Knowledge Graph Embeddings 101

(AKA Neural Link Predictors)

Pasquale Minervini
p.minervini@ed.ac.uk
Main References

Very good introductions to the field:

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](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

(Tour Eiffel, located in, 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*

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

<table>
<thead>
<tr>
<th>s</th>
<th>p</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tour Eiffel</td>
<td>Is located in</td>
<td>Paris</td>
</tr>
<tr>
<td>Paris</td>
<td>Is a Place</td>
<td>Place</td>
</tr>
<tr>
<td>James</td>
<td>Has lived in</td>
<td>Tour Eiffel</td>
</tr>
<tr>
<td>James</td>
<td>Is born on</td>
<td>Jan 1 1984</td>
</tr>
<tr>
<td>James</td>
<td>Has visited</td>
<td>Louvre</td>
</tr>
<tr>
<td>Louvre</td>
<td>Is a Museum</td>
<td>Museum</td>
</tr>
<tr>
<td>Louvre</td>
<td>Is located in</td>
<td>Paris</td>
</tr>
</tbody>
</table>

...
| Knowledge Graph | # Triples ($|G|$) | # Entities |
|-----------------|-----------------|------------|
| yago            | 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 (or 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.

parentOf(Beth, Rick)
parentOf(Morty, Beth)
grandParentOf(X, Y) ← parentOf(X, Z) ∧ parentOf(Z, Y)

↓

grandParentOf(Morty, 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.

parentOf(Beth, Rick)

parentOf(Morty, Beth)

**pos:** grandParentOf(Morty, Rick)

**neg:** grandParentOf(Beth, Rick)

…

\[
\downarrow
\]

grandParentOf(X, Y) ← parentOf(X, Z) ∧ 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

Input → Extract Features → Mapping from Features → Output

Learned from data
Introducing (Graph) Representation Learning

Classic Machine Learning Pipeline

Input → Extract Features → Mapping from Features → Output

Learned from data

Representation Learning Pipeline

Input → Extract Features → Mapping from Features → Output
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

- RESCAL
- TransE
- DistMult
- TransH
- HoIE
- TransR
- TransD
- TransA
- ComplEx
- STransE
- ANALOGY
- TorusE
- ConvKB
- ConvE
- Simple
- CapsE
- TuckER
- CrossE
- RotatE
- ConvR
- RSN

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Symmetry</th>
<th>Antisymmetry</th>
<th>Inversion</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>TransE</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TransX</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>DistMult</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>ComplEx</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>RotatE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

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

- **Louvre** → Subject
- **Located In** → Predicate
- **Paris** → Object

**Encoder** $\theta$:

$E \cup R \rightarrow \mathbb{R}^d$

**Scoring Function** $\in \mathbb{R}^d$

**Loss Function** $L(\theta)$

Measures "how badly" the score function behaves when using the embeddings $f(\cdot)\theta$

**Optimiser** $\arg\min \theta L(\theta)$

Finds the embeddings that minimise the loss $\theta L(\theta)$
Knowledge Graph Embeddings — Components

\[ \text{enc}_\theta : E \cup R \to \mathbb{R}^d \]

Encoder

Subject

predicate

object

Louvre

Located In

Paris

\[ \in \mathbb{R}^d \]

Measures "how badly" the score function behaves when using the embeddings.

\[ \arg \min \theta L(\theta) \]

Finds the embeddings that minimise the loss.

\[ f : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \]
Knowledge Graph Embeddings — Components

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

Encoder

Scoring Function

\[ f : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \]

\[ \mathbb{R} \]

Louvre → Subject

Located In → Predicate

Paris → Object

\( f \) measures "how badly" the score function behaves when using the embeddings \( f(\cdot)\theta \).

\[ \arg \min \theta \mathcal{L}(\theta) \]

Optimiser

Finds the embeddings that minimise the loss \( \theta \mathcal{L}(\theta) \).

\( f \) maps \( \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \) to \( \mathbb{R} \).

