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 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

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

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

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

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

Encoder

\[ \text{Encoder} \]

\[ \ell(\theta) \]

**Loss Function**

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

**Optimiser**

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

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

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

\( \text{Louvre} \)
Subject

\( \text{Located In} \)
Predicate

\( \text{Paris} \)
Object

\( f : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \)

Scoring Function
Encoder

\[ \arg \min_{\theta} L(\theta) \]

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

\( f \cdot \theta \):
\( \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \)
Knowledge Graph Embeddings — Components

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

Subject: Louvre

Predicate: Located In

Object: Paris

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

Scoring Function

\[ L(\theta) \]

Loss Function

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

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

---

**Encoder**

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

---

**Scoring Function**

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

---

**Loss Function**

- \[ L(\theta) \]

---

**Optimiser**

- \[ \text{arg min}_{\theta} L(\theta) \]

---

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:

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

$$G = \{ \langle s, p, o \rangle \} \subseteq E \times R \times E$$

$$\text{L} \left( \text{G} \right) = \text{L} \left( \text{G} \right)$$

$$\theta^* = \arg \min \theta \text{L} \left( \text{G} \right)$$
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 $p$):

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

**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:

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

**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 $\theta^*$ 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 $p$):

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

Simplest possible encoder: **embedding layer** — it stores all entity representations in a $E \in \mathbb{R}^{|E| \times d}$ 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>index(Paris) $\rightarrow$ 6</td>
<td>$e_{\text{Paris}} = E_{6,:} \in \mathbb{R}^d$</td>
</tr>
</tbody>
</table>
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:

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

Translation of

with $s$

Located in

Distance between $s$ and $o$

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

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

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

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

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

Bordes et al. Translating Embeddings for Modeling Multi-Relational Data. NIPS 2013
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 $s, p, o \in \mathbb{R}^d$

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

**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 $W_p \in \mathbb{R}^{d \times d}$

$$f(s, W_p, o) = s^\top W_p o$$

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

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

---

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

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

Equivalent to RESCAL with diagonal $W_p$

**Problem** — $\langle \text{Louvre}, \text{locatedIn}, \text{Paris} \rangle$ and $\langle \text{Paris}, \text{locatedIn}, \text{Louvre} \rangle$ have the same score!
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 $s, p, o \in \mathbb{C}^d$

