Knowledge Graph Embeddings 101

(AKA Neural Link Predictors)

Pasquale Minervini
p.minervini@ed.ac.uk

Main References

Very good introductions to the field:

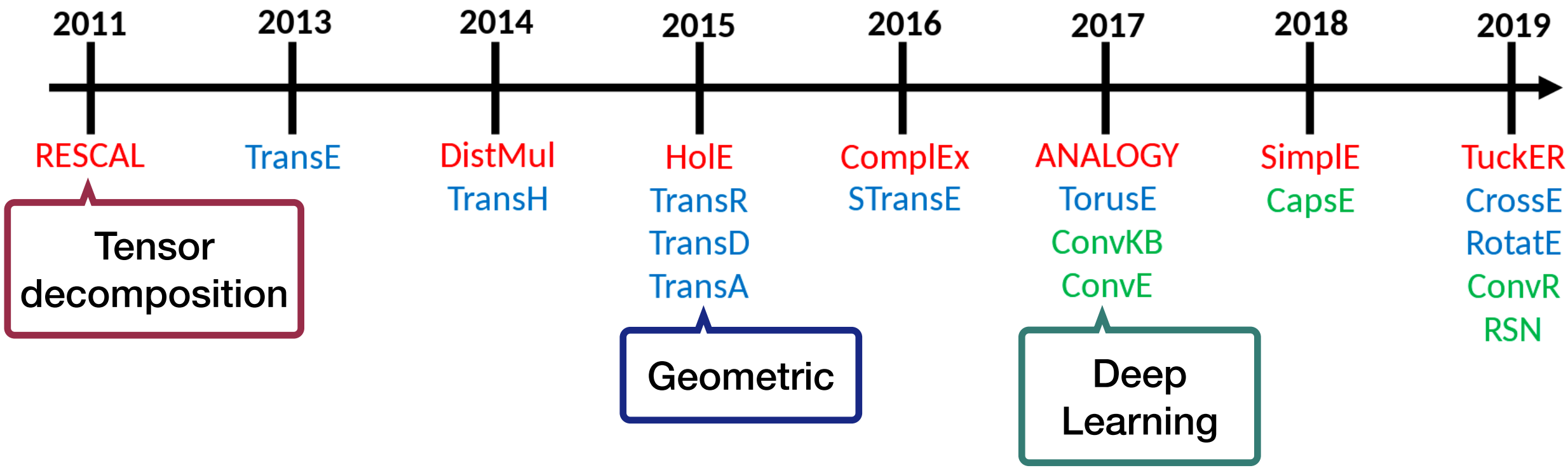
1. Nickel et al., A Review of Relational Machine Learning for Knowledge Graphs. https://arxiv.org/abs/1503.00759 ⭐
2. Hamilton, Graph Representation Learning Book. https://www.cs.mcgill.ca/~wlh/grl_book/ 📚
3. Goodfellow, Deep Learning. https://www.deeplearningbook.org/ 📚

Very simple models can work surprisingly well:

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

Knowledge Graph Embeddings

Some KGE models in the recent published literature



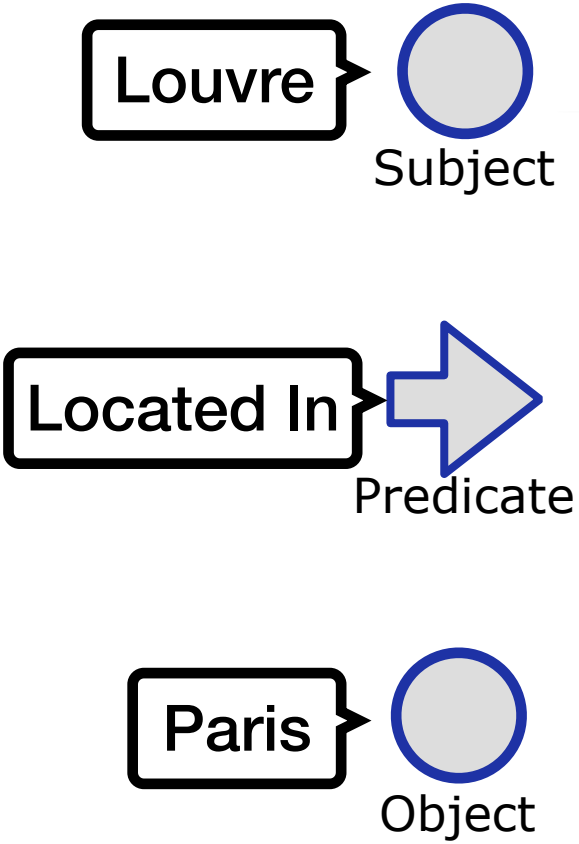
Must be able to capture graph properties — relation properties (e.g., (a)symmetry), hierarchies, type constraints, transitivity, homophily, long range dependencies, etc.