\( \mathbb{R} \)
Knowledge Graph Embeddings — Components

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

- **Encoder**
  - \( \theta \)
  - \( E \cup R \rightarrow \mathbb{R}^d \)

- **Scoring Function**
  - \( f : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \)

- **Loss Function**
  - \( L(\theta) \)

**Example:**
- Subject: Louvre
- Predicate: Located In
- Object: Paris


Measures “how badly” the score function \( f( \cdot ) \) behaves when using the embeddings \( \theta \).
Knowledge Graph Embeddings — Components

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

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

Loss Function: $L(\theta)$

Optimiser: $\arg\min_{\theta} L(\theta)$

Subject: Louvre

Predicate: Located In

Object: Paris

Measures “how badly” the score function $f(\cdot)$ behaves when using the embeddings $\theta$.

Finds the embeddings $\theta$ that minimise the loss $L(\theta)$. 
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 $e \in \mathbb{R}^d$ for $e$ (resp. $p \in \mathbb{R}^{d'}$ for $p$):

$$\text{enc}_\theta : E \rightarrow \mathbb{R}^d \quad \text{and} \quad \text{enc}_\theta : R \rightarrow \mathbb{R}^{d'}$$
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 $e \in \mathbb{R}^d$ for $e$ (resp. $p \in \mathbb{R}^{d'}$ for $p$):

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

**Scoring Function:** Given the representations $s \in \mathbb{R}^d$, $p \in \mathbb{R}^{d'}$, $o \in \mathbb{R}^d$ 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:

$$f : \mathbb{R}^d \times \mathbb{R}^{d'} \times \mathbb{R}^d \rightarrow \mathbb{R}$$
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 $e \in \mathbb{R}^d$ for $e$ (resp. $p \in \mathbb{R}^{d'}$ for $p$):

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

Scoring Function: Given the representations $s \in \mathbb{R}^d$, $p \in \mathbb{R}^{d'}$, $o \in \mathbb{R}^d$ 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:

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

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 $\theta^*$ 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 $e \in \mathbb{R}^d$ for $e$ (resp. $p \in \mathbb{R}^{d'}$ 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 $e \in \mathbb{R}^d$ for $e$ (resp. $p \in \mathbb{R}^{d'}$ for $p$):

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

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

<table>
<thead>
<tr>
<th>Entity</th>
<th>Index</th>
<th>Lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>$\text{index}(\text{Paris}) \to 6$</td>
<td>$e_{\text{Paris}} = E_{6,:} \in \mathbb{R}^d$</td>
</tr>
</tbody>
</table>
Scoring Function: Given the representations $s \in \mathbb{R}^d$, $p \in \mathbb{R}^{d'}$, $o \in \mathbb{R}^d$ 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:

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

**Scoring Function:** Given the representations $s \in \mathbb{R}^d$, $p \in \mathbb{R}^{d'}$, $o \in \mathbb{R}^d$ 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:

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

Translation of $s$ with $p$

Distance between $s + p$ and $o$
Scoring Functions

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

$$f(s, W_p, o) = s^T W_p o$$

Scoring Functions

**RESCAL**: subject and object embeddings \( s, o \in \mathbb{R}^d \), predicate embedding \( W_p \in \mathbb{R}^{d \times d} \)

\[
f(s, W_p, o) = s^T W_p o
\]

**DistMult**: subject, predicate, and object embeddings \( s, p, o \in \mathbb{R}^d \)

\[
f(s, p, o) = \langle s, p, o \rangle = (s \odot p)^T o = \sum_{i=1}^{d} s_i p_i o_i
\]

\( W_p \in \mathbb{R}^{d \times d}, o \in \mathbb{R}^d \)

\( s^T \in \mathbb{R}^{1 \times d} \)

\( s^T W_p o \in \mathbb{R} \)

Equivalent to RESCAL with diagonal \( W_p \)

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

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

$$f(s, W_p, o) = s^T W_p o$$

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

$$f(s, p, o) = \langle s, p, o \rangle = (s \odot p)^T o = \sum_{i=1}^d s_i p_i o_i$$

---

# [B] Tensor
# .. and then compute the tri-linear dot product $\langle s, p, o \rangle$, producing the resulting scores.
res = torch.sum(s_emb * p_emb * o_emb, 1)
Scoring Functions

**RESCAL:** subject and object embeddings \( s, o \in \mathbb{R}^d \), predicate embedding \( W_p \in \mathbb{R}^{d \times d} \)

\[
f(s, W_p, o) = s^\top W_p o
\]

