⭐
2. Hamilton, Graph Representation Learning Book. https://www.cs.mcgill.ca/~wlh/grl_book/ 📚
3. Goodfellow, Deep Learning. https://www.deeplearningbook.org/ 📚

Very simple models can work surprisingly well:

1. Ruffinelli et al., You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings. https://openreview.net/forum?id=BkxSmlBFvr 👩‍💻

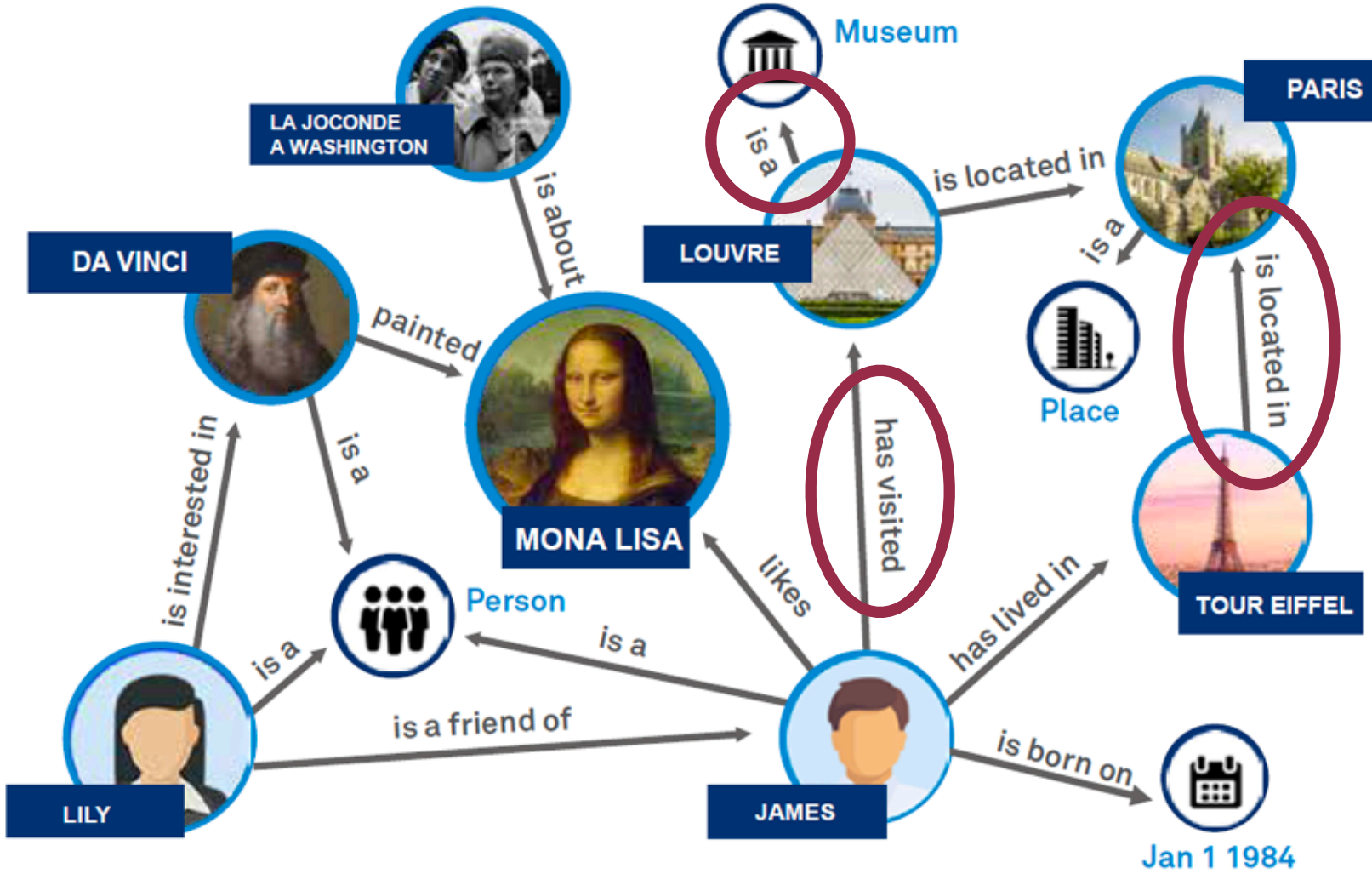
Knowledge Graphs

Knowledge Graph — graph-structured Knowledge Base, where knowledge about the world is encoded in the form of *relationships between entities*



Knowledge Graphs

Knowledge Graph — graph-structured Knowledge Base, where knowledge about the world is encoded in the form of *relationships between entities*



Nodes denote entities

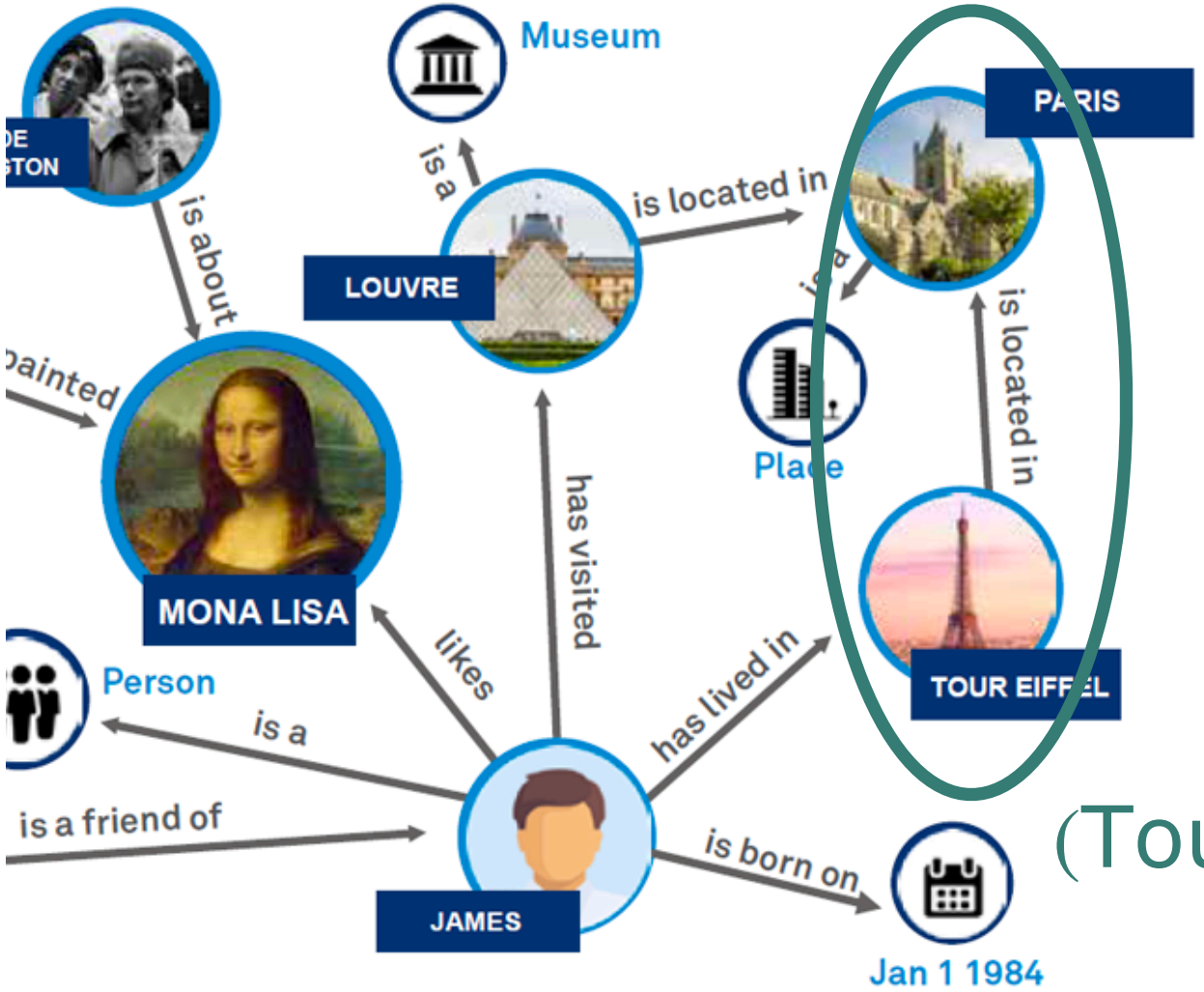
Edges denote relationships between entities

Multiple types of relations

All relations are binary

Knowledge Graphs

Knowledge Graph — graph-structured Knowledge Base, where knowledge about the world is encoded in the form of *relationships between entities*



$$G = \{(s, p, o)\} \subseteq E \times R \times E$$

s: subject of the triple

p: predicate of the triple

o: object of the triple

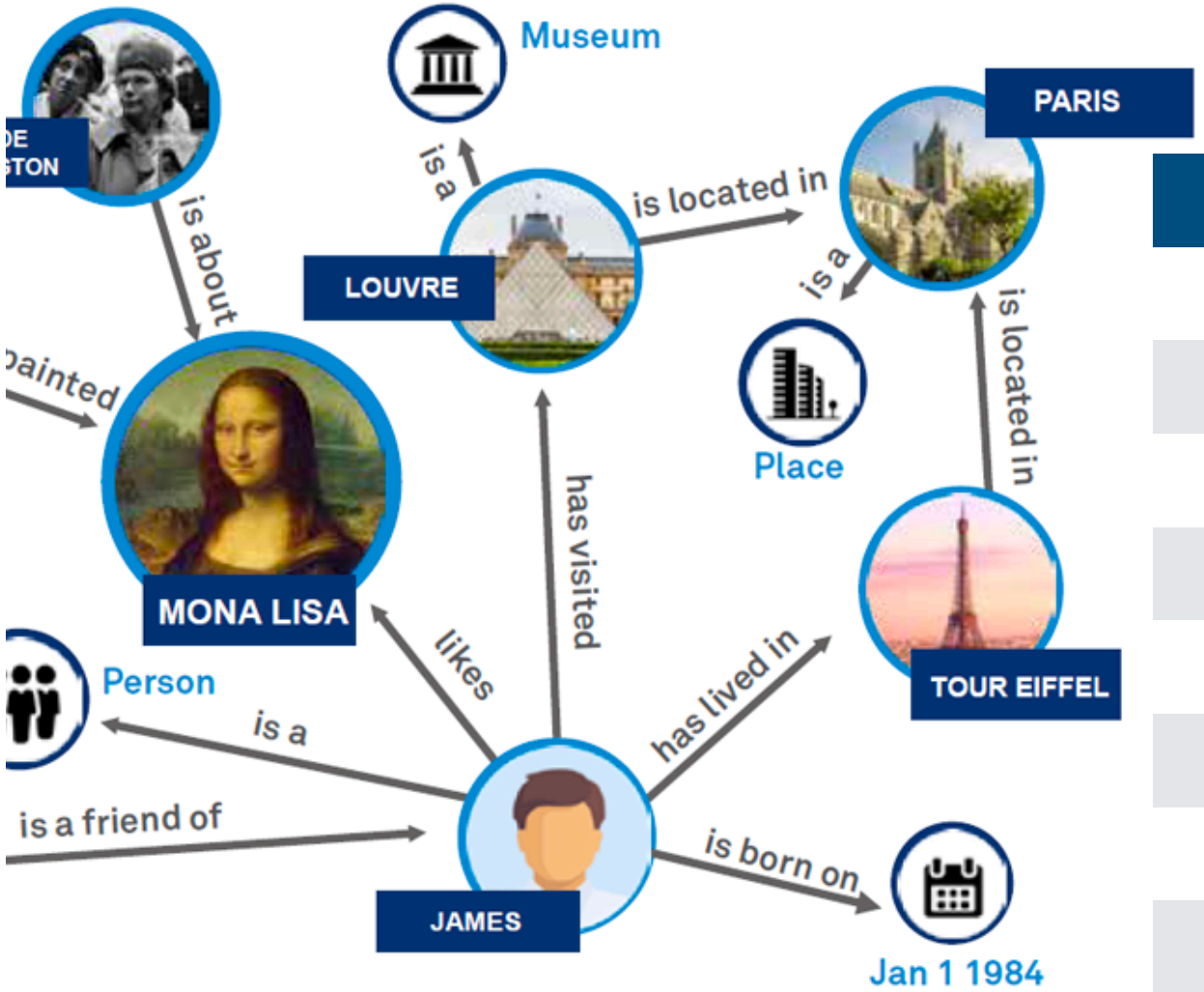
E: set of all entities

R: set of all relation types

$$(\text{Tour Eiffel}, \text{located in}, \text{Paris}) \in G$$

Knowledge Graphs

Knowledge Graph — graph-structured Knowledge Base, where knowledge about the world is encoded in the form of *relationships between entities*



$$G = \{(s, p, o)\} \subseteq E \times R \times E$$

s	*p*	*o*
Tour Eiffel	Is located in	Paris
Paris	Is a	Place
James	Has lived in	Tour Eiffel
James	Is born on	Jan 1 1984
James	Has visited	Louvre
Louvre	Is a	Musem
Louvre	Is located in	Paris
	…	

Knowledge Graphs

Knowledge Graph	# Triples ($\lvert G \rvert$)	# Entities
yago select knowledge	120M	10M
WIKIDATA	16.7B	107M
DBpedia	850M	6M
GDELT	3.5B	364M

Open World Assumption

Closed-World Assumption — A statement that is true is also known to be true. Therefore, what is not currently known to be true, is false.

Open World Assumption

Closed-World Assumption — A statement that is true is also known to be true. Therefore, what is not currently known to be true, is false.

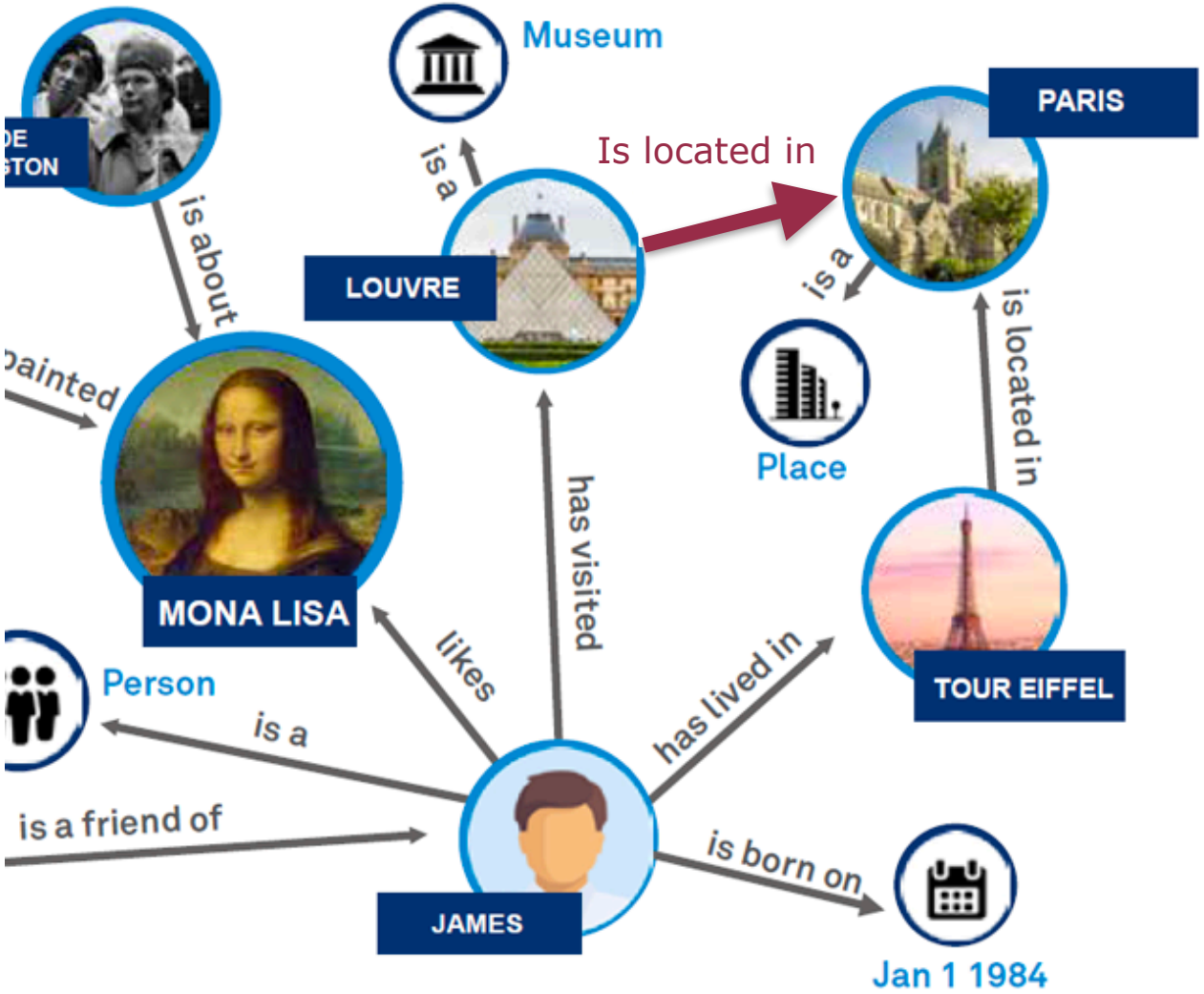
Open World Assumption — The absence of a particular statement means, in principle, that the statement has not been made explicitly yet, irrespective of whether it would be true or not. From the absence of a statement alone, we cannot infer that the statement is false.

KGs adopt this assumption

Machine Learning on KGs

Link Prediction/Triple Classification — Predicting the existence (of likelihood of correctness) of edges in the graph.