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

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.
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.

```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:

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

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

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

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

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

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.

$Louvre, locatedIn, Paris$  

<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>

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

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

<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: $\text{MRR} = \frac{1}{|T|} \sum_{t \in T} \frac{1}{\text{Rank}_t}$
**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>Mars</td>
<td>-0.5</td>
<td>4</td>
</tr>
<tr>
<td>Louvre</td>
<td>locatedIn</td>
<td>Paris</td>
<td>1.4</td>
<td>1</td>
</tr>
</tbody>
</table>

**Mean Reciprocal Rank:**

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

**Hits@k:**

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

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

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

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 $\mathbf{t}$ 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)

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 $\langle \text{Louvre, locatedIn, Paris} \rangle \in G$, we can assume that the triple $\langle \text{Louvre, locatedIn, Edinburgh} \rangle \notin G$ 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.

**Synthetic Negatives:**

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

**Negatives:**

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

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 \( \mathbf{t} \) denotes the embedding of \( t \).

Let’s call this the per-triple loss \( \ell(t) \).
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 \( \text{LogLoss}(t, y) = \log \left( 1 + \exp \left( -y \cdot f(t) \right) \right) \)

Let’s call this the per-triple loss \( \ell(t) \).
Logistic Loss:

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

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

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

One-versus-All (1vsAll) Loss:

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

One-versus-All (1vsAll) Loss:

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

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

One-versus-All (1vsAll) Loss:

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

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

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

One-versus-All (1vsAll) Loss:

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

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

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

Friendly reminder that
\[
\log \frac{a}{b} = \log a - \log b \quad \text{and} \quad \log \left( \exp(a) \right) = a
\]

Negative Conditional Log-Likelihood

\[
L(G) = \sum_{\langle s, p, o \rangle \in G} - \log P\left( o \mid \langle s, p, \cdot \rangle \right)
\]
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 $\theta(0) \in \mathbb{R}^n$
2. Until convergence, for $i = 1, \ldots, n$:
   - Return $\theta(i) \in \mathbb{R}^n$
   - $\theta(i) = \theta(i - 1) - \eta \nabla_{\theta} L(\theta(i - 1))$
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
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 \( i = 1, \ldots, n \):
   
   \[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} L(\theta^{(i-1)}) \]
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.,

$\theta^* = \operatorname{arg\ max}_\theta L(\theta)$.

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., $\theta^* = \arg \max_\theta L(\theta)$.

**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 $\theta^{(n)}$
Imagine you’re blindfolded and on a hilly terrain, and you want to find the lowest point (valley) on this terrain. You can feel the slope of the ground beneath your feet.

To find the lowest point:

1. You take a small step in the direction where the slope descends the most steeply.
2. You repeat this, adjusting the direction at each step based on the slope you feel underfoot.
3. Gradually, you'll move closer to the valley.
Imagine you’re blindfolded and on a hilly terrain, and you want to find the lowest point (valley) on this terrain. You can feel the slope of the ground beneath your feet.

To find the lowest point:

1. You take a small step in the direction where the slope descends the most steeply.
Gradient Descent - Intuition

Imagine you’re blindfolded and on a hilly terrain, and you want to find the lowest point (valley) on this terrain. You can feel the slope of the ground beneath your feet.