**DistMult:** subject, predicate, and object embeddings \( s, p, o \in \mathbb{R}^d \)

\[
f(s, p, o) = \langle s, p, o \rangle = (s \odot p)^\top o = \sum_{i=1}^{d} s_ip_io_i
\]

**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 $s, o \in \mathbb{R}^d$, predicate embedding $W_p \in \mathbb{R}^{d \times d}$

\[ f(s, W_p, o) = s^\top W_p o \]

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

\[ f(s, p, o) = \langle s, p, o \rangle = (s \odot p)^\top o = \sum_{i=1}^{d} s_i p_i o_i \]

*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 $s, p, o \in \mathbb{C}^d$

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

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

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

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

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

```python
# [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:

\[
\begin{aligned}
+ \langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle & - \langle \text{Louvre}, \text{locatedIn}, \text{Edinburgh} \rangle \\
+ \langle \text{Tour Eiffel}, \text{locatedIn}, \text{Paris} \rangle & - \langle \text{Tour Eiffel}, \text{locatedIn}, \text{London} \rangle \\
\vdots
\end{aligned}
\]

\[\text{Accuracy} = \frac{\# \text{ Correct Classifications}}{\# \text{ Test Triples}}\]
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:

\[ \text{Accuracy} = \frac{\# \text{ Correct Classifications}}{\# \text{ Test Triples}} \]

+ \( \langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle \)

- \( \langle \text{Louvre}, \text{locatedIn}, \text{Edinburgh} \rangle \)

+ \( \langle \text{Tour Eiffel}, \text{locatedIn}, \text{Paris} \rangle \)

- \( \langle \text{Tour Eiffel}, \text{locatedIn}, \text{London} \rangle \)

\[ \vdots \]

Requires translating the scores produced by 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

**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:

\[
\begin{align*}
+ \langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle \\
- \langle \text{Louvre}, \text{locatedIn}, \text{Edinburgh} \rangle \\
+ \langle \text{Tour Eiffel}, \text{locatedIn}, \text{Paris} \rangle \\
- \langle \text{Tour Eiffel}, \text{locatedIn}, \text{London} \rangle \\
\ldots
\end{align*}
\]

\[
\text{Accuracy} = \frac{\text{# Correct Classifications}}{\text{# Test Triples}}
\]

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.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Score</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Louvre</td>
<td>locatedIn</td>
<td>London</td>
<td>0.4</td>
<td>2</td>
</tr>
<tr>
<td>Louvre</td>
<td>locatedIn</td>
<td>Edinburgh</td>
<td>0.2</td>
<td>3</td>
</tr>
<tr>
<td>Louvre</td>
<td>locatedIn</td>
<td>Paris</td>
<td>1.4</td>
<td>1</td>
</tr>
<tr>
<td>Louvre</td>
<td>locatedIn</td>
<td>Mars</td>
<td>-0.5</td>
<td>4</td>
</tr>
</tbody>
</table>

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

\[
\text{Hits@}k : \text{Hits@}k = |\{ t \in T \mid \text{Rank } t \leq k \}|
\]
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.

\[
\langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle \rightarrow \begin{array}{cccccc}
\text{Subject} & \text{Predicate} & \text{Object} & \text{Score} & \text{Rank} \\
\text{Louvre} & \text{locatedIn} & \text{London} & 0.4 & 2 \\
\text{Louvre} & \text{locatedIn} & \text{Edinburgh} & 0.2 & 3 \\
\text{Louvre} & \text{locatedIn} & \text{Paris} & 1.4 & 1 \\
\text{Louvre} & \text{locatedIn} & \text{Mars} & -0.5 & 4 \\
\end{array}
\]

\[
\text{Mean Reciprocal Rank: } \text{MRR} = \frac{1}{|T|} \sum_{t \in T} \frac{1}{\text{Rank}_t} \nonumber
\]
**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.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Score</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Louvre</td>
<td>locatedIn</td>
<td>London</td>
<td>0.4</td>
<td>2</td>
</tr>
<tr>
<td>Louvre</td>
<td>locatedIn</td>
<td>Edinburgh</td>
<td>0.2</td>
<td>3</td>
</tr>
<tr>
<td>Louvre</td>
<td>locatedIn</td>
<td>Paris</td>
<td>1.4</td>
<td>1</td>
</tr>
<tr>
<td>Louvre</td>
<td>locatedIn</td>
<td>Mars</td>
<td>-0.5</td>
<td>4</td>
</tr>
</tbody>
</table>

**Mean Reciprocal Rank:**

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

**Hits@k:**

$$\text{Hits}@k = \frac{|\{t \in T | \text{Rank}_t \leq k\}|}{|T|}$$
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 $G$. 

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

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:

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