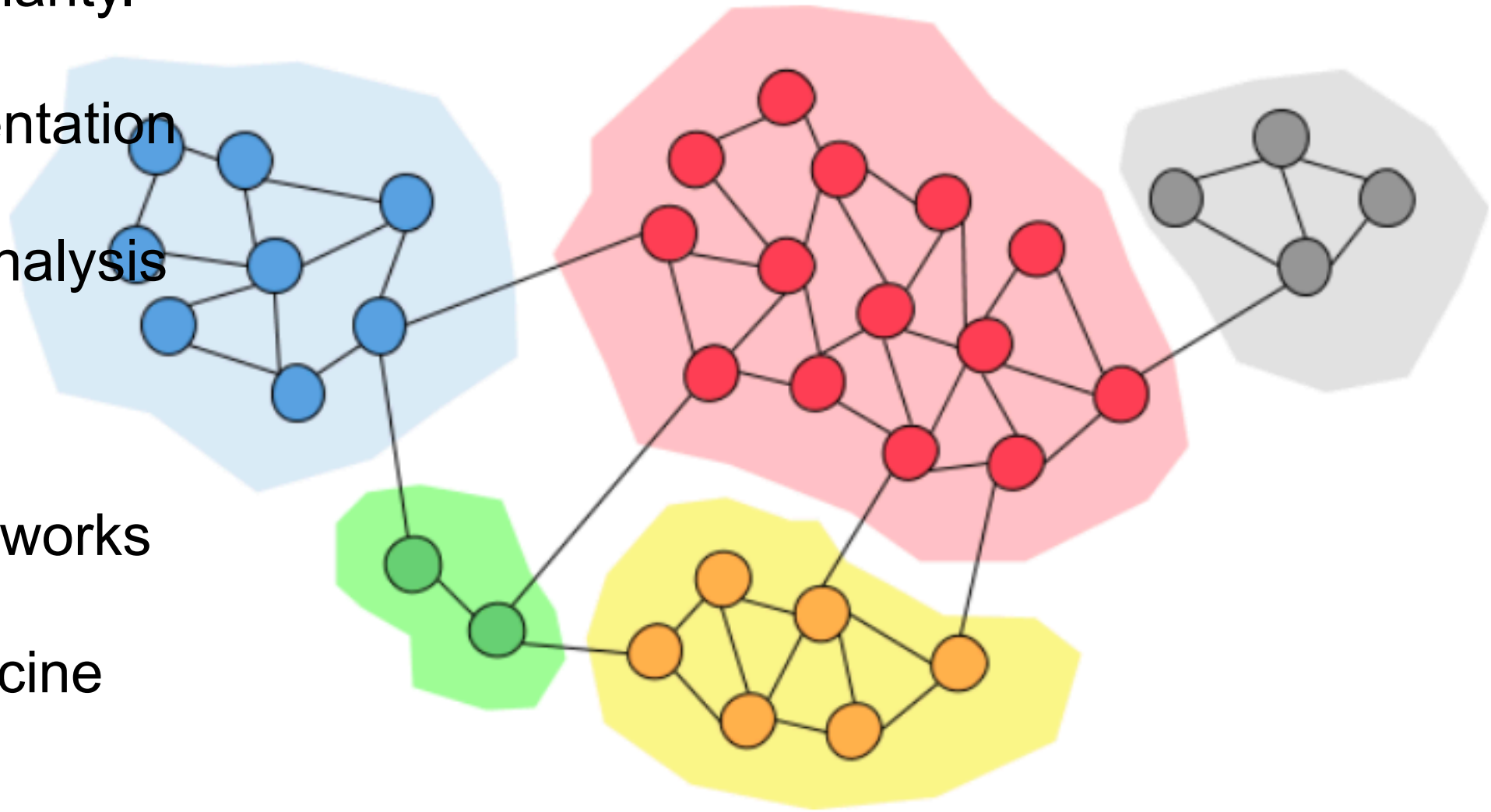
- Knowledge Graph Completion
- Recommendation
- Question/Query Answering
- Social Network Analysis
- Drug Repurposing/Retargeting



Machine Learning on KGs

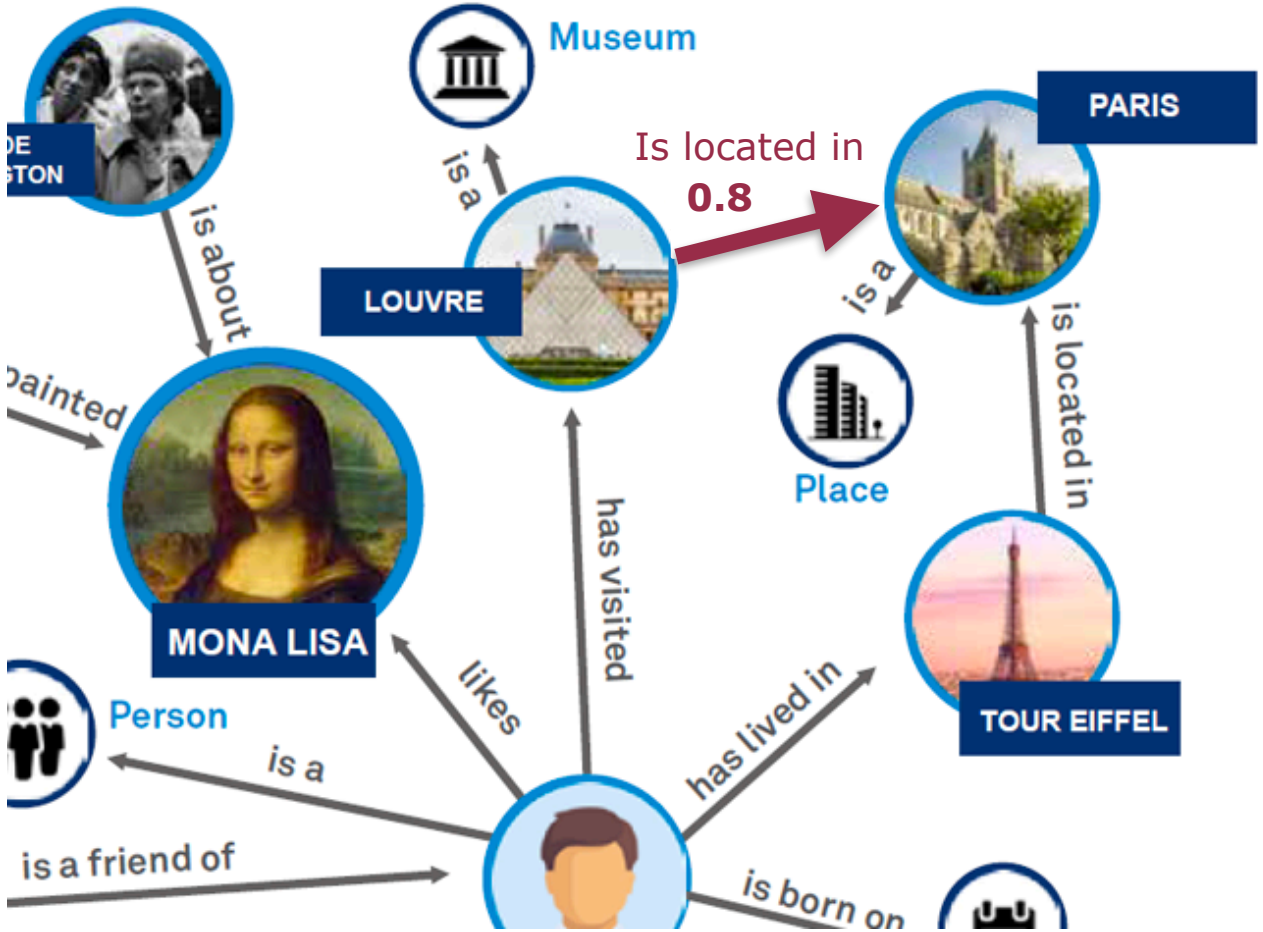
Link-Based Clustering — Identify groups of entities in relational data based on their similarity.

- Customer Segmentation
- Social Network Analysis
- Fraud Detection
- Collaboration Networks
- Biology and Medicine



Machine Learning on KGs

Link Prediction/Triple Classification — Predicting the existence (of likelihood of correctness) of edges in the graph. **Task:** Assign a score proportional to the likelihood that an unseen triple is true.



Machine Learning on KGs

Link Prediction/Triple Classification — Predicting the existence (of likelihood of correctness) of edges in the graph. **Task:** Assign a score proportional to the likelihood that an unseen triple is true.

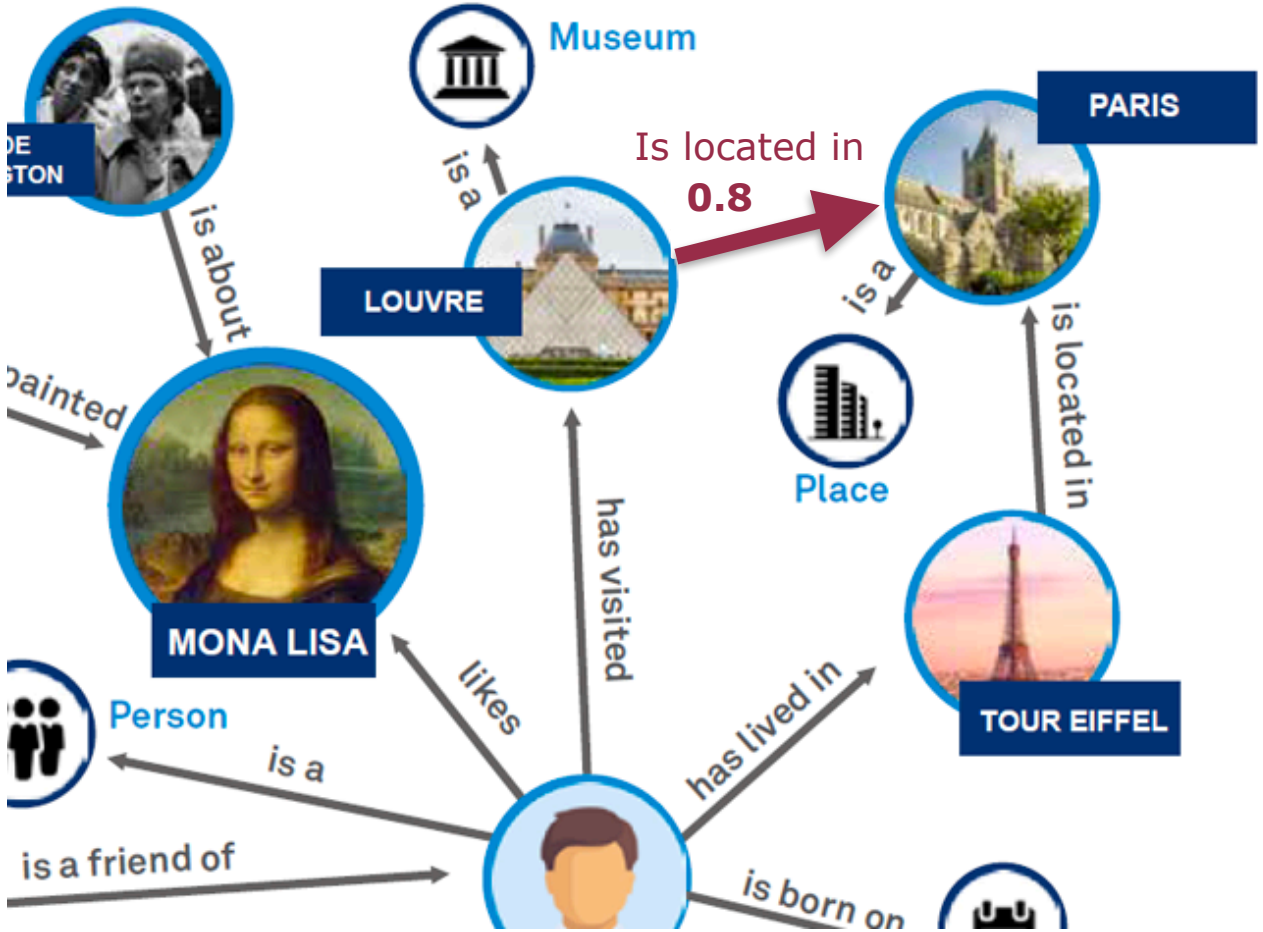
Link Prediction:

Learning to Rank problem

Information Retrieval Metrics

No ground truth negatives

$$\langle \text{Louvre}, \text{is located in}, X \rangle$$



Machine Learning on KGs

Link Prediction/Triple Classification — Predicting the existence (of likelihood of correctness) of edges in the graph. **Task:** Assign a score proportional to the likelihood that an unseen triple is true.

Link Prediction:

Learning to Rank problem

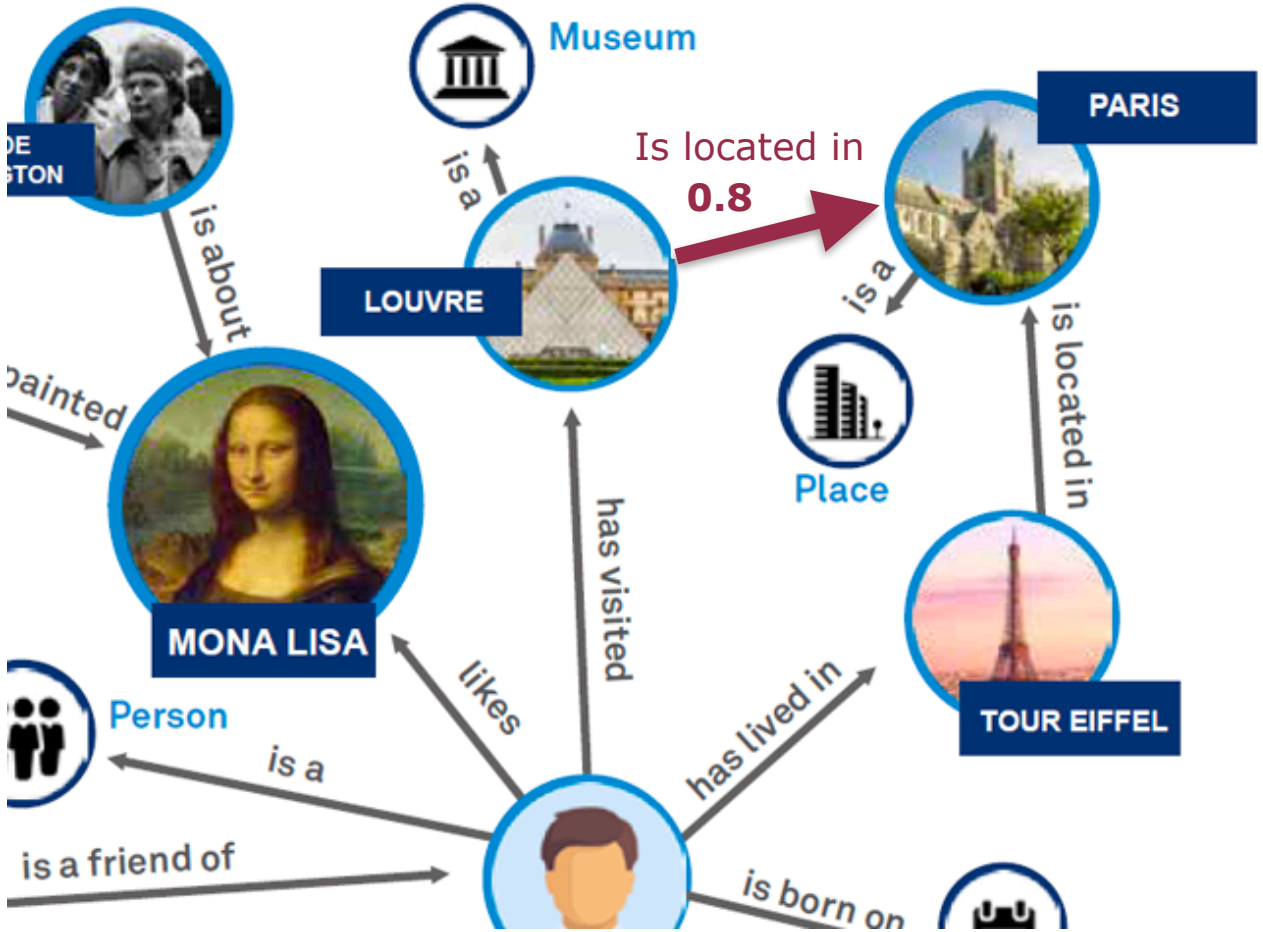
Information Retrieval Metrics

No ground truth negatives

Triple Classification:

Binary Classification Task/Metrics

Evaluation requires positive and ground-truth negative examples



Rule-Based Approaches

Logic Programming — Given a set of facts and rules, use *deductive reasoning* to infer new true facts.