To find the lowest point:

1. You take a small step in the direction where the slope descends the most steeply.

2. You repeat this, adjusting the direction at each step based on the slope you feel underfoot.
Imagine you’re blindfolded and on a hilly terrain, and you want to find the lowest point (valley) on this terrain. You can feel the slope of the ground beneath your feet.

To find the lowest point:

1. You take a small step in the direction where the slope descends the most steeply.

2. You repeat this, adjusting the direction at each step based on the slope you feel underfoot.

3. Gradually, you’ll move closer to the valley.
Gradient Descent

Gradient Descent on $y = x^2 - 2x - 3$ -- Iteration 0

$y = x^2 - 2x - 3$
Gradient Descent

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

1. Initialise $\theta^{(0)} \in \mathbb{R}^n$ at random
2. Until convergence, for $i = 1, \ldots, n$:
   $\cdot \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} L(\theta^{(i-1)})$
3. Return $\theta^{(n)}$

Learning rate
Automatic Learning Rate Selection

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 $\theta^{(n)}$

Fixed Learning Rate:

\[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} L(\theta^{(i-1)}) \]
Automatic Learning Rate Selection

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

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

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

Fixed Learning Rate:

$$\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} L \left( \theta^{(i-1)} \right)$$

Momentum:

$$v = \beta v + (1 - \beta) \nabla_{\theta} L \left( \theta^{(i-1)} \right), \quad \theta^{(i)} = \theta^{(i-1)} - \eta v$$
Automatic Learning Rate Selection

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 $\theta^{(n)}$

---

Fixed Learning Rate:

\[
\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_\theta L(\theta^{(i-1)})
\]

Momentum:

\[
v = \beta v + (1 - \beta) \nabla_\theta L(\theta^{(i-1)}), \quad \theta^{(i)} = \theta^{(i-1)} - \eta v
\]

Adagrad:

\[
\theta^{(i+1)} = \theta^{(i)} - \frac{\eta}{\sqrt{\sum_{j=0}^{i} g_j^2}} \nabla_\theta L(\theta^{(i)})
\]
Automatic Learning Rate Selection

Fixed Learning Rate:
\[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_\theta L(\theta^{(i-1)}) \]

Momentum:
\[ v = \beta v + (1 - \beta) \nabla_\theta L(\theta^{(i-1)}) \]
\[ \theta^{(i)} = \theta^{(i-1)} - \eta v \]

Adagrad:
\[ \theta^{(i+1)} = \theta^{(i)} - \eta \sum_{j=0}^{i} g_j^2 \]

Adam:
\[ m_i = \beta_1 m_{i-1} + (1 - \beta_1) \nabla_\theta L(\theta^{(i)}) \]
\[ v_i = \beta_2 v_{i-1} + (1 - \beta_2) \nabla_\theta L(\theta^{(i)})^2 \]
\[ \theta^{(i+1)} = \theta^{(i)} - \eta \frac{\hat{m}_i}{\sqrt{\hat{v}_i} + \epsilon} \]

- Running average of the gradient
- Accumulator of previous gradients
- Estimate of first moment of gradient
- Estimate of second moment of gradient
Automatic Learning Rate Selection
Automatic Learning Rate Selection
Automatic Learning Rate Selection
Gradient Descent — Variants

\[ \theta^* = \arg \min_{\theta} L(\theta), \text{ with } L(\theta) = \ell_{\theta}(d_1) + \ldots + \ell_{\theta}(d_n) = \sum_{d \in D} \ell_{\theta}(d) \]
Gradient Descent — Variants

\[ \theta^* = \arg \min_{\theta} L(\theta), \text{ with } L(\theta) = \ell_\theta(d_1) + \ldots + \ell_\theta(d_n) = \sum_{d \in D} \ell_\theta(d) \]

(Batch) Gradient Descent: use all samples \( d_1, \ldots, d_n \) in the dataset \( D \) at each iteration:

\[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_\theta L(\theta^{(i-1)}) \]  

We consider the full dataset \( D \)
Gradient Descent — Variants

\[ \theta^* = \arg \min_{\theta} L(\theta), \text{ with } L(\theta) = \ell_{\theta}(d_1) + \ldots + \ell_{\theta}(d_n) = \sum_{d \in D} \ell_{\theta}(d) \]

(Batch) Gradient Descent: use all samples \( d_1, \ldots, d_n \) in the dataset \( D \) at each iteration:

\[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} L(\theta^{(i-1)}) \]  
\( \text{We consider the full dataset } D \)

Stochastic Gradient Descent (SGD): use one sample \( d_i \) at each iteration:

\[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} \ell_{\theta}(d), \text{ for each } d \in D \]  
\( \text{We iterate over all samples } \ d \in D, \text{ one at a time} \)
Gradient Descent — Variants

\[ \theta^* = \arg \min_{\theta} L(\theta), \text{ with } L(\theta) = \ell_{\theta}(d_1) + \ldots + \ell_{\theta}(d_n) = \sum_{d \in D} \ell_{\theta}(d) \]

(Batch) Gradient Descent: use all samples \( d_1, \ldots, d_n \) in the dataset \( D \) at each iteration:

\[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} L \left( \theta^{(i-1)} \right) \]
We consider the full dataset \( D \)

Stochastic Gradient Descent (SGD): use one sample \( d_i \) at each iteration:

\[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} \ell_{\theta} \left( d \right), \text{ for each } d \in D \]
We iterate over all samples \( d \in D \), one at a time

