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

- RESCAL (2011)
- TransE (2013)
- DistMult (2014)
- TransH (2014)
- HoLE (2015)
- TransR (2015)
- ComplEx (2016)
- STransE (2016)
- ANALOGY (2017)
- TorusE (2017)
- ConvKB (2017)
- ConvE (2018)
- SimpleCapsE (2018)
- Tucker (2019)
- CrossE (2019)
- RotatE (2019)
- ConvR (2019)
- RSN (2019)

Tensor decomposition

Geometric

Deep Learning

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>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TransE</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TransX</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DistMult</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>ComplEx</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</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

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

**Encoder** $\theta: E \cup R \rightarrow \mathbb{R}^d$

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

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

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

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

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

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

- **Louvre**
  - Subject
  - \( \in \mathbb{R}^d \)
- **Located In**
  - Predicate
- **Paris**
  - Object

**Encoder**

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

**Optimiser**

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

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

Loss Function: \( L(\theta) \)

Optimiser: arg\( \min_\theta L(\theta) \)

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

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

- **Encoder**
  - \[ \text{Enc} \]
  - \[ \mathbb{E} \cup \mathbb{R} \rightarrow \mathbb{R}^d \]

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

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

Subject → Louvre

 Predicate → Located In

 Object → Paris

**loss function**

\[ L(\theta) \]

**measures** “how badly” the score function \( f(\cdot) \) behaves when using the embeddings \( \theta \)

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

**finds the embeddings** that minimise the loss \( \theta L(\theta) \)
Knowledge Graph Embeddings — Components

encoder: \( E \cup R \rightarrow \mathbb{R}^d \)

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

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

Optimiser: \( \text{arg min}_\theta L(\theta) \)

Loss Function: \( L(\theta) \)

Finds the embeddings \( \theta \) that minimise the loss \( L(\theta) \)
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:

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

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:

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

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)

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.

**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:** $N(\langle s, p, o \rangle) \subseteq \{ \langle \hat{s}, p, o \rangle | \hat{s} \in E, \langle \hat{s}, p, o \rangle \notin G \}$

**Negatives:** $\cup \{ \langle s, p, \hat{o} \rangle | \hat{o} \in E, \langle s, p, \hat{o} \rangle \notin G \}$

Loss Functions

Margin-based Ranking Loss:

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

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

Let's call this the per-triple loss \( \ell(t) \).
Loss Functions

Margin-based Ranking Loss:

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

where \( t \in G \) is a triple from the graph \( G \), and \( f(t) \) 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 \( \text{LogLoss}(t, y) = \log\left(1 + \exp\left(-y \cdot f(t)\right)\right) \)

Loss for positive triples

Loss for negative triples

Score × Loss plot
Loss Functions

K-versus-All (KvsAll) Loss:

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

Dettmers et al. Convolutional 2D Knowledge Graph Embeddings. AAAI 2018
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:

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

Lacroix et al. Canonical Tensor Decomposition for Knowledge Graph Completion. ICML 2018
Loss Functions

One-versus-All (1vsAll) Loss:

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

One-versus-All (1vsAll) Loss:

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

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

One-versus-All (1vsAll) Loss:

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

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

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

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

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

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

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

Negative Conditional Log-Likelihood

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

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

3. Return $\theta^{(n)}$
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.
3. Gradually, you’ll move closer to the valley.
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.
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.
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.

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

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
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 $\theta^{(n)}$
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 $i = 1, \ldots, n$:
   - $\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_\theta L(\theta^{(i-1)})$

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

Fixed Learning Rate:

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

Momentum:

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

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

**Fixed Learning Rate:**

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

**Momentum:**

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

**Adagrad:**

\[ \theta^{(i+1)} = \theta^{(i)} - \frac{\eta}{\sqrt{\sum_{j=1}^{i} g_j^2}} \odot \nabla_\theta L(\theta^{(i)}) \]

- **Magnitude of previous gradients**

**Adam:**

\[ m_i = \beta_1 m_{i-1} + (1 - \beta_1) \nabla_\theta L(\theta^{(i)}) \]
\[ v_i = \beta_2 v_{i-1} + (1 - \beta_2) \nabla_\theta L(\theta^{(i)})^2 \]

\[ \theta^{(i+1)} = \theta^{(i)} - \eta \frac{\hat{m}_i}{\sqrt{\hat{v}_i} + \epsilon} \]

- **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:

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

We consider the full dataset \( D \)

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

\[ \theta^{(i)} = \theta^{(i-1)} - \eta \nabla_\theta \ell_\theta(d), \text{ for each } d \in D \]

We iterate over all samples \( d \in D \), 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 \(B_1, \ldots, B_m \in 2^D\) 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**:

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

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

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

1. Initialise the embeddings $\theta^{(0)}$ randomly
2. Until convergence, for $i = 1, \ldots, n$: E.g., based on the validation MRR
   - Sample a batch of triples
   - Define $\hat{L}(B) = \ell_{\theta}(t_1) + \cdots + \ell_{\theta}(t_m) = \sum_{t \in B} \ell_{\theta}(t)$
Most approaches use **Mini-Batch Gradient Descent**: 

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

2. Until convergence, for \(i = 1, \ldots, n\): \(\text{E.g., based on the validation MRR}\)

   - Sample a batch of triples \(B = \{t_1, \ldots, t_m\} \subseteq G\)
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 $B = \{t_1, \ldots, t_m\} \subseteq G$
   - Define $\hat{L}(B) = \ell_{\theta}(t_1) + \ldots + \ell_{\theta}(t_m) = \sum_{t \in B} \ell_{\theta}(t)$ over

Check best practices from the Deep Learning textbook

E.g., based on the validation MRR

Updated parameters
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 $B = \{t_1, \ldots, t_m\} \subseteq G$

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

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

   - The regulariser $\Omega(\theta)$ controls the magnitude of $\theta$

   - Check best practices from the Deep Learning textbook

   - E.g., based on the validation MRR

---

Updated parameters

Learning rate (Adagrad/Adam)
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 $\Omega(\theta)$?

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

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:** $\Omega(\theta) = \sum_i |\theta_i|$
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: $\Omega_p^\alpha(\theta) = \sum_i \|\theta_i^2\|^\alpha_p$
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: $\Omega^\alpha_p(\theta) = \sum_i \|\theta_i^2\|^\alpha_p$

Auxiliary training objectives; Norm clipping; [...]
## Results

<table>
<thead>
<tr>
<th></th>
<th>FB15K-237</th>
<th></th>
<th>WNRR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MRR</td>
<td>Hits@10</td>
<td>MRR</td>
<td>Hits@10</td>
</tr>
<tr>
<td><strong>First</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESCAL (Wang et al., 2019)</td>
<td>27.0</td>
<td>42.7</td>
<td>42.0</td>
<td>44.7</td>
</tr>
<tr>
<td>TransE (Nguyen et al., 2018)</td>
<td>29.4</td>
<td>46.5</td>
<td>22.6</td>
<td>50.1</td>
</tr>
<tr>
<td>DistMult (Dettmers et al., 2018)</td>
<td>24.1</td>
<td>41.9</td>
<td>43.0</td>
<td>49.0</td>
</tr>
<tr>
<td>ComplEx (Dettmers et al., 2018)</td>
<td>24.7</td>
<td>42.8</td>
<td>44.0</td>
<td>51.0</td>
</tr>
<tr>
<td>ConvE (Dettmers et al., 2018)</td>
<td>32.5</td>
<td>50.1</td>
<td>43.0</td>
<td>52.0</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESCAL</td>
<td>35.7</td>
<td>54.1</td>
<td>46.7</td>
<td>51.7</td>
</tr>
<tr>
<td>TransE</td>
<td>31.3</td>
<td>49.7</td>
<td>22.8</td>
<td>52.0</td>
</tr>
<tr>
<td>DistMult</td>
<td>34.3</td>
<td>53.1</td>
<td>45.2</td>
<td>53.1</td>
</tr>
<tr>
<td>ComplEx</td>
<td>34.8</td>
<td>53.6</td>
<td>47.5</td>
<td>54.7</td>
</tr>
<tr>
<td>ConvE</td>
<td>33.9</td>
<td>52.1</td>
<td>44.2</td>
<td>50.4</td>
</tr>
<tr>
<td><strong>Recent</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TuckER (Balazevic et al., 2019)</td>
<td>35.8</td>
<td>54.4</td>
<td>47.0</td>
<td>52.6</td>
</tr>
<tr>
<td>RotatE (Sun et al., 2019a)</td>
<td>33.8</td>
<td>53.3</td>
<td>47.6</td>
<td>57.1</td>
</tr>
<tr>
<td>SACN (Shang et al., 2019)</td>
<td>35.0</td>
<td>54.0</td>
<td>47.0</td>
<td>54.4</td>
</tr>
<tr>
<td><strong>Large</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DistMult (Salehi et al., 2018)</td>
<td>35.7</td>
<td>54.8</td>
<td>45.5</td>
<td>54.4</td>
</tr>
<tr>
<td>ComplEx-N3 (Lacroix et al., 2018)</td>
<td>37.0</td>
<td>56.0</td>
<td>49.0</td>
<td>58.0</td>
</tr>
</tbody>
</table>
Code!