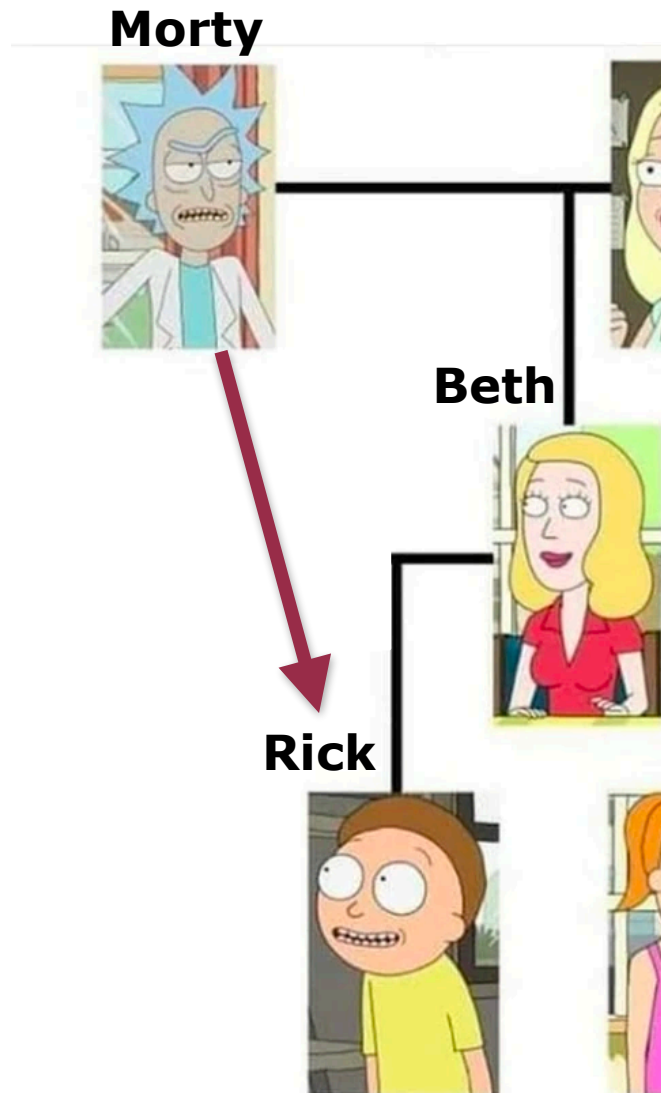
$$\text{parentOf}(\text{Beth}, \text{Rick})$$

$$\text{parentOf}(\text{Morty}, \text{Beth})$$

$$\text{grandParentOf}(X, Y) \leftarrow \text{parentOf}(X, Z) \wedge \text{parentOf}(Z, Y)$$

$$\Downarrow$$

$$\text{grandParentOf}(\text{Morty}, \text{Rick})$$



Rule-Based Approaches

Inductive **Logic Programming** — Given a set of (positive and negative) facts, identify the *rules* that explain the positive facts without explaining the negative ones

$\text{parentOf}(\text{Beth}, \text{Rick})$

$\text{parentOf}(\text{Morty}, \text{Beth})$

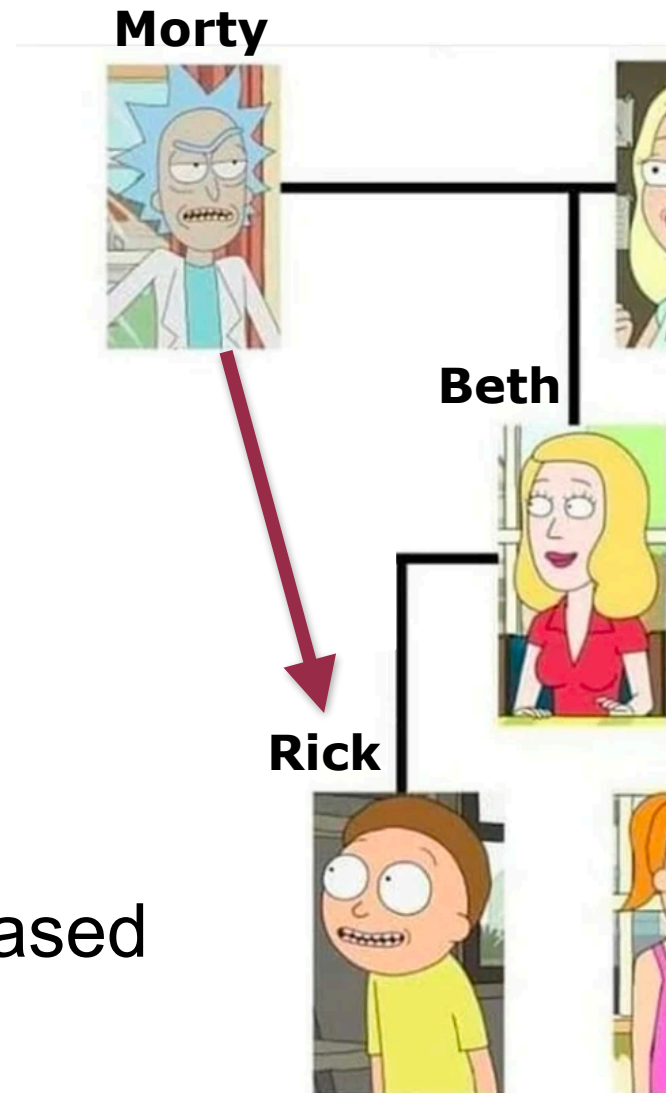
pos: $\text{grandParentOf}(\text{Morty}, \text{Rick})$

neg: $\text{grandParentOf}(\text{Beth}, \text{Rick})$

…

$$\Downarrow$$

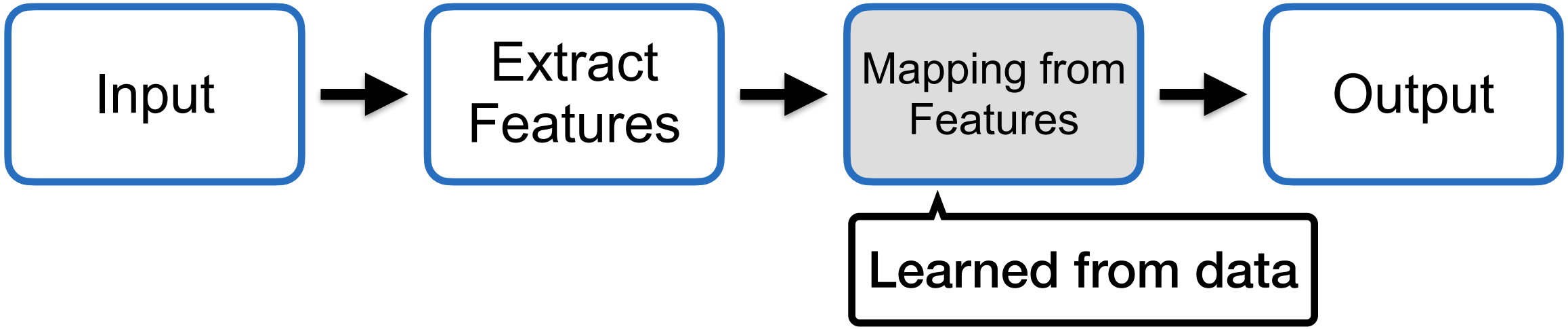
$$\text{grandParentOf}(X, Y) \leftarrow \text{parentOf}(X, Z) \wedge \text{parentOf}(Z, Y)$$



Rule Mining — (e.g., AMIE+, AnyBURL) extract rules based on the support in the KG

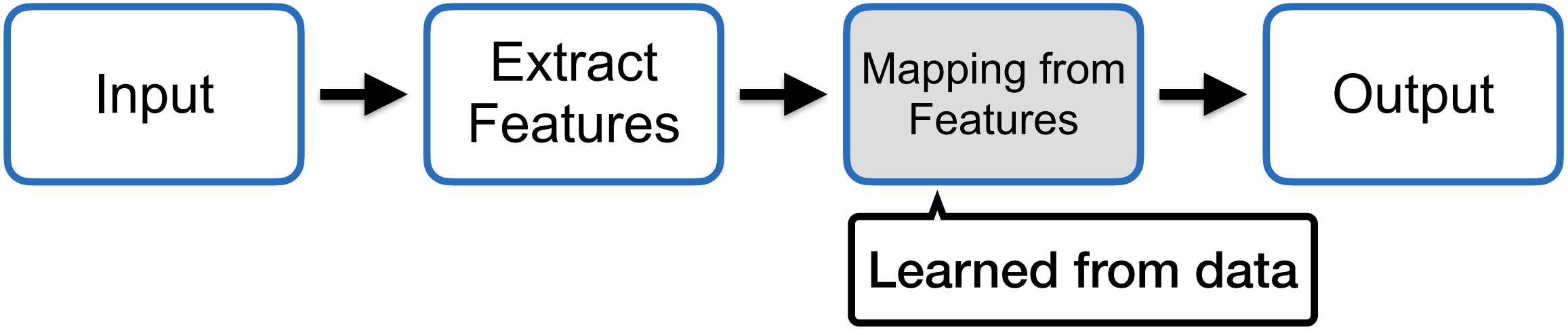
Introducing (Graph) Representation Learning

Classic Machine Learning Pipeline

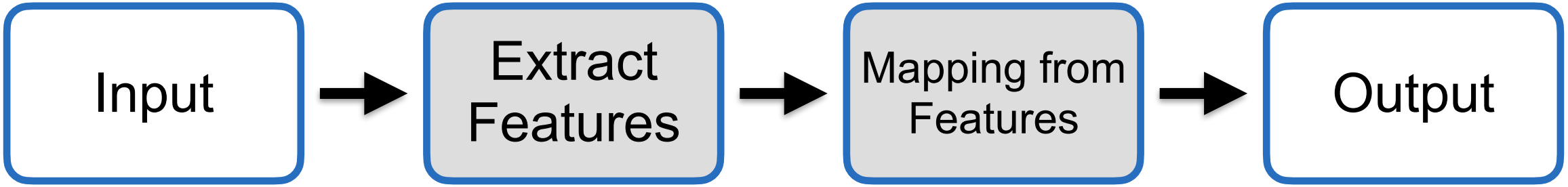


Introducing (Graph) Representation Learning

Classic Machine Learning Pipeline



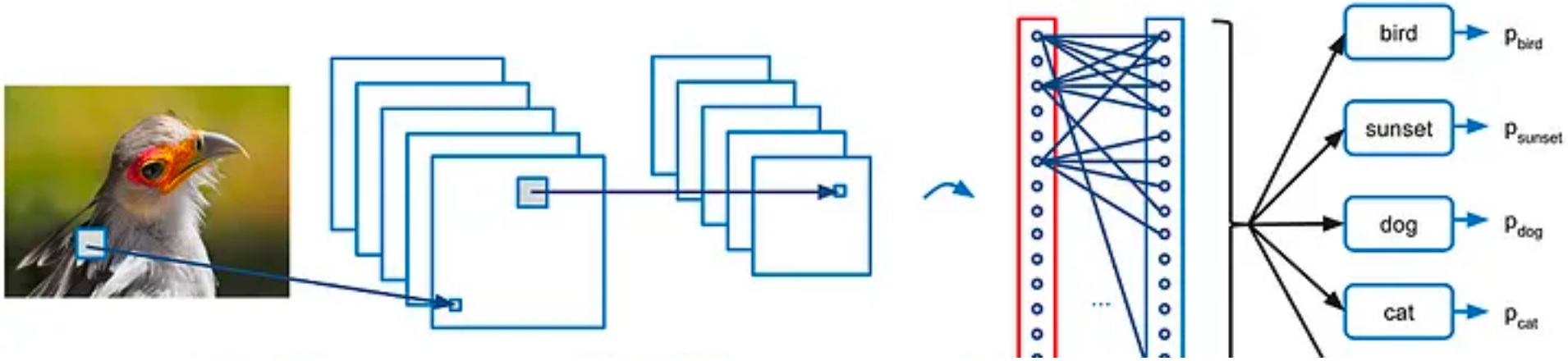
Representation Learning Pipeline



Graph Representation Learning

Can we use standard Representation Learning/Deep Learning tools?

It's tricky — CNNs are designed for grids (e.g., images), while RNNs and Transformers are designed for sequences (e.g., text)



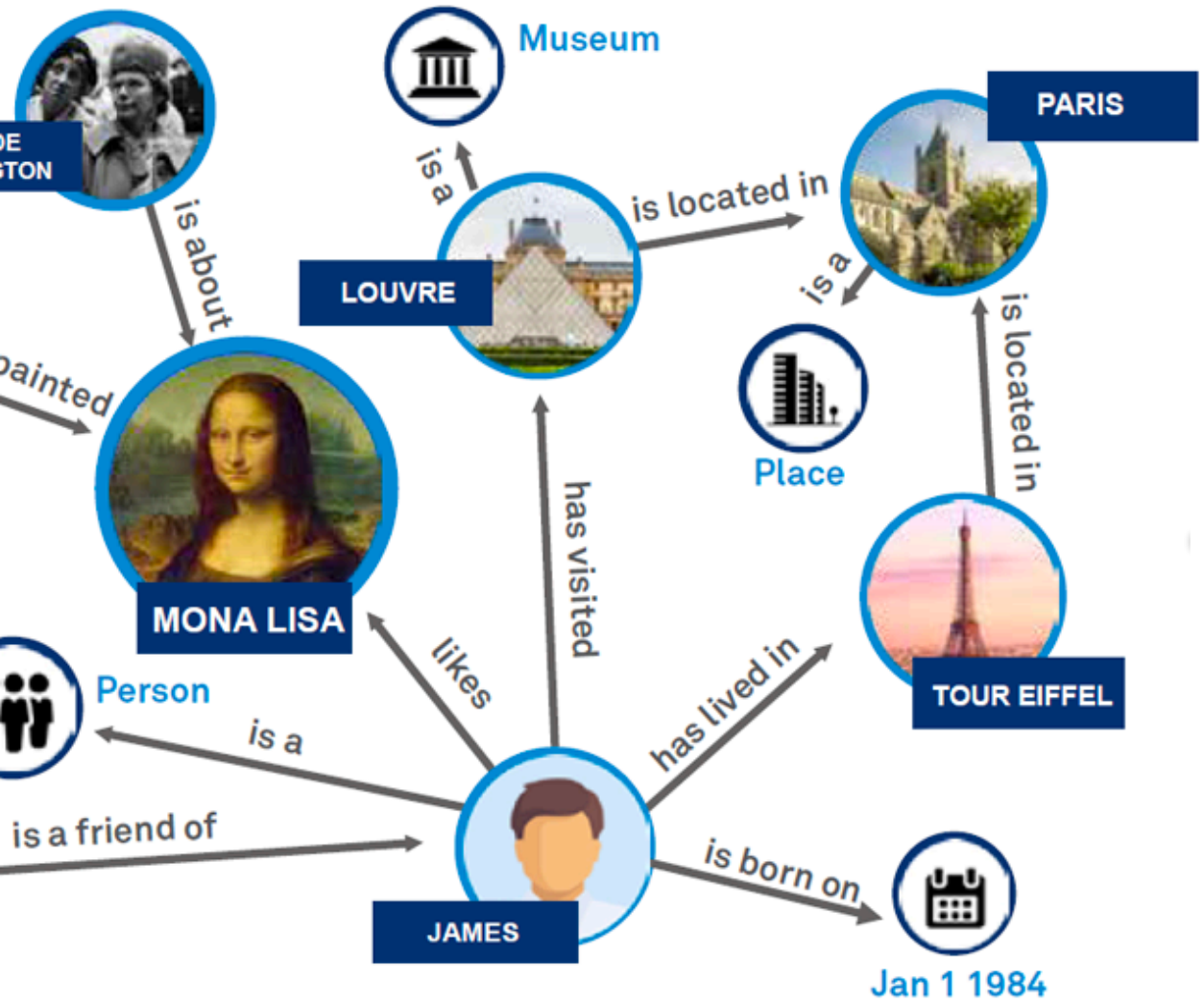
Knowledge Graphs, however, have different properties:

No spatial locality, no fixed-node ordering, multi-modal (concepts, textual descriptions, numbers, timestamps..)

We need ad-hoc models that can work with this type of data!

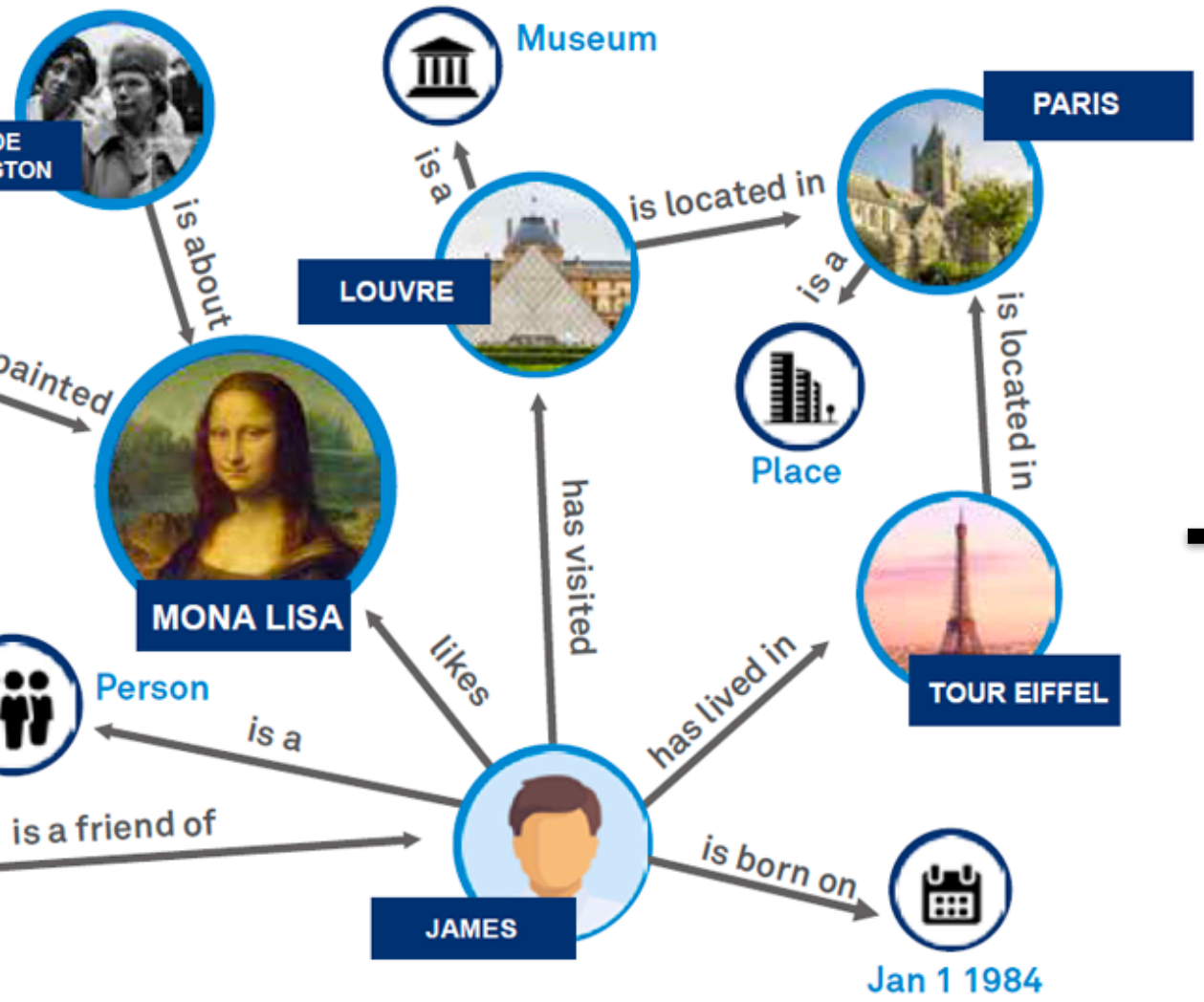
Graph Representation Learning

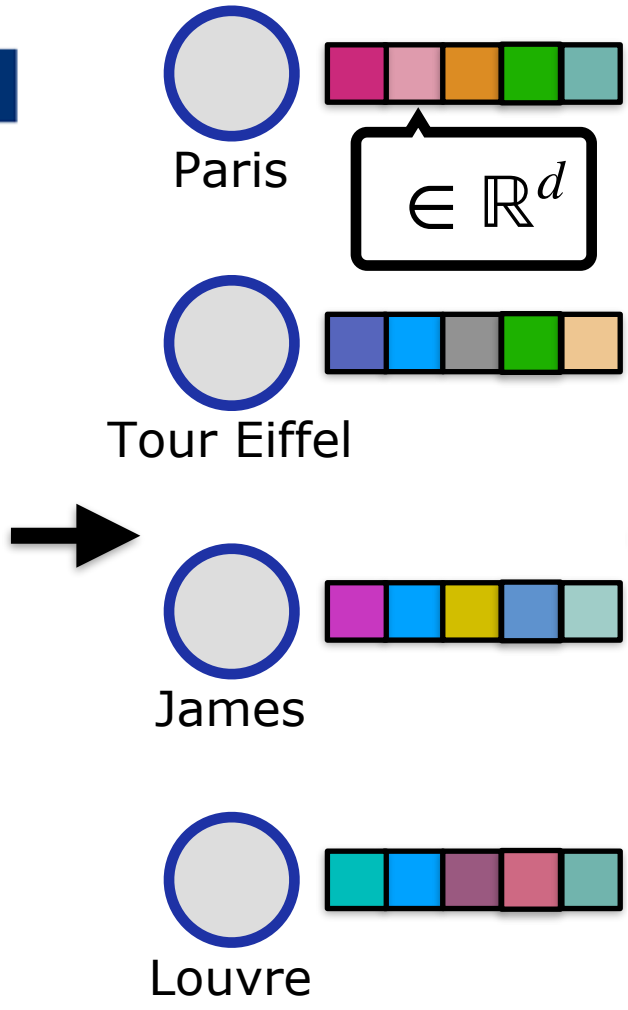
Idea — Learn representations of nodes and edges in the graph