Knowledge Graph Embeddings

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

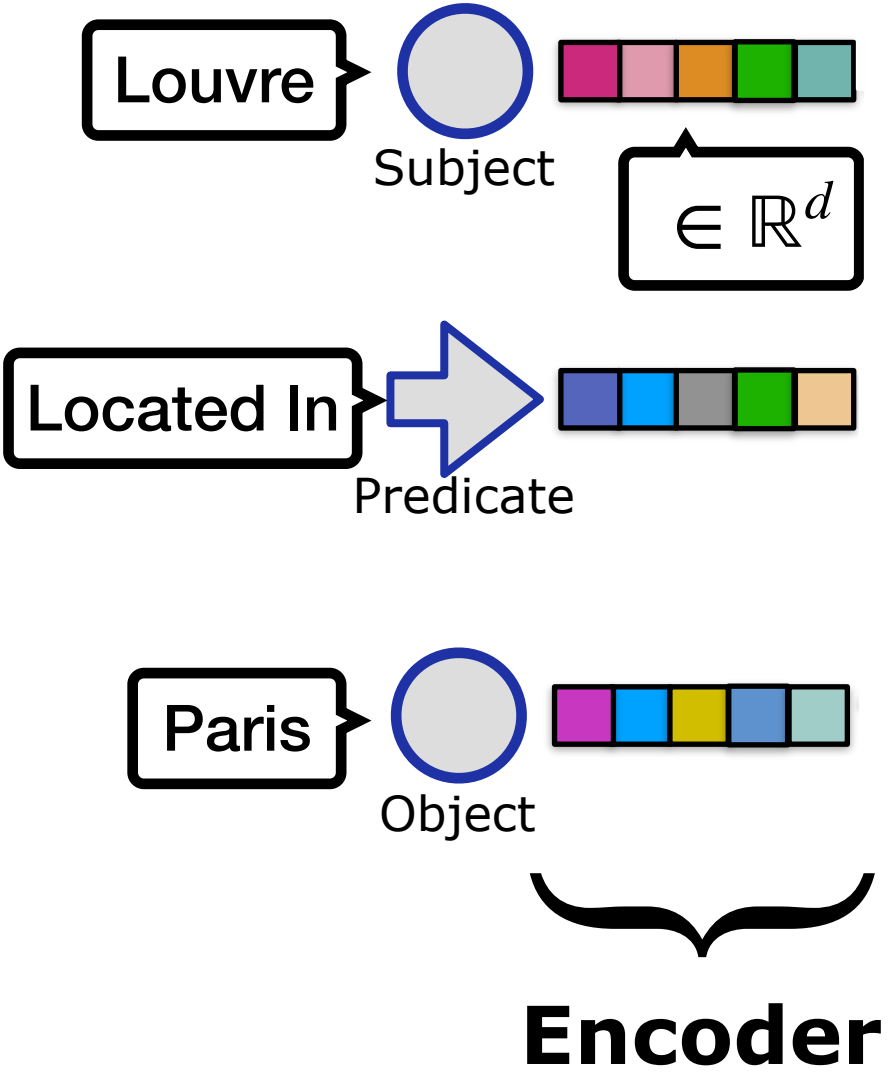
From Sun et al. - RotatE: Knowledge Graph Embeddings by Relational Rotation in Complex Space