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 $t$ 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*:

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

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 $t$ denotes the embedding of $t$.

---

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

**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.

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.

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.
**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.

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

Loss Functions

Margin-based Ranking Loss:

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

where \( t \in G \) is a triple from the graph \( G \), and \( t \) denotes the embedding of \( t \).

Logistic Loss:

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

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

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

where \( \text{LogLoss}(t, y) = \log \left( 1 + \exp \left( -y \cdot f(t) \right) \right) \)
K-versus-All (KvsAll) Loss:

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

K-versus-All (KvsAll) Loss:

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

One-versus-All (1vsAll) Loss:

\[ L(G) = \sum_{\langle s, p, o \rangle \in G} -f(s, p, o) + \log \left( \sum_{\hat{o} \in E} \exp \left( f(s, p, \hat{o}) \right) \right) \]
One-versus-All (1vsAll) Loss:

$$L(G) = \sum_{\langle s, p, o \rangle \in G} -f(s, p, o) + \log \left( \sum_{\hat{o} \in E} \exp(f(s, p, \hat{o})) \right)$$
One-versus-All (1vsAll) Loss:

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

\[ P(o \in E | \langle s, p, \cdot \rangle) = \text{softmax} \left( f(s, p, \cdot) \right)_o \]
Loss Functions

One-versus-All (1vsAll) Loss:

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

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

\[
= \frac{\exp(f(s, p, o))}{\sum_{\hat{o} \in E} \exp(f(s, p, \hat{o}))}
\]
One-versus-All (1vsAll) Loss:

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

\[
P(o \in E \mid \langle s, p, \cdot \rangle) = \text{softmax}(f(s, p, \cdot))_o
\]

\[
= \frac{\exp(f(s, p, o))}{\sum_{\hat{o} \in E} \exp(f(s, p, \hat{o}))}
\]

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

Negative Log-Likelihood!

\[
L(G) = \sum_{\langle s, p, o \rangle \in G} \left( -\log P(o \mid \langle s, p, \cdot \rangle) \right)
\]
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 $\theta^*$ 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., $\theta^* = \arg \max_\theta L(\theta)$. 

**Gradient Descent** works as follows: 

1. Initialise at random 
2. Until convergence, for $i = 1, \ldots, n$: 
   - $\theta(i) = \theta(i-1) - \eta \nabla_\theta L(\theta(i-1))$
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., \( \theta^* = \arg \max_\theta L(\theta) \).

**Gradient Descent** works as follows:

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

\[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} L(\theta^{(i-1)}) \]
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., $\theta^* = \arg \max_{\theta} L(\theta)$.

**Gradient Descent** works as follows:

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

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

   
   - $\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} L(\theta^{(i-1)})$
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., $\theta^* = \arg\max_\theta L(\theta)$.

**Gradient Descent** works as follows:

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

2. Until convergence, for $i = 1, \ldots, n$:
   
   \[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_\theta L(\theta^{(i-1)}) \]

   - The function $L(\cdot)$ should be differentiable
   - Learning rate
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., $\theta^* = \text{arg max}_\theta L(\theta)$.

**Gradient Descent** works as follows:

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

2. Until convergence, for $i = 1, \ldots, n$:
   
   $$\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} L (\theta^{(i-1)})$$

3. Return $\theta^{(n)}$
Gradient Descent on $y = x^2 - 2x - 3$ -- Iteration 0
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 $\theta^*$ 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 $\theta^*$ 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

   E.g., based on a validation MRR

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

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

   Learning rate — hyperparameter

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