Graph Representation Learning

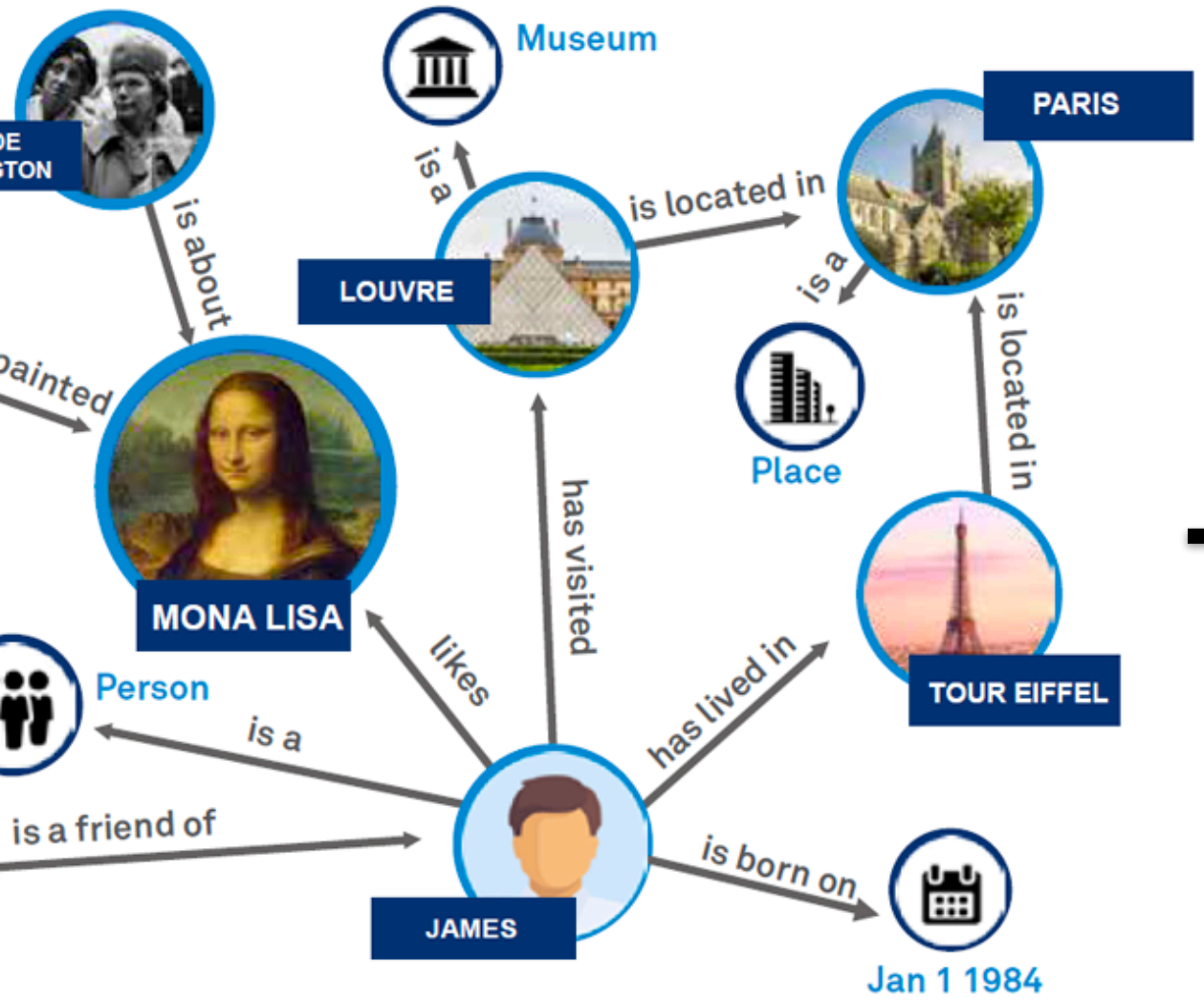
Idea — Learn representations of nodes and edges in the graph

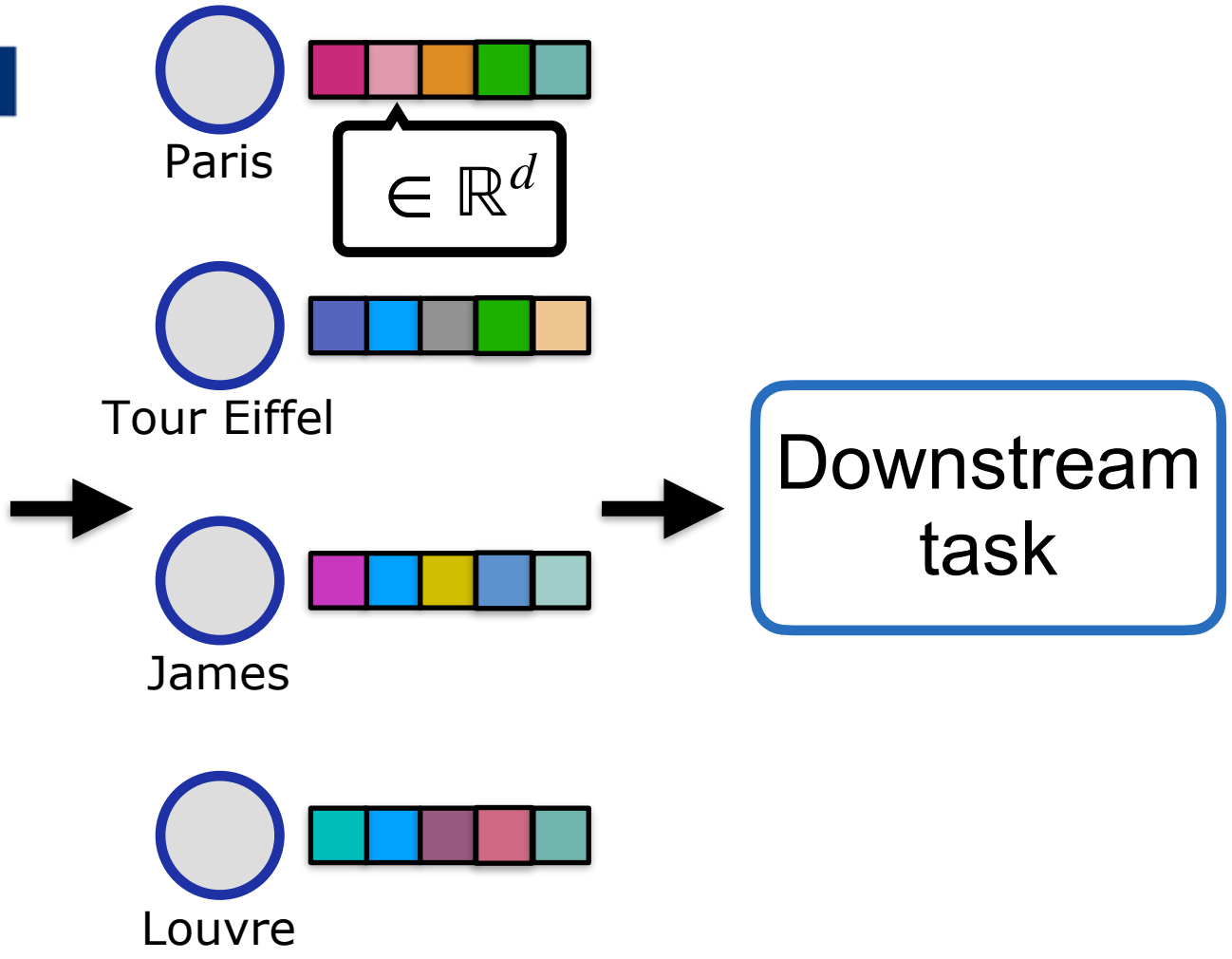




Graph Representation Learning

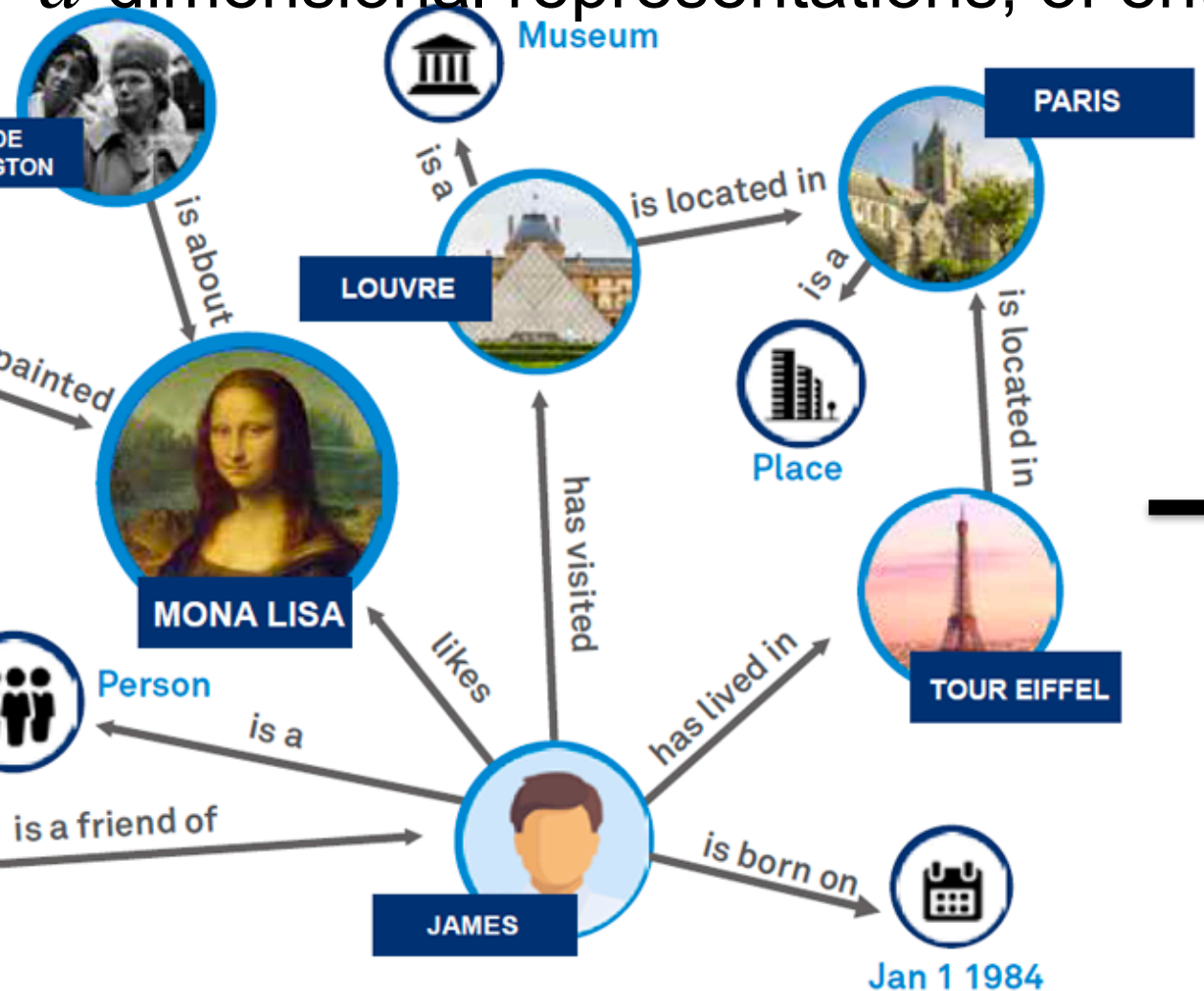
Idea — Learn representations of nodes and edges in the graph

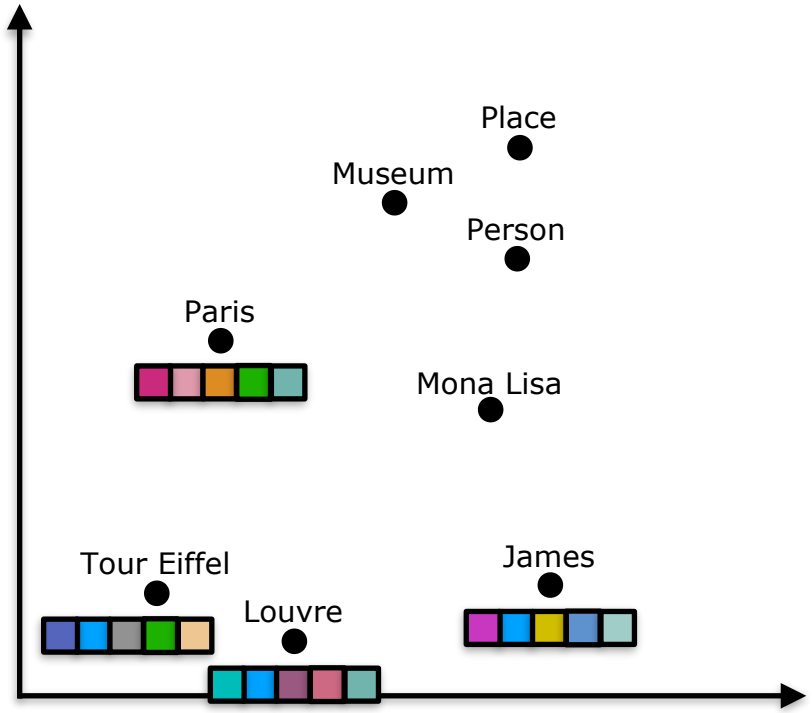




Knowledge Graph Embeddings

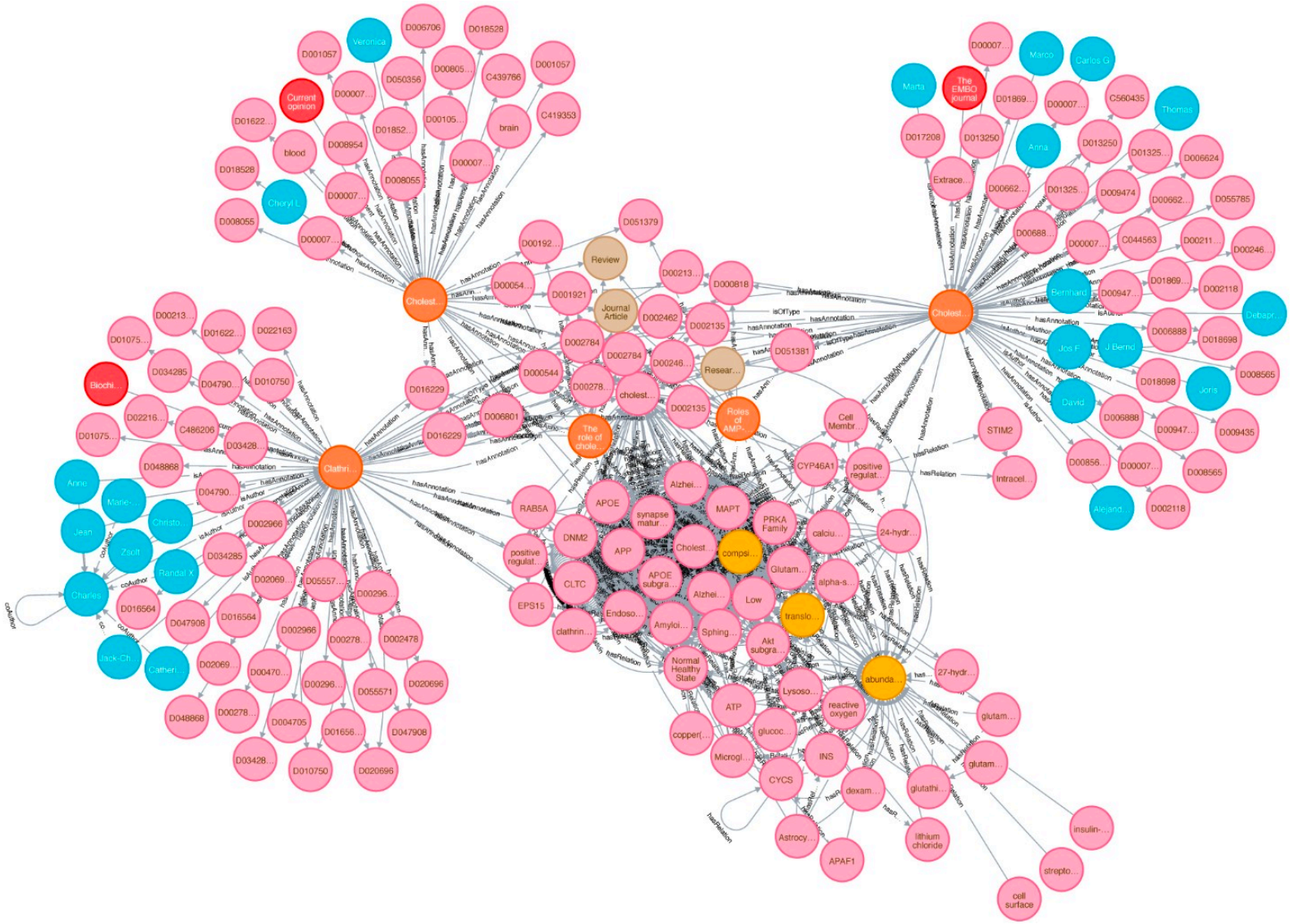
Knowledge Graph Embedding methods learn *embeddings*, i.e. continuous d-dimensional representations, of entities and relations in a KG