Knowledge Graph Embeddings — Components

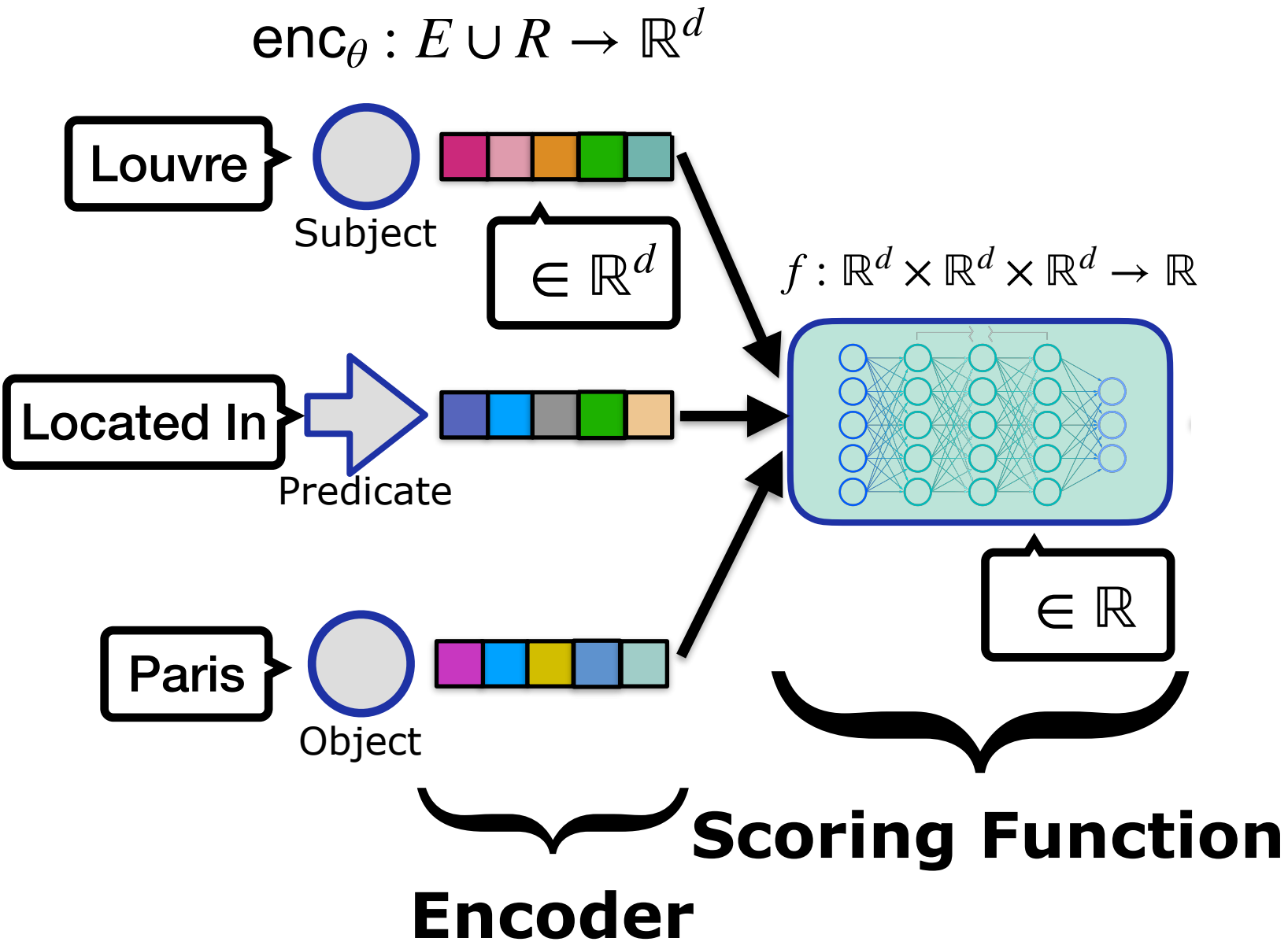


Knowledge Graph Embeddings — Components