Mini-Batch Gradient Descent: use a batch of samples \( B \subseteq D \) at each iteration:

\[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} \hat{L} \left( \theta^{(i-1)} \right), \text{ with } \hat{L}(\theta) = \sum_{d \in B} \ell_{\theta}(d) \]
We partition \( D \) in subsets \( B_1, \ldots, B_m \subseteq 2^D \) and iterate over the subsets (called batches)
Training KG Embedding Models

**Problem:** 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 approaches use **Mini-Batch Gradient Descent**:

- **Regulariser**: \( \Omega(\theta) \)
- **Learning rate (Adagrad/Adam)**

E.g., based on the validation MRR

1. Initialise the embeddings randomly
2. Until convergence, for:
   - Sample a batch of triples \( B = \{ t_1, \ldots, t_m \} \subseteq G \)
   - Define \( \hat{L}(B) = \sum_{t \in B} \ell_{\theta}(t) \)
   - \( \theta(i) = \theta(i-1) - \eta \nabla_{\theta}[\hat{L}(B) + \lambda \Omega(\theta)] \)
Most approaches use **Mini-Batch Gradient Descent**:

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

Check best practices from the Deep Learning textbook
Training KG Embedding Models

Most approaches use **Mini-Batch Gradient Descent**:

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

2. Until convergence, for $i = 1, \ldots, n$:
   
   - Sample a batch of triples
   - Define $\hat{L}(B) = \ell_{\theta}(t_1) + \cdots + \ell_{\theta}(t_m) = \sum_{t \in B} \ell_{\theta}(t)$
   - Updated parameters $\theta(i) = \theta(i-1) - \eta \nabla_{\theta} \left[ \hat{L}(B) + \lambda \Omega(\theta) \right]$
Training KG Embedding Models

Most approaches use **Mini-Batch Gradient Descent**:

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

2. Until convergence, for $i = 1, \ldots, n$:
   - Sample a batch of triples $B = \{t_1, \ldots, t_m\} \subseteq G$

   E.g., based on the validation MRR

Check best practices from the Deep Learning textbook
Most approaches use **Mini-Batch Gradient Descent**:

1. Initialise the embeddings $\theta^{(0)}$ \textcolor{blue}{randomly}. \textcolor{red}{Check best practices from the Deep Learning textbook}

2. Until convergence, for $i = 1, \ldots, n$: \textcolor{blue}{E.g., based on the validation MRR}

   - Sample a batch of triples $B = \{t_1, \ldots, t_m\} \subseteq G$

   Define $\hat{L}(B) = \ell_{\theta}(t_1) + \ldots + \ell_{\theta}(t_m) = \sum_{t \in B} \ell_{\theta}(t)$ over
Training KG Embedding Models

Most approaches use **Mini-Batch Gradient Descent**:

1. Initialise the embeddings $\theta^{(0)}$ randomly. Check best practices from the Deep Learning textbook

2. Until convergence, for $i = 1, \ldots, n$: E.g., based on the validation MRR
   - Sample a batch of triples $B = \{t_1, \ldots, t_m\} \subseteq G$
   - Define $\hat{L}(B) = \ell_\theta(t_1) + \ldots + \ell_\theta(t_m) = \sum_{t \in B} \ell_\theta(t)$ over
   - $\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_{\theta} \left[ \hat{L}(B) + \lambda \Omega(\theta) \right]$

Updated parameters

Learning rate (Adagrad/Adam)

The regulariser $\Omega(\theta)$ controls the magnitude of $\theta$
Why Using a Regulariser $\Omega(\theta)$?

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

The training process can minimise the loss by artificially increasing the norm of the embeddings [Bordes et al. 2015]. How does the regulariser $\Omega(\theta)$ look like?
Why Using a Regulariser $\Omega(\theta)$?

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

The training process can minimise the loss by artificially increasing the norm of the embeddings [Bordes et al. 2015]. How does the regulariser $\Omega(\theta)$ look like?

$L_2$ Norm: $\Omega(\theta) = \sum_i \theta_i^2$
Why Using a Regulariser $\Omega(\theta)$?

The training process can minimise the loss by artificially increasing the norm of the embeddings [Bordes et al. 2015]. How does the regulariser $\Omega(\theta)$ look like?

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

$L_2$ Norm: $\Omega(\theta) = \sum_i \theta_i^2$

$L_1$ Norm: $\Omega(\theta) = \sum_i |\theta_i|$
Why Using a Regulariser $\Omega(\theta)$?

The training process can minimise the loss by artificially increasing the norm of the embeddings [Bordes et al. 2015]. How does the regulariser $\Omega(\theta)$ look like?

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

$L_2$ Norm: $\Omega(\theta) = \sum_i \theta_i^2$

$L_1$ Norm: $\Omega(\theta) = \sum_i |\theta_i|$

Generalised $L_p$-norms: $\Omega_p^\alpha(\theta) = \sum_i \|\theta_i^2\|^\alpha$
Why Using a Regulariser $\Omega(\theta)$?

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

The training process can minimise the loss by artificially increasing the norm of the embeddings [Bordes et al. 2015]. How does the regulariser $\Omega(\theta)$ look like?

**$L_2$ Norm:** $\Omega(\theta) = \sum_i \theta_i^2$

**$L_1$ Norm:** $\Omega(\theta) = \sum_i |\theta_i|$

**Generalised $L_p$-norms:** $\Omega_p^\alpha(\theta) = \sum_i \|\theta_i^2\|_p^\alpha$

Auxiliary training objectives; Norm clipping; [..]