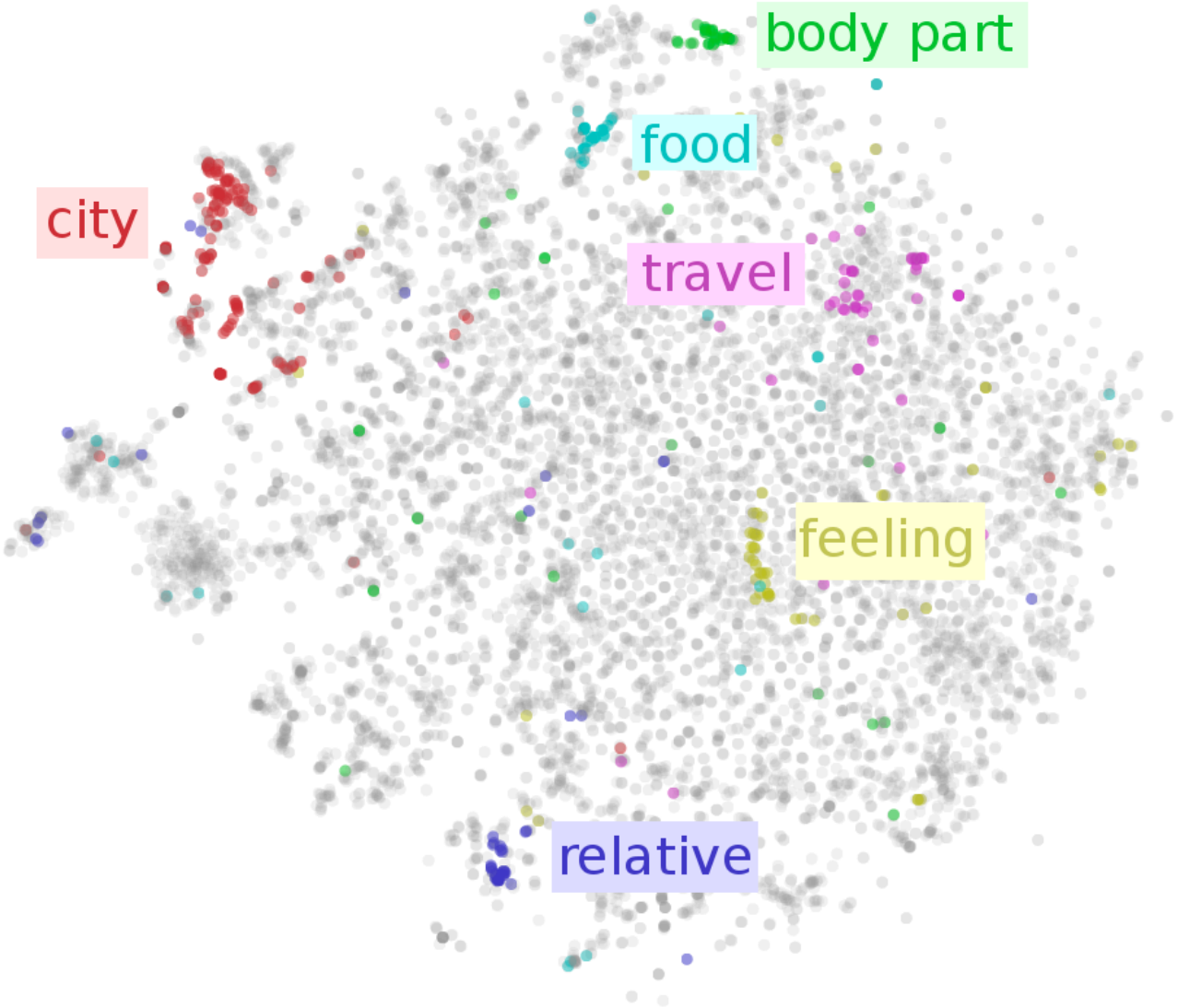
Knowledge Graph Embeddings

KGEs map nodes and edge types ..



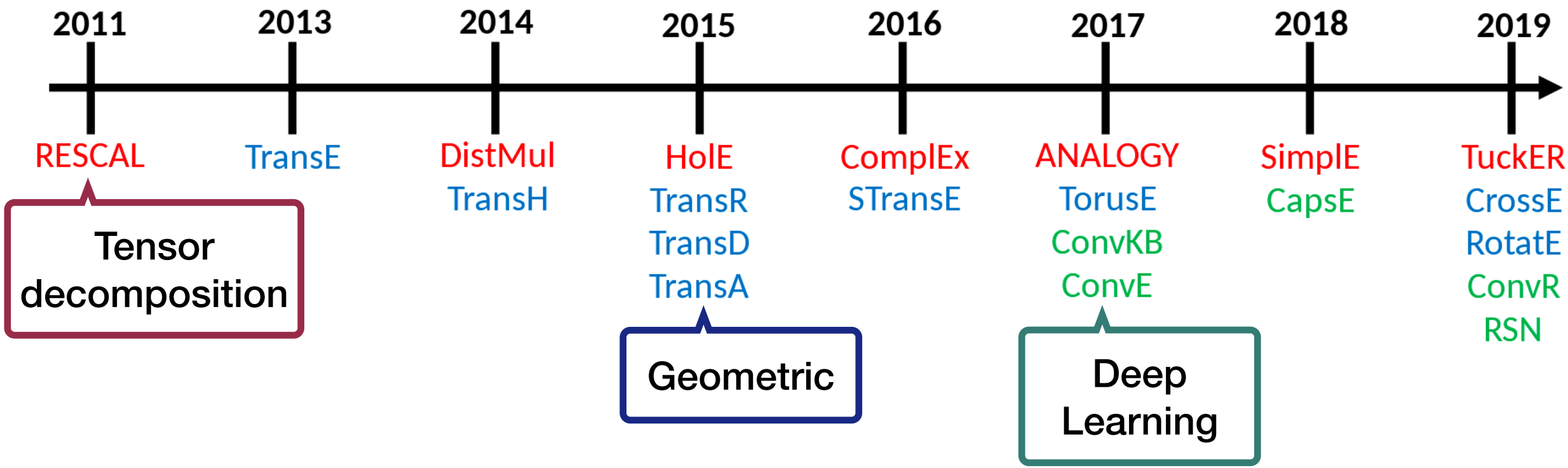
Knowledge Graph Embeddings

.. to semantically meaningful dense vector representations ($\in \mathbb{R}^d$)



Knowledge Graph Embeddings

Some KGE models in the recent published literature



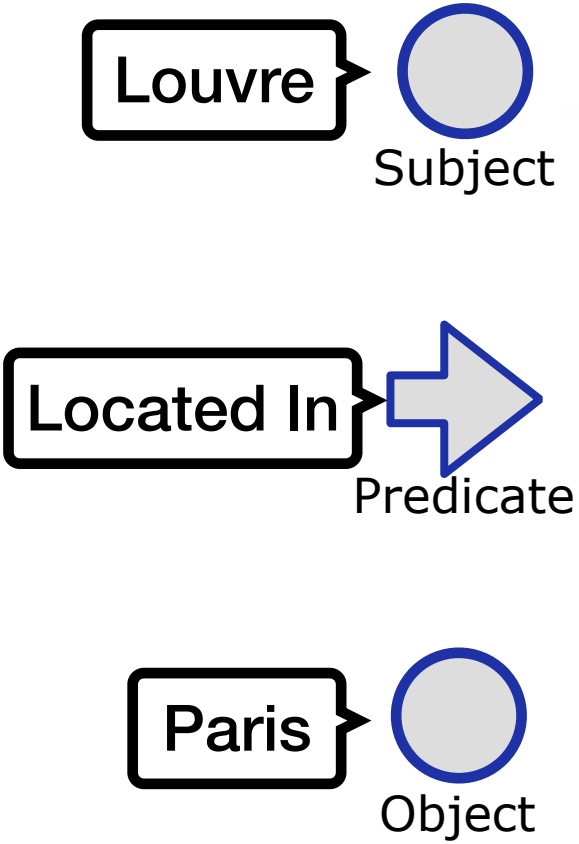
Must be able to capture graph properties — relation properties (e.g., (a)symmetry), hierarchies, type constraints, transitivity, homophily, long range dependencies, etc.

Knowledge Graph Embeddings

Model	Symmetry	Antisymmetry	Inversion	Composition
SE	✗	✗	✗	✗
TransE	✗	✓	✓	✓
TransX	✓	✓	✗	✗
DistMult	✓	✗	✗	✗
ComplEx	✓	✓	✓	✗
RotatE	✓	✓	✓	✓

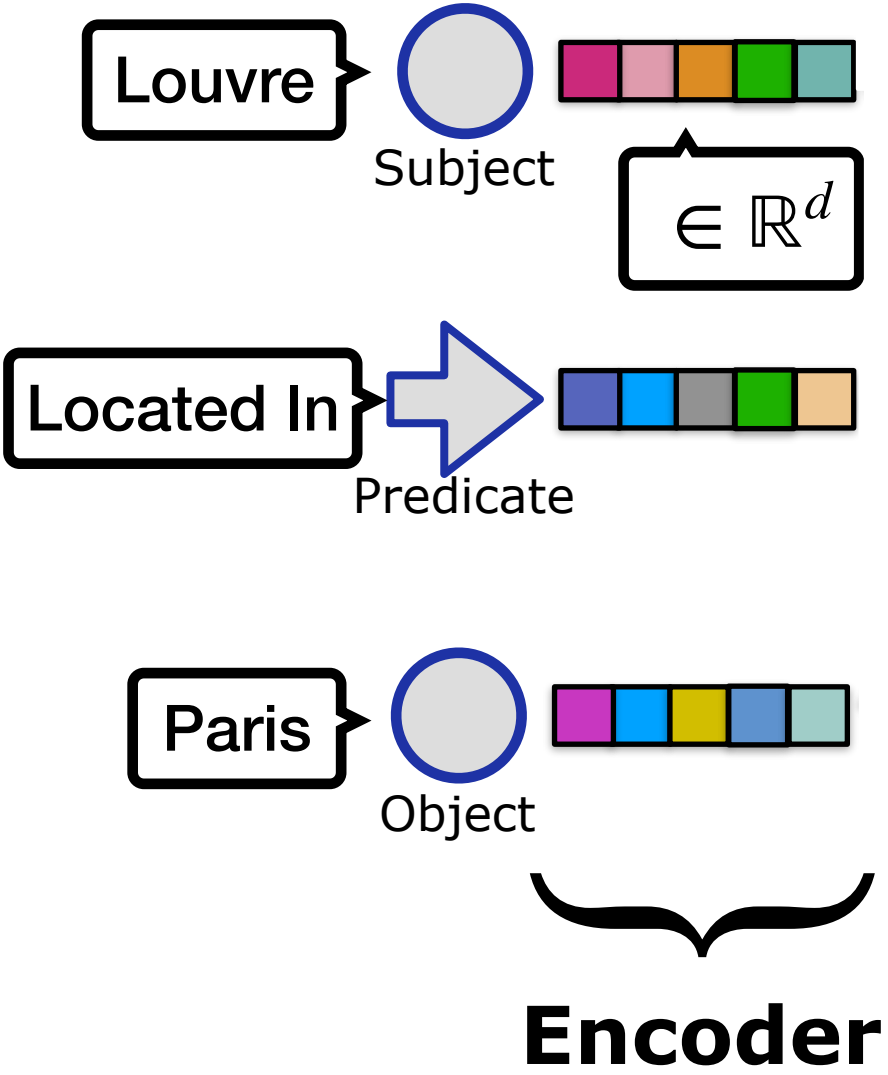
From Sun et al. - RotatE: Knowledge Graph Embeddings by Relational Rotation in Complex Space

Knowledge Graph Embeddings — Components

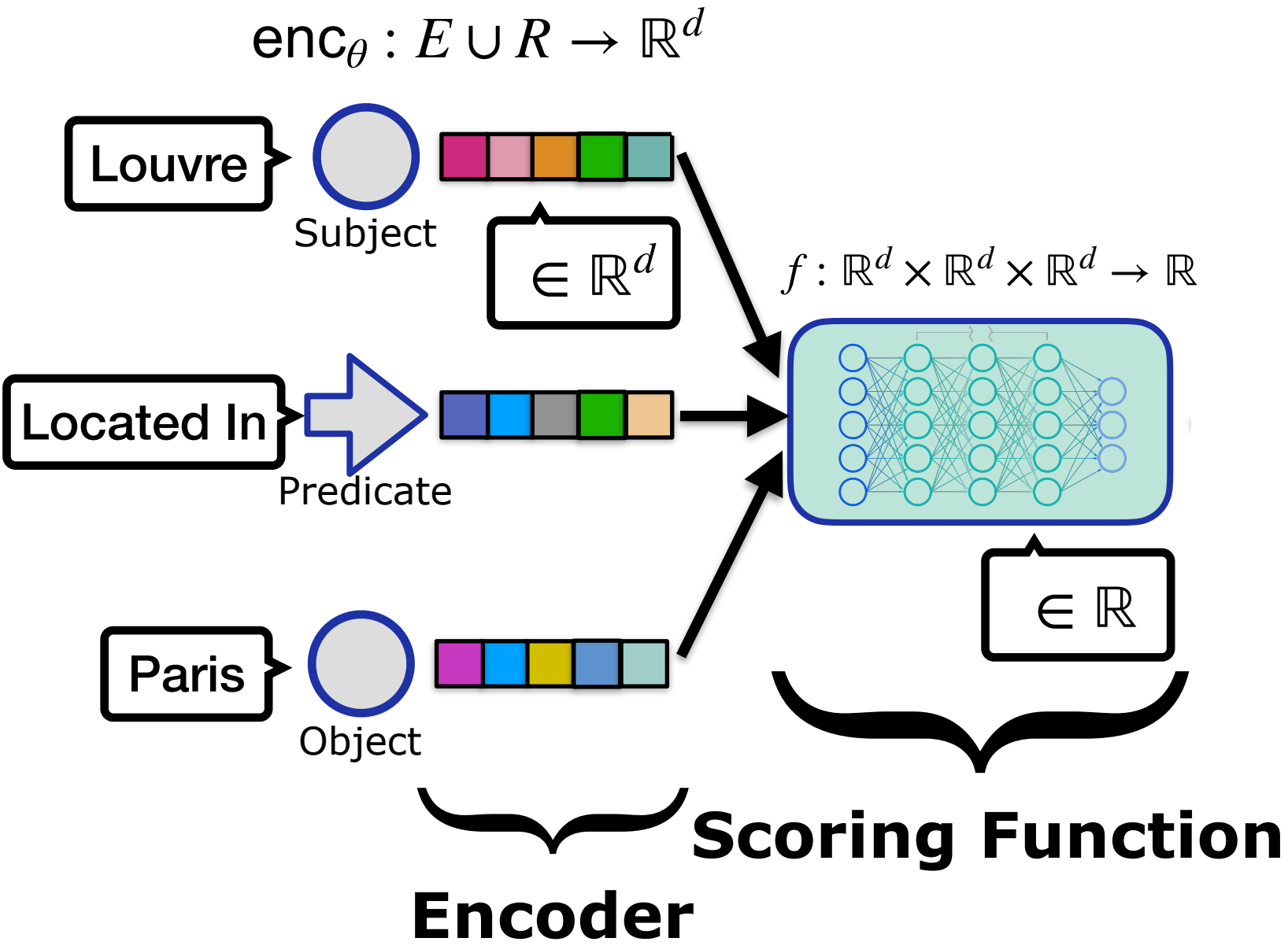


Knowledge Graph Embeddings — Components

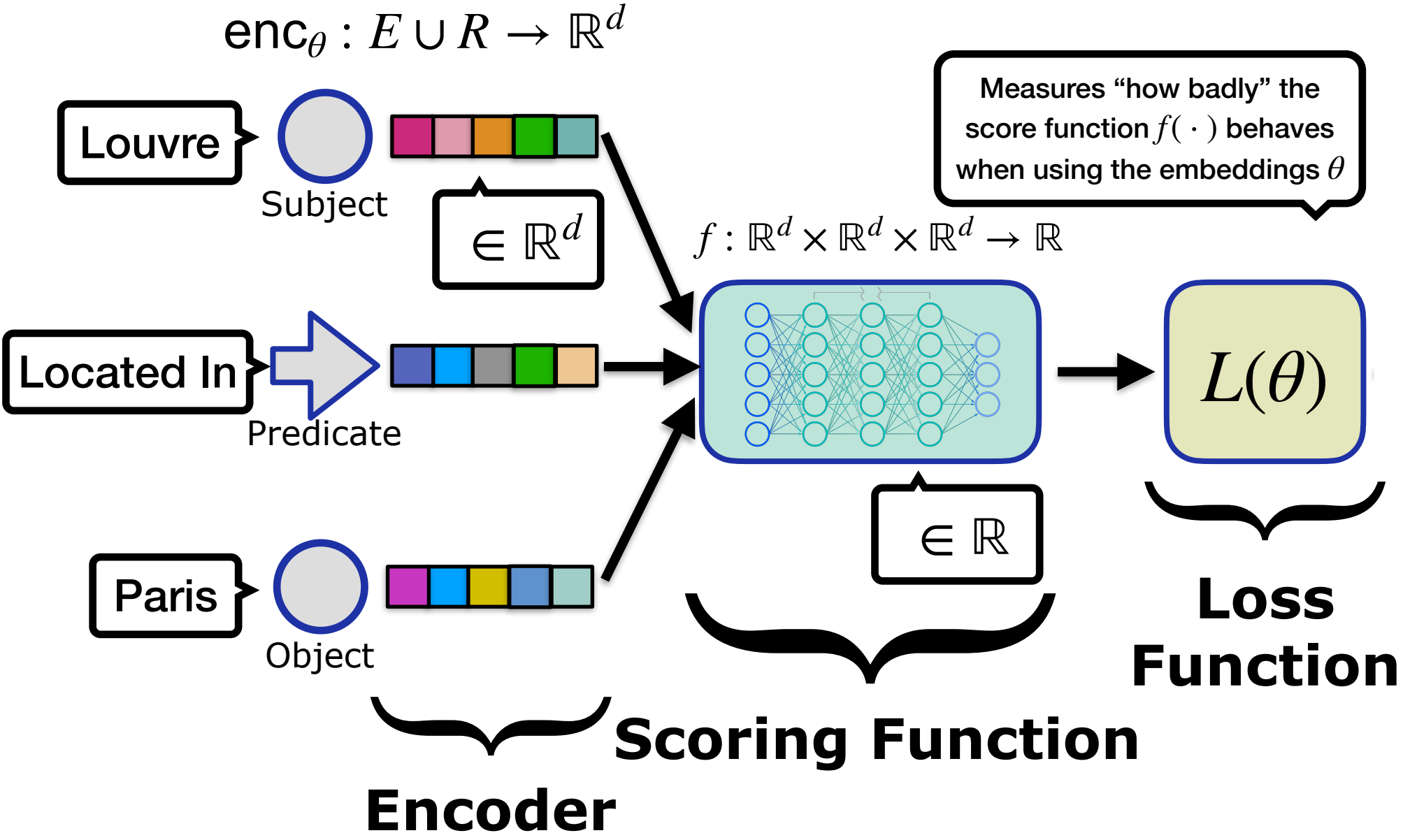
$\text{enc}_\theta : E \cup R \rightarrow \mathbb{R}^d$