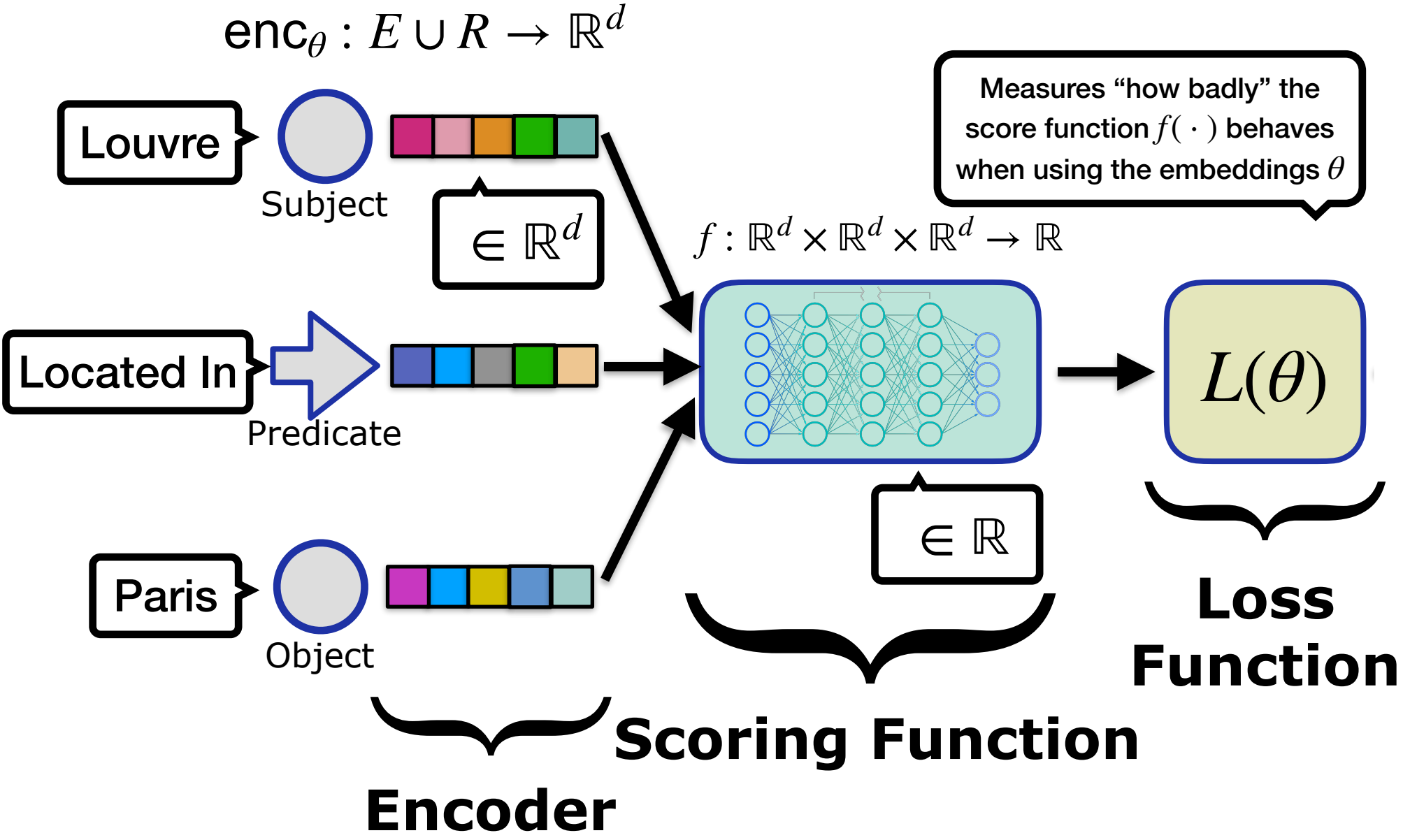
$\text{enc}_\theta : E \cup R \rightarrow \mathbb{R}^d$