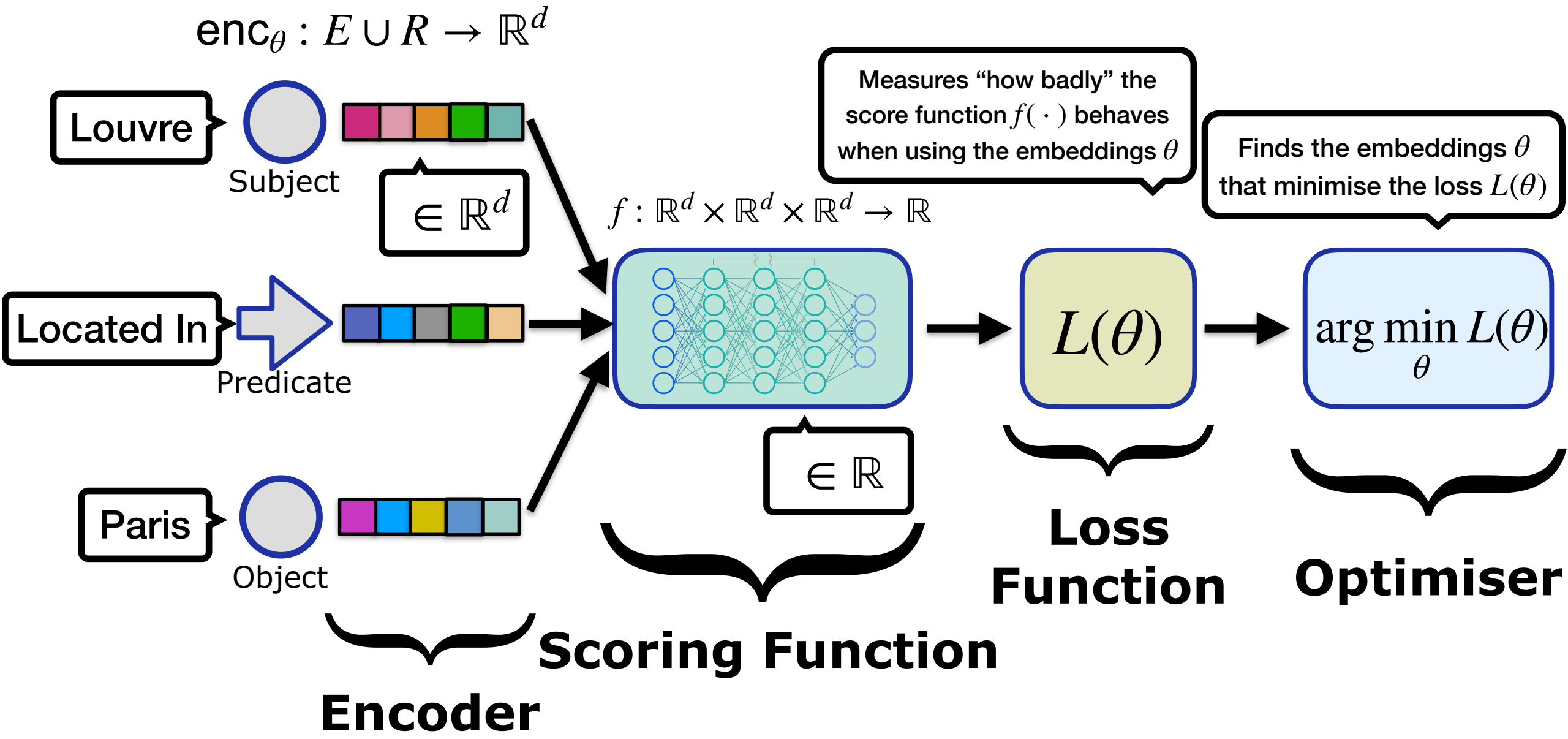
Knowledge Graph Embeddings — Components



Knowledge Graph Embeddings — Components



Knowledge Graph Embeddings — Components



Knowledge Graph Embeddings — Components

Encoder: Given an entity $e \in E$ (resp. predicate $p \in R$), $\text{enc}_\theta(\,\cdot\,)$ produces a dense vector representation $\mathbf{e} \in \mathbb{R}^d$ for e (resp. $\mathbf{p} \in \mathbb{R}^{d'}$ for p):

$$\text{enc}_\theta : E \to \mathbb{R}^d \quad \text{and} \quad \text{enc}_\theta : R \to \mathbb{R}^{d'}$$

Knowledge Graph Embeddings — Components

Encoder: Given an entity $e \in E$ (resp. predicate $p \in R$), $\text{enc}_\theta(\,\cdot\,)$ produces a dense vector representation $\mathbf{e} \in \mathbb{R}^d$ for e (resp. $\mathbf{p} \in \mathbb{R}^{d'}$ for p):

$$\text{enc}_\theta : E \to \mathbb{R}^d \quad \text{and} \quad \text{enc}_\theta : R \to \mathbb{R}^{d'}$$

Scoring Function: Given the representations $\mathbf{s} \in \mathbb{R}^d$, $\mathbf{p} \in \mathbb{R}^{d'}$, $\mathbf{o} \in \mathbb{R}^d$ of the subject s, predicate p, and object o of a triple $\langle s, p, o \rangle \in E \times R \times E$, calculates the likelihood that the fact holds true:

$$f : \mathbb{R}^d \times \mathbb{R}^{d'} \times \mathbb{R}^d \to \mathbb{R}$$

Knowledge Graph Embeddings — Components

Encoder: Given an entity $e \in E$ (resp. predicate $p \in R$), $\text{enc}_\theta(\cdot)$ produces a dense vector representation $\mathbf{e} \in \mathbb{R}^d$ for e (resp. $\mathbf{p} \in \mathbb{R}^{d'}$ for p):

$$\text{enc}_\theta : E \to \mathbb{R}^d \quad \text{and} \quad \text{enc}_\theta : R \to \mathbb{R}^{d'}$$

Scoring Function: Given the representations $\mathbf{s} \in \mathbb{R}^d$, $\mathbf{p} \in \mathbb{R}^{d'}$, $\mathbf{o} \in \mathbb{R}^d$ of the subject s, predicate p, and object o of a triple $\langle s, p, o \rangle \in E \times R \times E$, calculates the likelihood that the fact holds true:

$$f : \mathbb{R}^d \times \mathbb{R}^{d'} \times \mathbb{R}^d \to \mathbb{R}$$

Loss function/Optimiser: Given a KG $G = \{\langle s, p, o \rangle\} \subseteq E \times R \times E$, the loss function $L(G)$ measures the discrepancy between the scores generated using the encoder $\text{enc}_\theta(\cdot)$ and $f(\cdot)$, and the graph G. The optimiser finds θ^* such that:

$$\theta^* = \arg\min_\theta L(G)$$

Encoder

Encoder: Given an entity $e \in E$ (resp. predicate $p \in R$), $\text{enc}_\theta(\,\cdot\,)$ produces a dense vector representation $\mathbf{e} \in \mathbb{R}^d$ for e (resp. $\mathbf{p} \in \mathbb{R}^{d'}$ for p):

$$\text{enc}_\theta : E \rightarrow \mathbb{R}^d \quad \text{and} \quad \text{enc}_\theta : R \rightarrow \mathbb{R}^{d'}$$

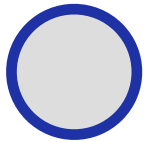
Encoder

Encoder: Given an entity $e \in E$ (resp. predicate $p \in R$), $\text{enc}_\theta(\,\cdot\,)$ produces a dense vector representation $\mathbf{e} \in \mathbb{R}^d$ for e (resp. $\mathbf{p} \in \mathbb{R}^{d'}$ for p):

$$\text{enc}_\theta : E \rightarrow \mathbb{R}^d \quad \text{and} \quad \text{enc}_\theta : R \rightarrow \mathbb{R}^{d'}$$

Simplest possible encoder: ***embedding layer*** — it stores all entity representations in a $\mathbf{E} \in \mathbb{R}^{|E| \times d}$ matrix, where each row is associated with a different entity $e \in E$:

Entity **Index** **Lookup**



Paris

index(Paris) $\rightarrow 6$

$\mathbf{e}_{\text{Paris}} = \mathbf{E}_{6,:} \in \mathbb{R}^d$

Scoring Functions

Scoring Function: Given the representations $\mathbf{s} \in \mathbb{R}^d$, $\mathbf{p} \in \mathbb{R}^{d'}$, $\mathbf{o} \in \mathbb{R}^d$ of the subject s, predicate p, and object o of a triple $\langle s, p, o \rangle \in E \times R \times E$, calculates the likelihood that the fact holds true:

$$f : \mathbb{R}^d \times \mathbb{R}^{d'} \times \mathbb{R}^d \rightarrow \mathbb{R}$$

Scoring Functions

Scoring Function: Given the representations $\mathbf{s} \in \mathbb{R}^d$, $\mathbf{p} \in \mathbb{R}^{d'}$, $\mathbf{o} \in \mathbb{R}^d$ of the subject s, predicate p, and object o of a triple $\langle s, p, o \rangle \in E \times R \times E$, calculates the likelihood that the fact holds true:

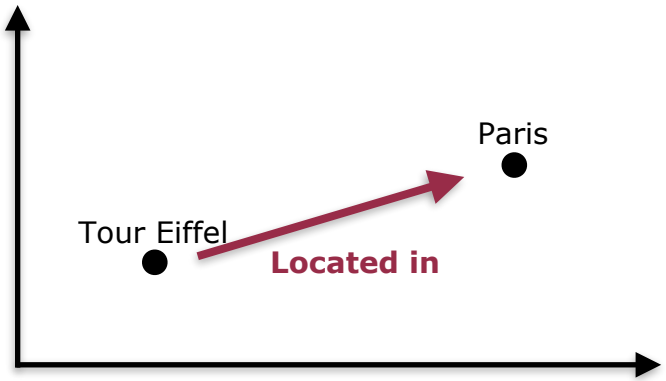
$$f : \mathbb{R}^d \times \mathbb{R}^{d'} \times \mathbb{R}^d \rightarrow \mathbb{R}$$

Many possible choices from the literature — a simple one is the **TransE** scoring function:

$$f(\mathbf{s}, \mathbf{p}, \mathbf{o}) = -\|(\mathbf{s} + \mathbf{p}) - \mathbf{o}\|_2$$

Translation of $\mathbf{s}$ with $\mathbf{p}$

Distance between $\mathbf{s} + \mathbf{p}$ and $\mathbf{o}$

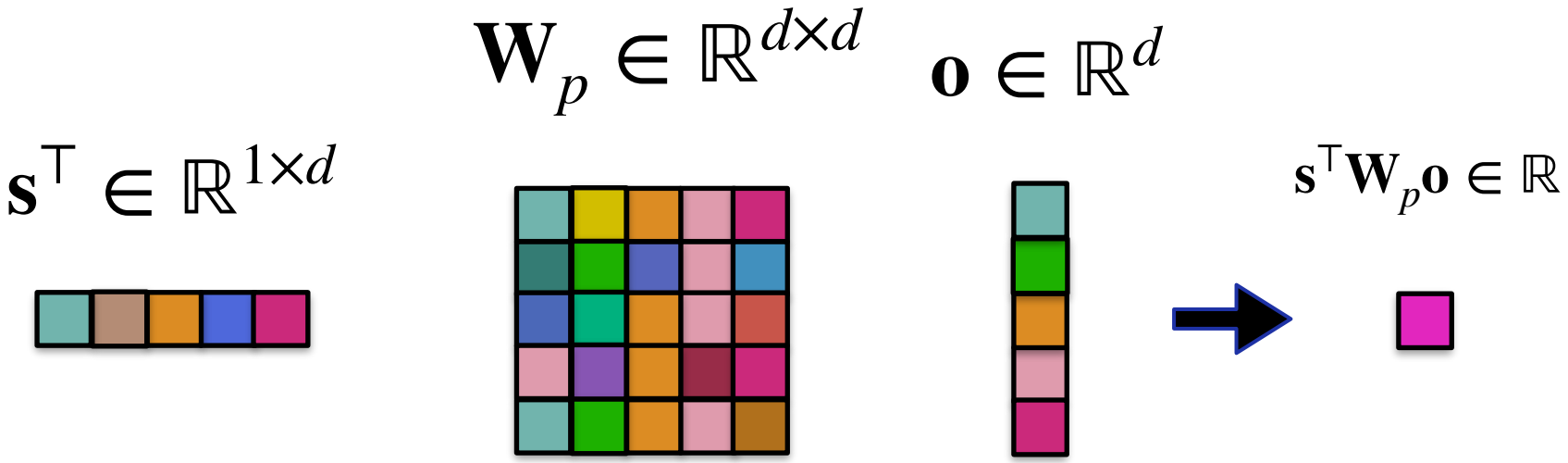


Bordes et al. Translating Embeddings for Modeling Multi-Relational Data. NIPS 2013

Scoring Functions

RESCAL: subject and object embeddings $\mathbf{s}, \mathbf{o} \in \mathbb{R}^d$, predicate embedding $\mathbf{W}_p \in \mathbb{R}^{d \times d}$

$$f(\mathbf{s}, \mathbf{W}_p, \mathbf{o}) = \mathbf{s}^\top \mathbf{W}_p \mathbf{o}$$



Nickel et al. A Three-Way Model for Collective Learning on Multi-Relational Data. ICML 2011

Scoring Functions

RESCAL: subject and object embeddings $\mathbf{s}, \mathbf{o} \in \mathbb{R}^d$, predicate embedding $\mathbf{W}_p \in \mathbb{R}^{d \times d}$

$$f(\mathbf{s}, \mathbf{W}_p, \mathbf{o}) = \mathbf{s}^\top \mathbf{W}_p \mathbf{o}$$

DistMult: subject, predicate, and object embeddings $\mathbf{s}, \mathbf{p}, \mathbf{o} \in \mathbb{R}^d$

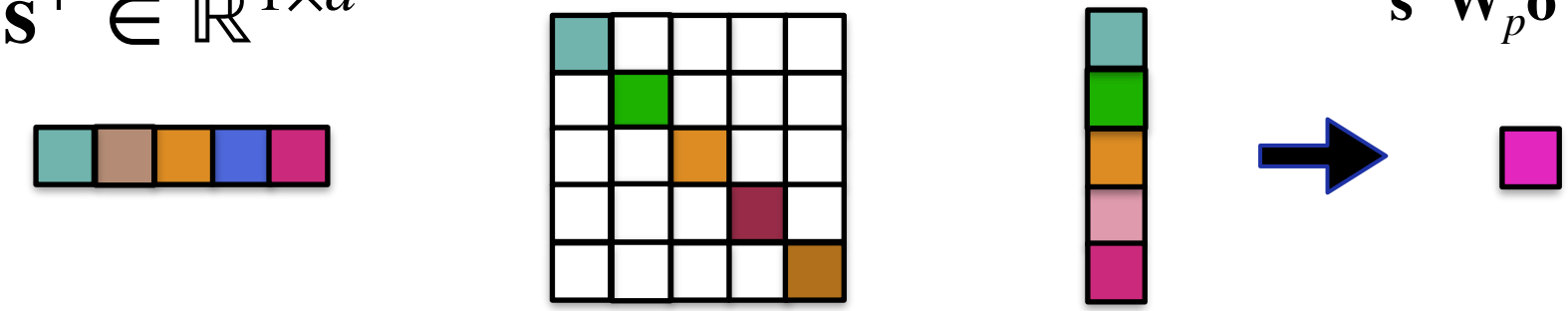
Equivalent to RESCAL with diagonal $\mathbf{W}_p$

$$f(\mathbf{s}, \mathbf{p}, \mathbf{o}) = \langle \mathbf{s}, \mathbf{p}, \mathbf{o} \rangle = \left(\mathbf{s} \odot \mathbf{p}\right)^\top \mathbf{o} = \sum_{i=1}^{d} \mathbf{s}_i \mathbf{p}_i \mathbf{o}_i$$

$\mathbf{W}_p \in \mathbb{R}^{d \times d}$ $\mathbf{o} \in \mathbb{R}^d$

$\mathbf{s}^\top \in \mathbb{R}^{1 \times d}$

$\mathbf{s}^\top \mathbf{W}_p \mathbf{o} \in \mathbb{R}$



Yang et al. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. ICLR 2015

Scoring Functions

RESCAL: subject and object embeddings $\mathbf{s}, \mathbf{o} \in \mathbb{R}^d$, predicate embedding $\mathbf{W}_p \in \mathbb{R}^{d \times d}$

$$f(\mathbf{s}, \mathbf{W}_p, \mathbf{o}) = \mathbf{s}^\top \mathbf{W}_p \mathbf{o}$$

DistMult: subject, predicate, and object embeddings $\mathbf{s}, \mathbf{p}, \mathbf{o} \in \mathbb{R}^d$

$$f(\mathbf{s}, \mathbf{p}, \mathbf{o}) = \langle \mathbf{s}, \mathbf{p}, \mathbf{o} \rangle = \left(\mathbf{s} \odot \mathbf{p}\right)^\top \mathbf{o} = \sum_{i=1}^{d} \mathbf{s}_i \mathbf{p}_i \mathbf{o}_i$$

```
# [B] Tensor
# .. and then compute the tri-linear dot product <s, p, o>, producing the resulting scores.
res = torch.sum(s_emb * p_emb * o_emb, 1)
```

Yang et al. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. ICLR 2015

Scoring Functions

RESCAL: subject and object embeddings $\mathbf{s}, \mathbf{o} \in \mathbb{R}^d$, predicate embedding $\mathbf{W}_p \in \mathbb{R}^{d \times d}$

$$f(\mathbf{s}, \mathbf{W}_p, \mathbf{o}) = \mathbf{s}^\top \mathbf{W}_p \mathbf{o}$$

DistMult: subject, predicate, and object embeddings $\mathbf{s}, \mathbf{p}, \mathbf{o} \in \mathbb{R}^d$

Equivalent to RESCAL with diagonal $\mathbf{W}_p$

$$f(\mathbf{s}, \mathbf{p}, \mathbf{o}) = \langle \mathbf{s}, \mathbf{p}, \mathbf{o} \rangle = \left(\mathbf{s} \odot \mathbf{p}\right)^\top \mathbf{o} = \sum_{i=1}^{d} \mathbf{s}_i \mathbf{p}_i \mathbf{o}_i$$

Problem — $\langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle$ and $\langle \text{Paris}, \text{locatedIn}, \text{Louvre} \rangle$ have the same score!

Scoring Functions

RESCAL: subject and object embeddings $\mathbf{s}, \mathbf{o} \in \mathbb{R}^d$, predicate embedding $\mathbf{W}_p \in \mathbb{R}^{d \times d}$

$$f(\mathbf{s}, \mathbf{W}_p, \mathbf{o}) = \mathbf{s}^\top \mathbf{W}_p \mathbf{o}$$

DistMult: subject, predicate, and object embeddings $\mathbf{s}, \mathbf{p}, \mathbf{o} \in \mathbb{R}^d$

Equivalent to RESCAL with diagonal $\mathbf{W}_p$

$$f(\mathbf{s}, \mathbf{p}, \mathbf{o}) = \langle \mathbf{s}, \mathbf{p}, \mathbf{o} \rangle = \left(\mathbf{s} \odot \mathbf{p}\right)^\top \mathbf{o} = \sum_{i=1}^{d} \mathbf{s}_i \mathbf{p}_i \mathbf{o}_i$$

Problem — $\langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle$ and $\langle \text{Paris}, \text{locatedIn}, \text{Louvre} \rangle$ have the same score!

ComplEx: *subject, predicate, and object embeddings* $\mathbf{s}, \mathbf{p}, \mathbf{o} \in \mathbb{C}^d$

$$f(\mathbf{s}, \mathbf{p}, \mathbf{o}) = \text{Re}\left(\langle \mathbf{s}, \mathbf{p}, \overline{\mathbf{o}} \rangle\right)$$

where $\text{Re}(\mathbf{x})$ denotes the real part of $\mathbf{x} \in \mathbb{C}^d$ and $\overline{\mathbf{x}} \in \mathbb{C}^d$ denotes its complex conjugate.

Scoring Functions

ComplEx: *subject, predicate, and object embeddings* $\mathbf{s}, \mathbf{p}, \mathbf{o} \in \mathbb{C}^d$

$$f(\mathbf{s}, \mathbf{p}, \mathbf{o}) = \text{Re}\left(\langle \mathbf{s}, \mathbf{p}, \overline{\mathbf{o}} \rangle\right)$$

where $\text{Re}(\mathbf{x})$ denotes the real part of $\mathbf{x} \in \mathbb{C}^d$ and $\overline{\mathbf{x}} \in \mathbb{C}^d$ denotes its complex conjugate.

```
# [B] Tensor
# This computes Re(<s, p, conj(o)>)
res = torch.sum((s_real * o_real + s_img * o_img) * p_real +
                (s_real * o_img - s_img * o_real) * p_img, 1)
```

Evaluation

Binary Classification Metrics — we have a set of test triples representing true and false facts (positive and negative examples); we try to classify each test triple in its correct class, and then calculate the accuracy of our classifier:

Evaluation

Binary Classification Metrics — we have a set of test triples representing true and false facts (positive and negative examples); we try to classify each test triple in its correct class, and then calculate the accuracy of our classifier:

$$
\begin{array}{ll}
+ & \langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle \\
- & \langle \text{Louvre}, \text{locatedIn}, \text{Edinburgh} \rangle \\
+ & \langle \text{Tour Eiffel}, \text{locatedIn}, \text{Paris} \rangle \\
- & \langle \text{Tour Eiffel}, \text{locatedIn}, \text{London} \rangle \\
 & \ldots
\end{array}
\Rightarrow
\text{Accuracy} = \frac{\#\ \text{Correct Classifications}}{\#\ \text{Test Triples}}
$$

Evaluation

Binary Classification Metrics — we have a set of test triples representing true and false facts (positive and negative examples); we try to classify each test triple in its correct class, and then calculate the accuracy of our classifier:

$$
\begin{array}{ll}
+ & \langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle \\
- & \langle \text{Louvre}, \text{locatedIn}, \text{Edinburgh} \rangle \\
+ & \langle \text{Tour Eiffel}, \text{locatedIn}, \text{Paris} \rangle \\
- & \langle \text{Tour Eiffel}, \text{locatedIn}, \text{London} \rangle \\
 & \cdots
\end{array}
\Rightarrow
\text{Accuracy} = \frac{\#\ \text{Correct Classifications}}{\#\ \text{Test Triples}}
$$

Requires translating the scores produced by $f(\cdot)$ in a binary decision — a way to do this is to find a threshold score that maximises the accuracy on some held-out validation set.

Evaluation

Information Retrieval Metrics — we have a set of test triples T representing true facts (positive examples). For each test triple we hide the object, and measure how well we can recover it using the model; then repeat the operation for the subject.

$\langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle$ →

Subject	Predicate	Object	Score	Rank
Louvre	locatedIn	London	0.4	2
Louvre	locatedIn	Edinburgh	0.2	3
Louvre	locatedIn	Paris	1.4	1
Louvre	locatedIn	Mars	-0.5	4
		..		

Evaluation

Information Retrieval Metrics — we have a set of test triples T representing true facts (positive examples). For each test triple we hide the object, and measure how well we can recover it using the model; then repeat the operation for the subject.

$\langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle$ →

Subject	Predicate	Object	Score	Rank
Louvre	locatedIn	London	0.4	2
Louvre	locatedIn	Edinburgh	0.2	3
Louvre	locatedIn	Paris	1.4	1
Louvre	locatedIn	Mars	-0.5	4

..

Mean Reciprocal Rank: $$\text{MRR} = \frac{1}{|T|} \sum_{t \in T} \frac{1}{\text{Rank}_t}$$

Evaluation

Information Retrieval Metrics — we have a set of test triples T representing true facts (positive examples). For each test triple we hide the object, and measure how well we can recover it using the model; then repeat the operation for the subject.

$\langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle$ →

Subject	Predicate	Object	Score	Rank
Louvre	locatedIn	London	0.4	2
Louvre	locatedIn	Edinburgh	0.2	3
Louvre	locatedIn	Paris	1.4	1
Louvre	locatedIn	Mars	-0.5	4

..

Mean Reciprocal Rank: $$\text{MRR} = \frac{1}{|T|} \sum_{t \in T} \frac{1}{\text{Rank}_t}$$

Hits@*k*: $$\text{Hits@}k = \frac{|\{t \in T \mid \text{Rank}_t \leq k\}|}{|T|}$$

Loss Functions

Loss function: Given a KG $G = \{\langle s, p, o \rangle\} \subseteq E \times R \times E$, the loss function $L(G)$ measures the discrepancy between the scores generated using the encoder $\text{enc}_\theta(\cdot)$ and the scoring function $f(\cdot)$, and the graph G.

Loss Functions

Loss function: Given a KG $G = \{\langle s, p, o \rangle\} \subseteq E \times R \times E$, the loss function $L(G)$ measures the discrepancy between the scores generated using the encoder $\text{enc}_\theta(\cdot)$ and the scoring function $f(\cdot)$, and the graph G.

In a nutshell — they measure to which extent triples in the graph receive lower scores than triples *not* in the graph. A first example is the *margin-based ranking loss*:

$$L(G) = \sum_{t \in G} \boxed{\sum_{t' \in N(t)} \left[\gamma - f(\mathbf{t}) + f(\mathbf{t}')\right]_+}$$

where $t \in G$ is a triple from the graph G, t' is a corruption of t obtained by changing either the subject or the object of t, and $\mathbf{t}$ denotes the embedding of t.

Bordes et al. Translating Embeddings for Modeling Multi-Relational Data. NIPS 2013

Loss Functions

Loss function: Given a KG $G = \{\langle s, p, o \rangle\} \subseteq E \times R \times E$, the loss function $L(G)$ measures the discrepancy between the scores generated using the encoder $\text{enc}_\theta(\cdot)$ and the scoring function $f(\cdot)$, and the graph G.

In a nutshell — they measure to which extent triples in the graph receive lower scores than triples *not* in the graph. A first example is the *margin-based ranking loss*:

$$L(G) = \sum_{t \in G} \boxed{\sum_{t' \in N(t)} \left[\gamma - f(\mathbf{t}) + f(\mathbf{t}')\right]_+}$$

Let's call this the per-triple loss $\ell(t)$

where $t \in G$ is a triple from the graph G, t' is a corruption of t obtained by changing either the subject or the object of t, and $\mathbf{t}$ denotes the embedding of t.

Bordes et al. Translating Embeddings for Modeling Multi-Relational Data. NIPS 2013

Negative Generation

Open World Assumption (OWA) — The absence of a particular statement means, in principle, that the statement has not been made explicitly yet, irrespective of whether it would be true or not. From the absence of a statement alone, we cannot infer that the statement is false.

Negative Generation

Open World Assumption (OWA) — The absence of a particular statement means, in principle, that the statement has not been made explicitly yet, irrespective of whether it would be true or not. From the absence of a statement alone, we cannot infer that the statement is false.

Local-Closed World Assumption (LCWA) — The KG is only "locally complete": if we observe a triple for a given entity $s \in E$ and predicate $p \in R$, we can assume that any non-existing triple $\langle s, p, \cdot \rangle$ represents a false fact.

Used to generate facts that are likely false (under some assumptions)

Nickel et al. A Review of Relational Machine Learning for Knowledge Graphs. IEEE 2015

Negative Generation

Open World Assumption (OWA) — The absence of a particular statement means, in principle, that the statement has not been made explicitly yet, irrespective of whether it would be true or not. From the absence of a statement alone, we cannot infer that the statement is false.

Local-Closed World Assumption (LCWA) — The KG is only "locally complete": if we observe a triple for a given entity $s \in E$ and predicate $p \in R$, we can assume that any non-existing triple $\langle s, p, \cdot \rangle$ represents a false fact.

Example: if we observe $\langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle \in G$, we can assume that the triple $\langle \text{Louvre}, \text{locatedIn}, \text{Edinburgh} \rangle \notin G$ denotes a false fact.

Nickel et al. A Review of Relational Machine Learning for Knowledge Graphs. IEEE 2015

Negative Generation

Local-Closed World Assumption (LCWA) — The KG is only "locally complete": if we observe a triple for a given entity $s \in E$ and predicate $p \in R$, we can assume that any non-existing triple $\langle s, p, \cdot \rangle$ represents a false fact.

Nickel et al. A Review of Relational Machine Learning for Knowledge Graphs. IEEE 2015

Negative Generation

Local-Closed World Assumption (LCWA) — The KG is only "locally complete": if we observe a triple for a given entity $s \in E$ and predicate $p \in R$, we can assume that any non-existing triple $\langle s, p, \cdot \rangle$ represents a false fact.

Example: if we observe $\langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle \in G$, we can assume that the triple $\langle \text{Louvre}, \text{locatedIn}, \text{Edinburgh} \rangle \notin G$ denotes a false fact.

Nickel et al. A Review of Relational Machine Learning for Knowledge Graphs. IEEE 2015

Negative Generation

Local-Closed World Assumption (LCWA) — The KG is only "locally complete": if we observe a triple for a given entity $s \in E$ and predicate $p \in R$, we can assume that any non-existing triple $\langle s, p, \cdot \rangle$ represents a false fact.

Example: if we observe $\langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle \in G$, we can assume that the triple $\langle \text{Louvre}, \text{locatedIn}, \text{Edinburgh} \rangle \notin G$ denotes a false fact.

Synthetic Negatives:

$$N\left(\langle s, p, o \rangle\right) \subseteq \{\langle \hat{s}, p, o \rangle \mid \hat{s} \in E, \langle \hat{s}, p, o \rangle \notin G\}$$

Sampled

$$\cup \{\langle s, p, \hat{o} \rangle \mid \hat{o} \in E, \langle s, p, \hat{o} \rangle \notin G\}$$

Nickel et al. A Review of Relational Machine Learning for Knowledge Graphs. IEEE 2015

Loss Functions

Margin-based Ranking Loss:

$$L(G) = \sum_{t \in G} \boxed{\sum_{t' \in N(t)} \left[\gamma - f(\mathbf{t}) + f(\mathbf{t}')\right]_+}$$

Let's call this the **per-triple loss** $\ell(t)$

where $t \in G$ is a triple from the graph G, and $\mathbf{t}$ denotes the embedding of t.

Logistic Loss:

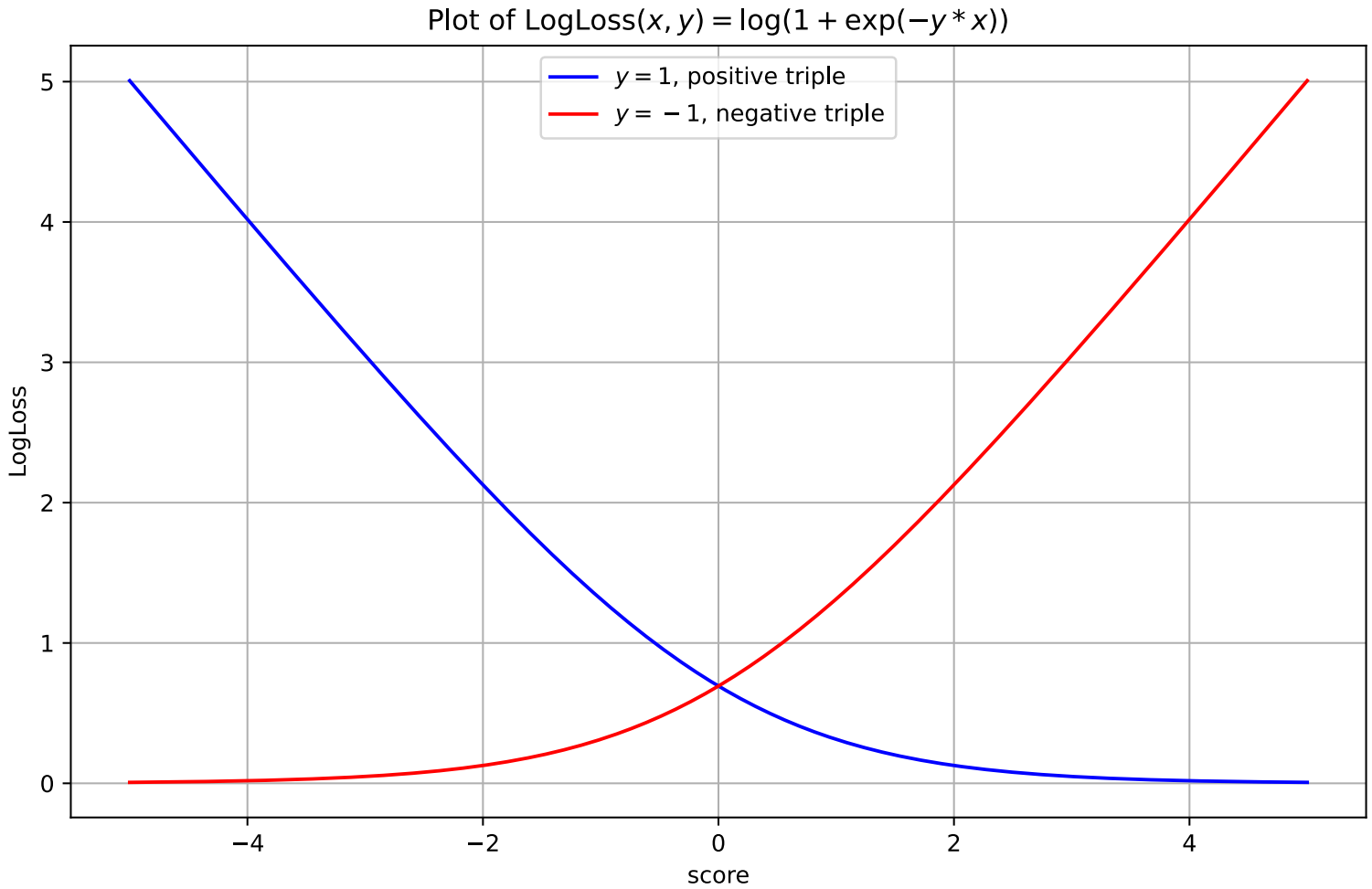
$$L(G) = \sum_{t \in G} \boxed{\text{LogLoss}(t, 1) + \sum_{t' \in N(t)} \text{LogLoss}(t', -1),}$$

where $\text{LogLoss}(t, y) = \log\left(1 + \exp\left(-y \cdot f(\mathbf{t})\right)\right)$