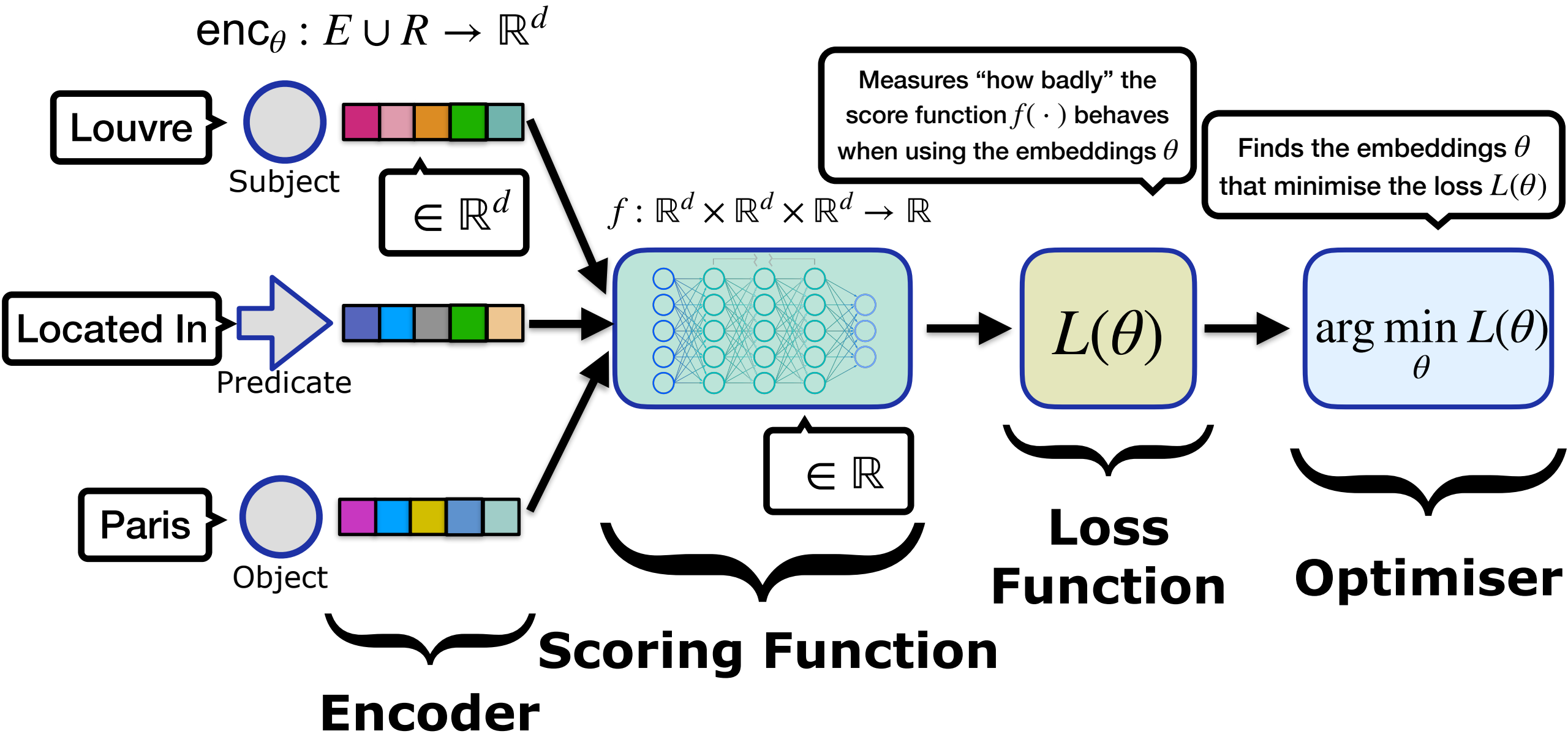
Knowledge Graph Embeddings — Components



Knowledge Graph Embeddings — Components



Knowledge Graph Embeddings — Components



Loss Functions

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

Loss Functions

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

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

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

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

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

Loss Functions

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

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

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

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

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

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

Negative Generation

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

Negative Generation

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

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

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

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

Negative Generation

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

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

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

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

Negative Generation

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

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

Synthetic Negatives:

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

Sampled

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

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

Loss Functions

Margin-based Ranking Loss:

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

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

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

Loss Functions

Margin-based Ranking Loss:

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

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

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

Logistic Loss:

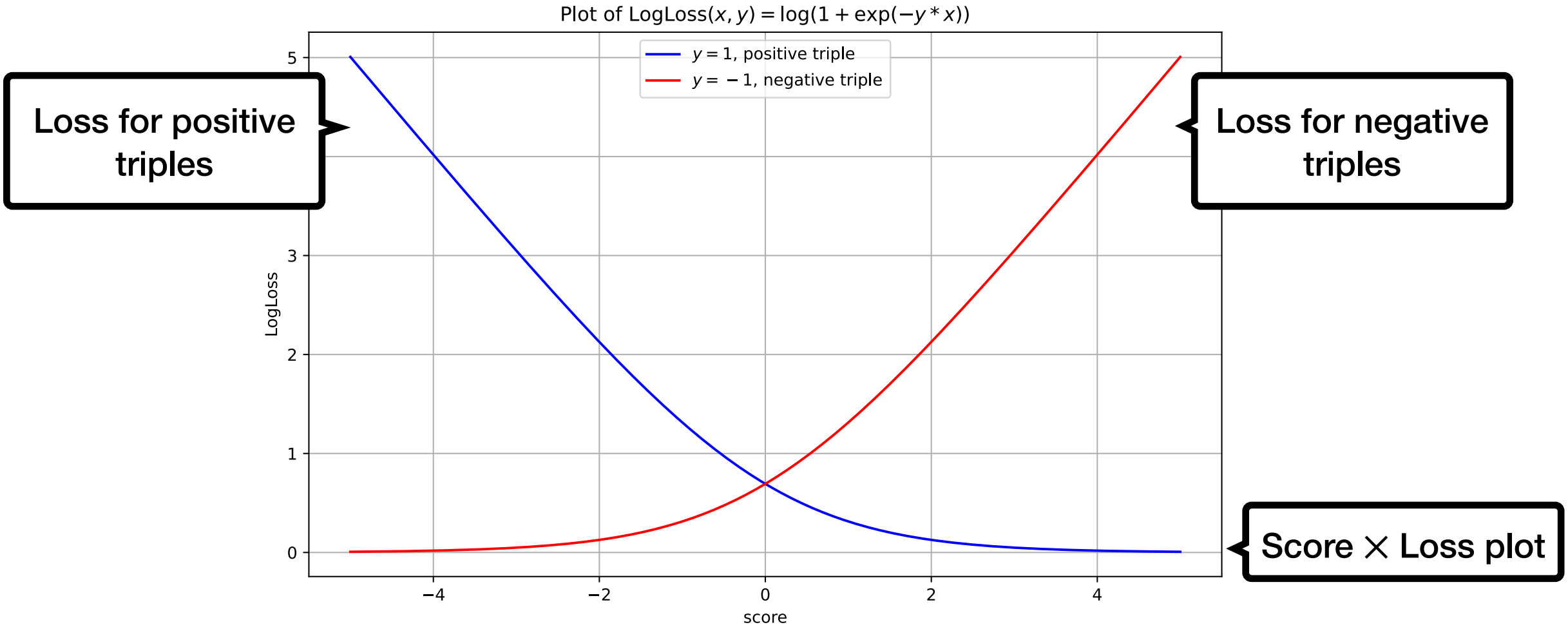
$$L(G) = \sum_{t \in G} \boxed{\text{LogLoss}(t, 1) + \sum_{t' \in N(t)} \text{LogLoss}(t', -1),}$$

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

Logistic Loss:

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

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



Loss Functions

K-versus-All (KvsAll) Loss:

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

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

Loss Functions

K-versus-All (KvsAll) Loss:

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

One-versus-All (1vsAll) Loss:

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

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

Loss Functions

Why tho!? 🤔

One-versus-All (1vsAll) Loss:

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

Loss Functions

Why tho!? 🤔

One-versus-All (1vsAll) Loss:

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

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

Loss Functions

Why tho!? 🤔

One-versus-All (1vsAll) Loss:

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

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

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

Loss Functions

Why tho!? 🤔

One-versus-All (1vsAll) Loss:

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

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

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

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

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

Negative Conditional Log-Likelihood

Minimising a Loss Function

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

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

Gradient Descent in a Nutshell

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

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

Gradient Descent in a Nutshell

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

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

Gradient Descent works as follows:

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

Gradient Descent in a Nutshell

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

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

Gradient Descent works as follows:

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

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

Gradient Descent in a Nutshell

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

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

Gradient Descent works as follows:

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

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

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

Gradient Descent in a Nutshell

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

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

Gradient Descent works as follows:

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

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

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

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

Gradient Descent - 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:

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.

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.

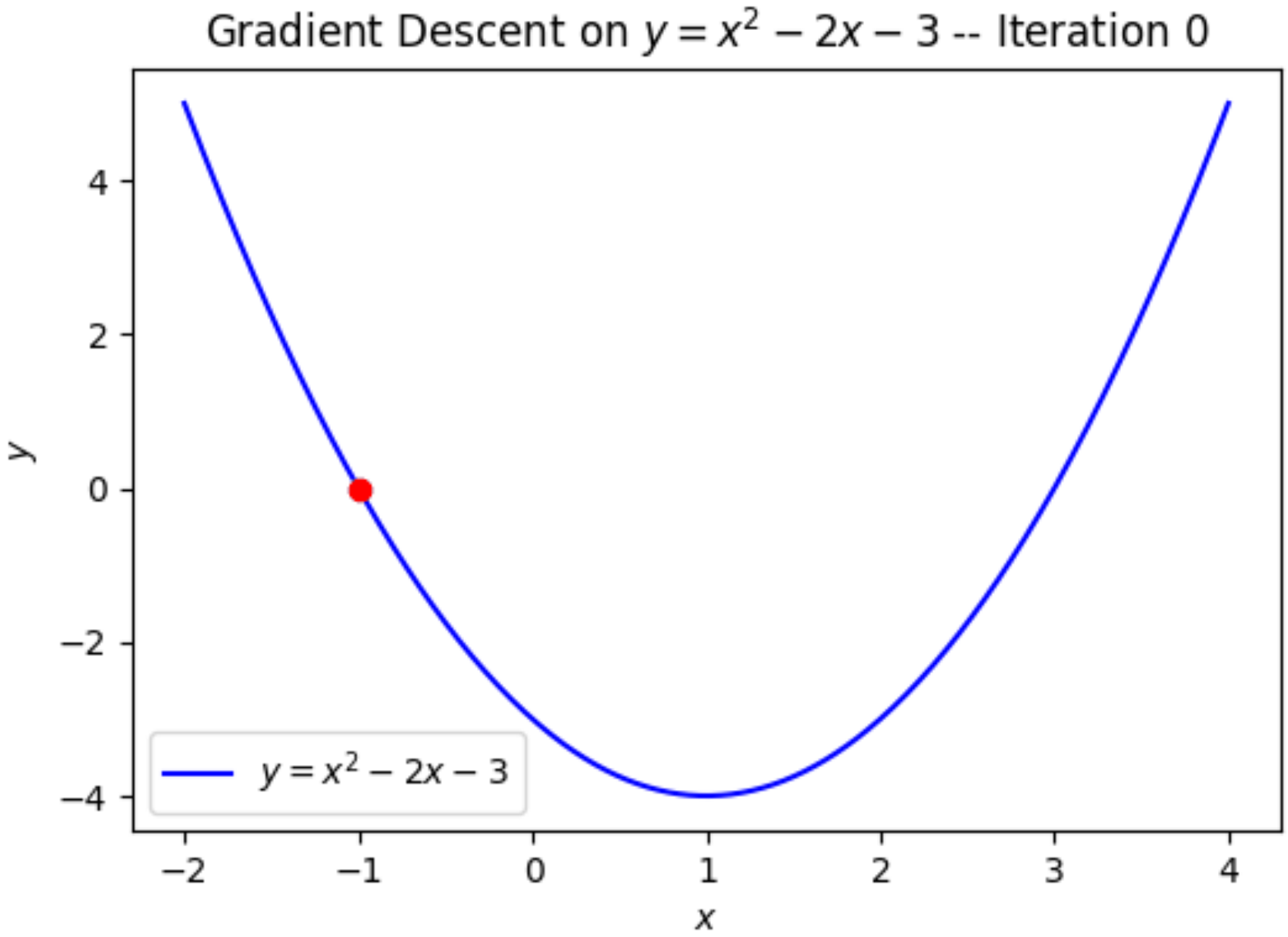
Gradient Descent - Intuition

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:

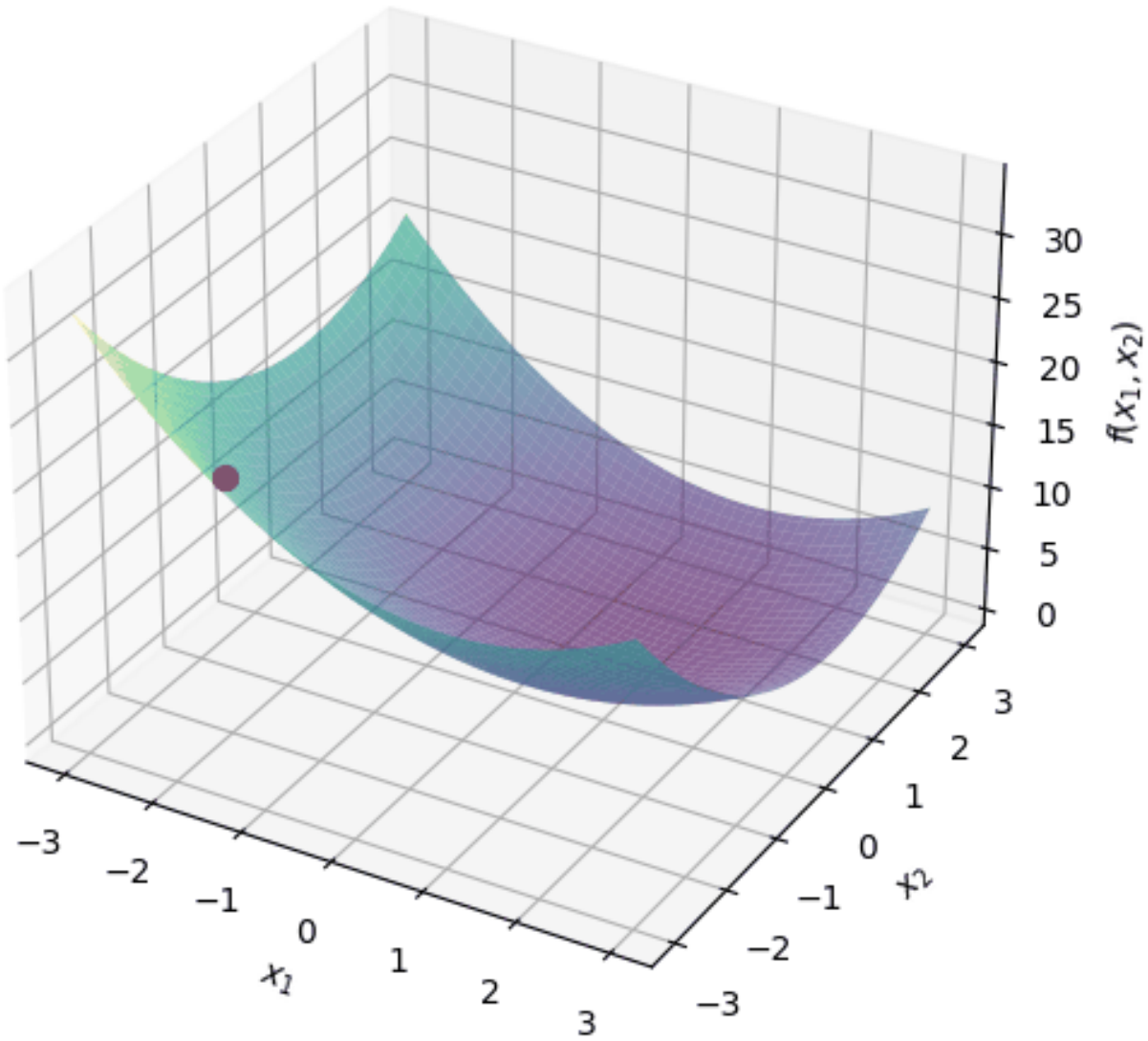
1. You take a small step in the direction where the slope descends the most steeply.
2. You repeat this, adjusting the direction at each step based on the slope you feel underfoot.
3. Gradually, you'll move closer to the valley.