Logistic Loss:

$$L(G) = \sum_{t \in G} \boxed{\text{LogLoss}(t,1) + \sum_{t' \in N(t)} \text{LogLoss}(t', -1),}$$

$$\text{where LogLoss}(t, y) = \log\left(1 + \exp\left(-y \cdot f(\mathbf{t})\right)\right)$$



Loss Functions

K-versus-All (KvsAll) Loss:

$$L(G) = \sum_{\langle s,p,\cdot\rangle \in G} \sum_{\hat{o} \in E} \text{LogLoss}(\langle s, p, \hat{o}\rangle, y_{sp\hat{o}}), \text{ where } y_{sp\hat{o}} = \begin{cases} 1 & \text{if } \langle s, p, \hat{o}\rangle \in G \\ -1 & \text{otherwise} \end{cases}$$

Loss Functions

K-versus-All (KvsAll) Loss:

$$L(G) = \sum_{\langle s,p,\cdot\rangle \in G} \sum_{\hat{o} \in E} \text{LogLoss}(\langle s, p, \hat{o}\rangle, y_{sp\hat{o}}), \text{ where } y_{sp\hat{o}} = \begin{cases} 1 & \text{if } \langle s, p, \hat{o}\rangle \in G \\ -1 & \text{otherwise} \end{cases}$$

One-versus-All (1vsAll) Loss:

$$L(G) = \sum_{\langle s,p,o\rangle \in G} -f\left(\mathbf{s}, \mathbf{p}, \mathbf{o}\right) + \log\left(\sum_{\hat{o} \in E} \exp\left(f(\mathbf{s}, \mathbf{p}, \hat{\mathbf{o}})\right)\right)$$

Loss Functions

Why tho!? 🤔

One-versus-All (1vsAll) Loss:

$$L(G) = \sum_{\langle s,p,o \rangle \in G} \boxed{-f\left(\mathbf{s}, \mathbf{p}, \mathbf{o}\right) + \log\left(\sum_{\hat{o} \in E} \exp\left(f(\mathbf{s}, \mathbf{p}, \hat{\mathbf{o}})\right)\right)}$$

Loss Functions

Why tho!? 🤔

One-versus-All (1vsAll) Loss:

$$L(G) = \sum_{\langle s,p,o \rangle \in G} \boxed{-f\left(\mathbf{s}, \mathbf{p}, \mathbf{o}\right) + \log\left(\sum_{\hat{o} \in E} \exp\left(f(\mathbf{s}, \mathbf{p}, \hat{\mathbf{o}})\right)\right)}$$

$$P\left(o \in E \mid \langle s, p, \cdot \rangle\right) = \text{softmax}\left(f\left(\mathbf{s}, \mathbf{p}, \cdot\right)\right)_o$$

Loss Functions

Why tho!? 🤔

One-versus-All (1vsAll) Loss:

$$L(G) = \sum_{\langle s,p,o \rangle \in G} \boxed{-f\left(\mathbf{s}, \mathbf{p}, \mathbf{o}\right) + \log\left(\sum_{\hat{o} \in E} \exp\left(f(\mathbf{s}, \mathbf{p}, \hat{\mathbf{o}})\right)\right)}$$

$$P\left(o \in E \mid \langle s, p, \cdot \rangle\right) = \text{softmax}\left(f\left(\mathbf{s}, \mathbf{p}, \cdot\right)\right)_o$$

$$= \frac{\exp\left(f\left(\mathbf{s}, \mathbf{p}, \mathbf{o}\right)\right)}{\sum_{\hat{o} \in E} \exp\left(f\left(\mathbf{s}, \mathbf{p}, \hat{\mathbf{o}}\right)\right)}$$

Loss Functions

Why tho!? 🤔

One-versus-All (1vsAll) Loss:

$$L(G) = \sum_{\langle s,p,o \rangle \in G} \boxed{-f\left(\mathbf{s}, \mathbf{p}, \mathbf{o}\right) + \log\left(\sum_{\hat{o} \in E} \exp\left(f(\mathbf{s}, \mathbf{p}, \hat{\mathbf{o}})\right)\right)}$$

$$P\left(o \in E \mid \langle s, p, \cdot \rangle\right) = \text{softmax}\left(f\left(\mathbf{s}, \mathbf{p}, \cdot\right)\right)_o$$

$$= \frac{\exp\left(f\left(\mathbf{s}, \mathbf{p}, \mathbf{o}\right)\right)}{\sum_{\hat{o} \in E} \exp\left(f\left(\mathbf{s}, \mathbf{p}, \hat{\mathbf{o}}\right)\right)}$$

Friendly reminder that $\log \frac{a}{b} = \log a - \log b$ and $\log\left(\exp(a)\right) = a$

$$L(G) = \sum_{\langle s,p,o \rangle \in G} \boxed{-\log P\left(o \mid \langle s, p, \cdot \rangle\right)}$$

Negative Log-Likelihood!

Minimising a Loss Function

Optimiser: Given a KG $G = \{\langle s, p, o \rangle\} \subseteq E \times R \times E$ and a loss function $L(G)$ for an encoder $\text{enc}_\theta(\cdot)$ and scoring function $f(\cdot)$, the optimiser finds θ^* such that:

$$\theta^* = \arg\min_\theta L(G)$$

Gradient Descent in a Nutshell

Problem: We have a function $L : \mathbb{R}^n \mapsto \mathbb{R}$ that, given some input $\theta \in \mathbb{R}^n$ (e.g., the model parameters), returns a scalar $L(\theta) \in \mathbb{R}$ (e.g., a loss value).

We want to find the input $\theta^* \in \mathbb{R}^n$ that minimises L, i.e., $\boxed{\theta^* = \arg\max_{\theta} L(\theta).}$

Gradient Descent in a Nutshell

Problem: We have a function $L : \mathbb{R}^n \mapsto \mathbb{R}$ that, given some input $\theta \in \mathbb{R}^n$ (e.g., the model parameters), returns a scalar $L(\theta) \in \mathbb{R}$ (e.g., a loss value).

We want to find the input $\theta^* \in \mathbb{R}^n$ that minimises L, i.e., $\boxed{\theta^* = \arg\max_{\theta} L(\theta).}$

Gradient Descent works as follows:

1. Initialise $\theta^{(0)} \in \mathbb{R}^n$ at random

Gradient Descent in a Nutshell

Problem: We have a function $L : \mathbb{R}^n \mapsto \mathbb{R}$ that, given some input $\theta \in \mathbb{R}^n$ (e.g., the model parameters), returns a scalar $L(\theta) \in \mathbb{R}$ (e.g., a loss value).

We want to find the input $\theta^* \in \mathbb{R}^n$ that minimises L, i.e., $\boxed{\theta^* = \arg\max_{\theta} L(\theta).}$

Gradient Descent works as follows:

1. Initialise $\theta^{(0)} \in \mathbb{R}^n$ at random

2. Until convergence, for $i = 1, \ldots, n$:

Gradient Descent in a Nutshell

Problem: We have a function $L : \mathbb{R}^n \mapsto \mathbb{R}$ that, given some input $\theta \in \mathbb{R}^n$ (e.g., the model parameters), returns a scalar $L(\theta) \in \mathbb{R}$ (e.g., a loss value).

We want to find the input $\theta^* \in \mathbb{R}^n$ that minimises L, i.e., $\boxed{\theta^* = \arg\max_{\theta} L(\theta).}$

Gradient Descent works as follows:

1. Initialise $\theta^{(0)} \in \mathbb{R}^n$ at random

2. Until convergence, for $i = 1, \ldots, n$:

   - $\theta^{(i)} = \theta^{(i-1)} - \underbrace{\boxed{\eta}}_{\text{Learning rate}} \boxed{\nabla_\theta L\left(\theta^{(i-1)}\right)}$ The function $L(\cdot)$ should be differentiable

Gradient Descent in a Nutshell

Problem: We have a function $L : \mathbb{R}^n \mapsto \mathbb{R}$ that, given some input $\theta \in \mathbb{R}^n$ (e.g., the model parameters), returns a scalar $L(\theta) \in \mathbb{R}$ (e.g., a loss value).

We want to find the input $\theta^* \in \mathbb{R}^n$ that minimises L, i.e., $\boxed{\theta^* = \arg\max_{\theta} L(\theta).}$

Gradient Descent works as follows:

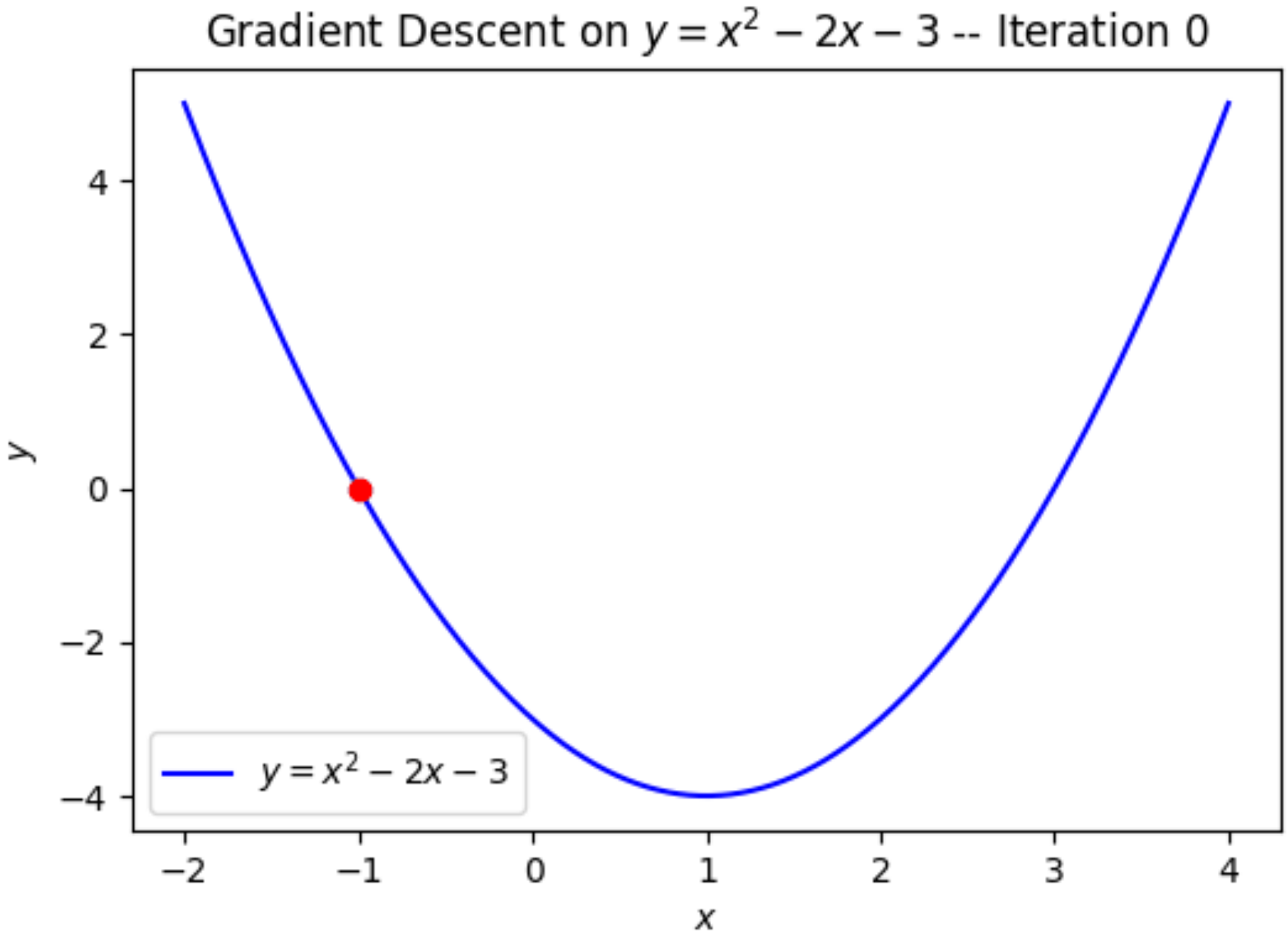
1. Initialise $\theta^{(0)} \in \mathbb{R}^n$ at random

2. Until convergence, for $i = 1, \ldots, n$:

   - $\theta^{(i)} = \theta^{(i-1)} - \underbrace{\boxed{\eta}}_{\text{Learning rate}} \boxed{\nabla_\theta L\left(\theta^{(i-1)}\right)}$ The function $L(\cdot)$ should be differentiable

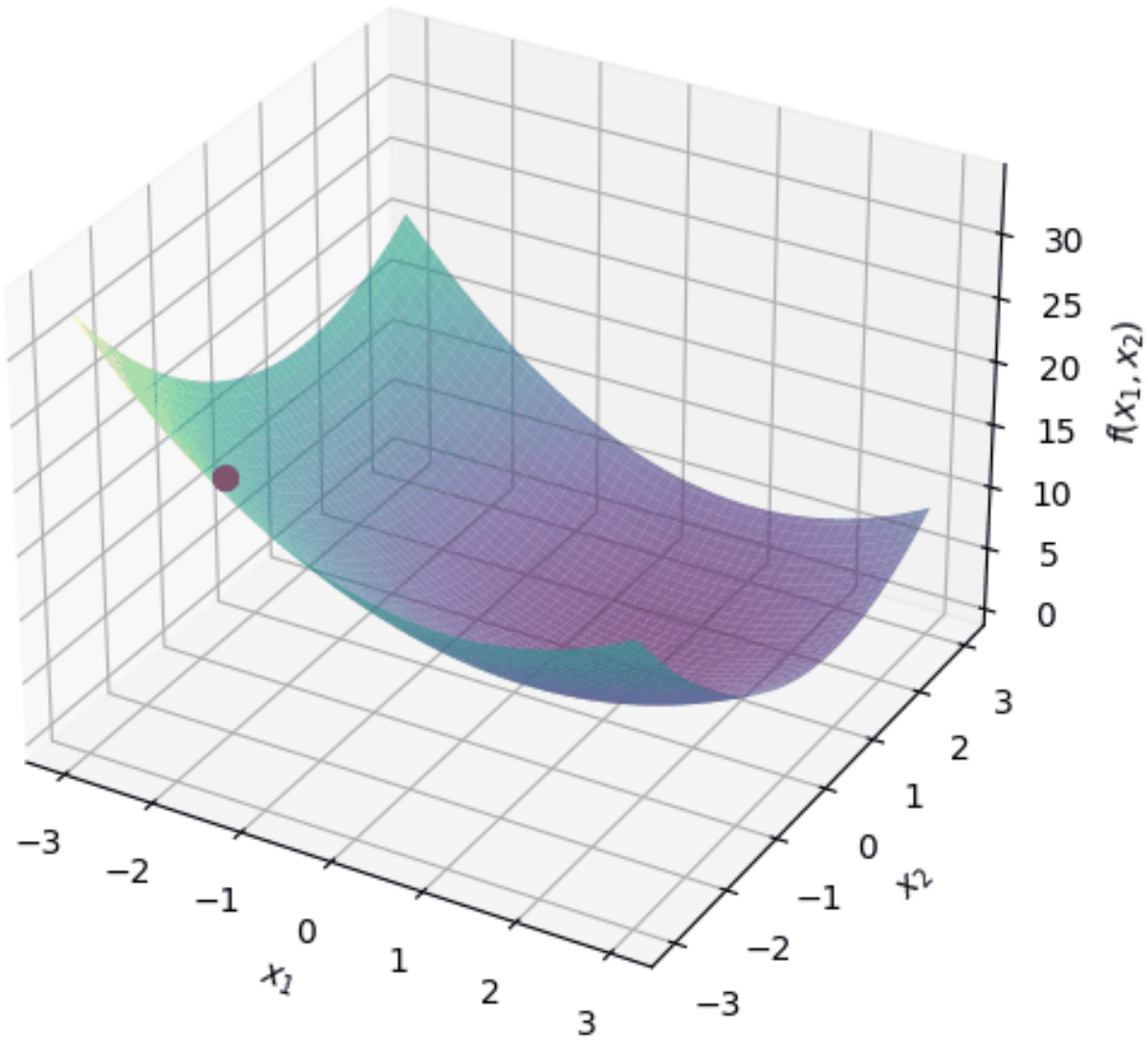
3. Return $\boxed{\theta^{(n)}}$

Gradient Descent



Gradient Descent

Gradient Descent on $f(x_1, x_2) = x_1^2 + x_2^2 - 2x_1 - 2x_2 + 3$ -- Iteration 0



Minimising a Loss Function

Optimiser: Given a KG $G = \{\langle s, p, o \rangle\} \subseteq E \times R \times E$ and a loss function $L(G)$ for an encoder $\text{enc}_\theta(\cdot)$ and scoring function $f(\cdot)$, the optimiser finds θ^* such that:

$$\theta^* = \arg\min_\theta L(G)$$

Minimising a Loss Function

Optimiser: Given a KG $G = \{\langle s, p, o \rangle\} \subseteq E \times R \times E$ and a loss function $L(G)$ for an encoder $\text{enc}_\theta(\cdot)$ and scoring function $f(\cdot)$, the optimiser finds θ^* such that:

$$\theta^* = \arg\min_\theta L(G)$$

Most common solution (by far) is based on **Gradient Descent** — in a nutshell:

1. Initialise $\theta^{(0)}$ (the embeddings) randomly

Follow best practices from Representation/Deep Learning

E.g., based on a validation MRR

2. Until convergence, for $i = 1, \ldots, n$:

Updated parameters

Gradient of $L(G) + \lambda\Omega(\theta)$ wrt. θ

1. $\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_\theta \left[L(G) + \lambda\Omega(\theta) \right]$

Learning rate — hyperparameter

$\Omega(\theta)$ controls the magnitude of θ