Gradient Descent



Gradient Descent

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 $i = 1, \ldots, n$:
   - $\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_\theta L\left(\theta^{(i-1)}\right)$
3. Return $\theta^{(n)}$

Learning rate

Automatic Learning Rate Selection

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

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

Fixed Learning Rate:

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

Automatic Learning Rate Selection

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

Learning rate

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

Fixed Learning Rate:

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

Momentum:

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

Automatic Learning Rate Selection

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

Learning rate

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

Fixed Learning Rate:

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

Momentum:

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

Adagrad:

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

Magnitude of previous gradients

Automatic Learning Rate Selection

Fixed Learning Rate:

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

Momentum:

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

Adagrad:

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

Magnitude of previous gradients

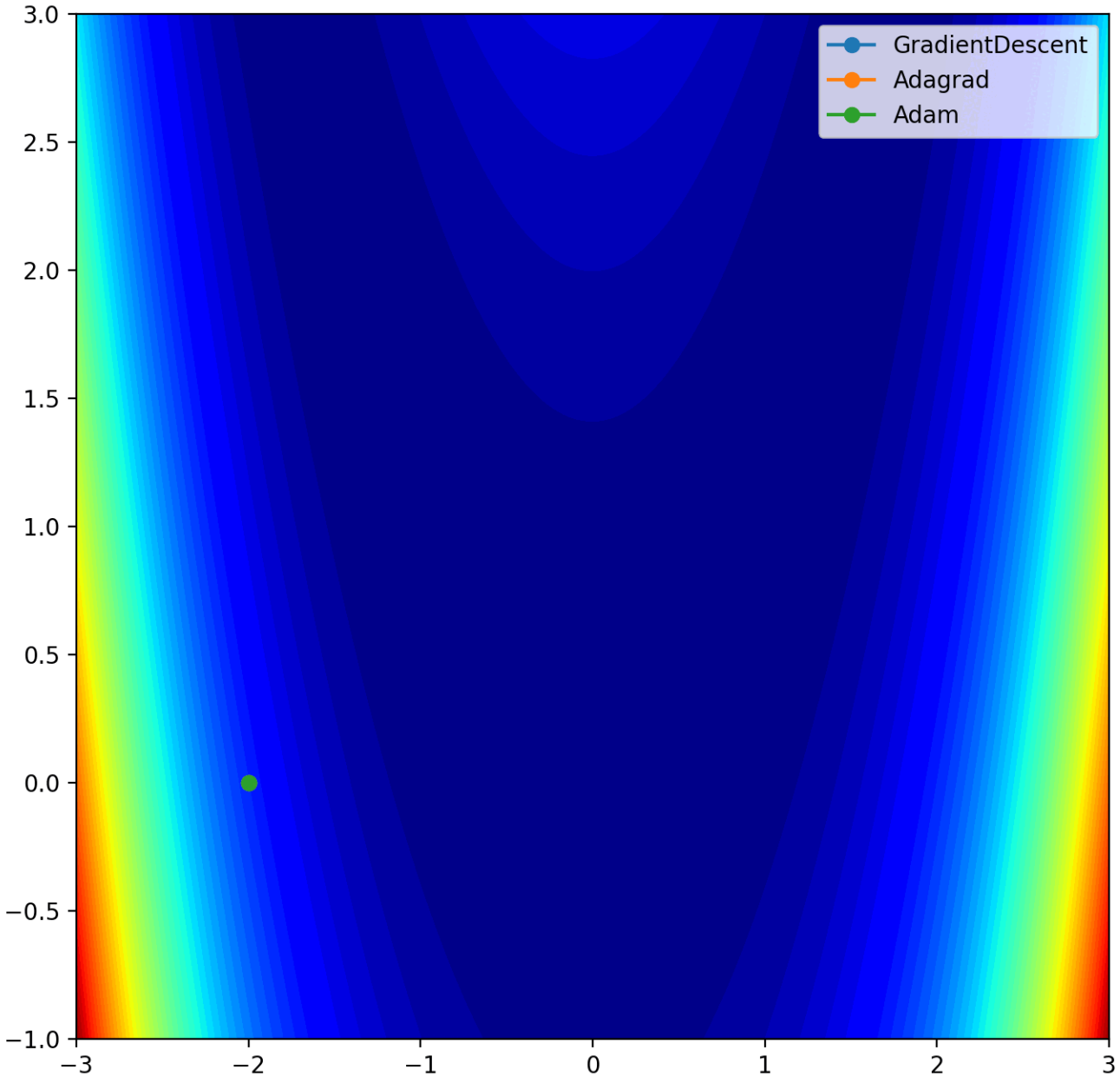
Adam:

$$\boxed{\begin{aligned} m_i &= \beta_1 m_{i-1} + (1-\beta_1)\nabla_\theta L\left(\theta^{(i)}\right) \\ v_i &= \beta_2 v_{i-1} + (1-\beta_2)\nabla_\theta L\left(\theta^{(i)}\right)^2 \end{aligned}} \quad \theta^{(i+1)} = \theta^{(i)} - \eta \frac{\boxed{\hat{m}_i}}{\sqrt{\boxed{\hat{v}_i}} + \epsilon}$$

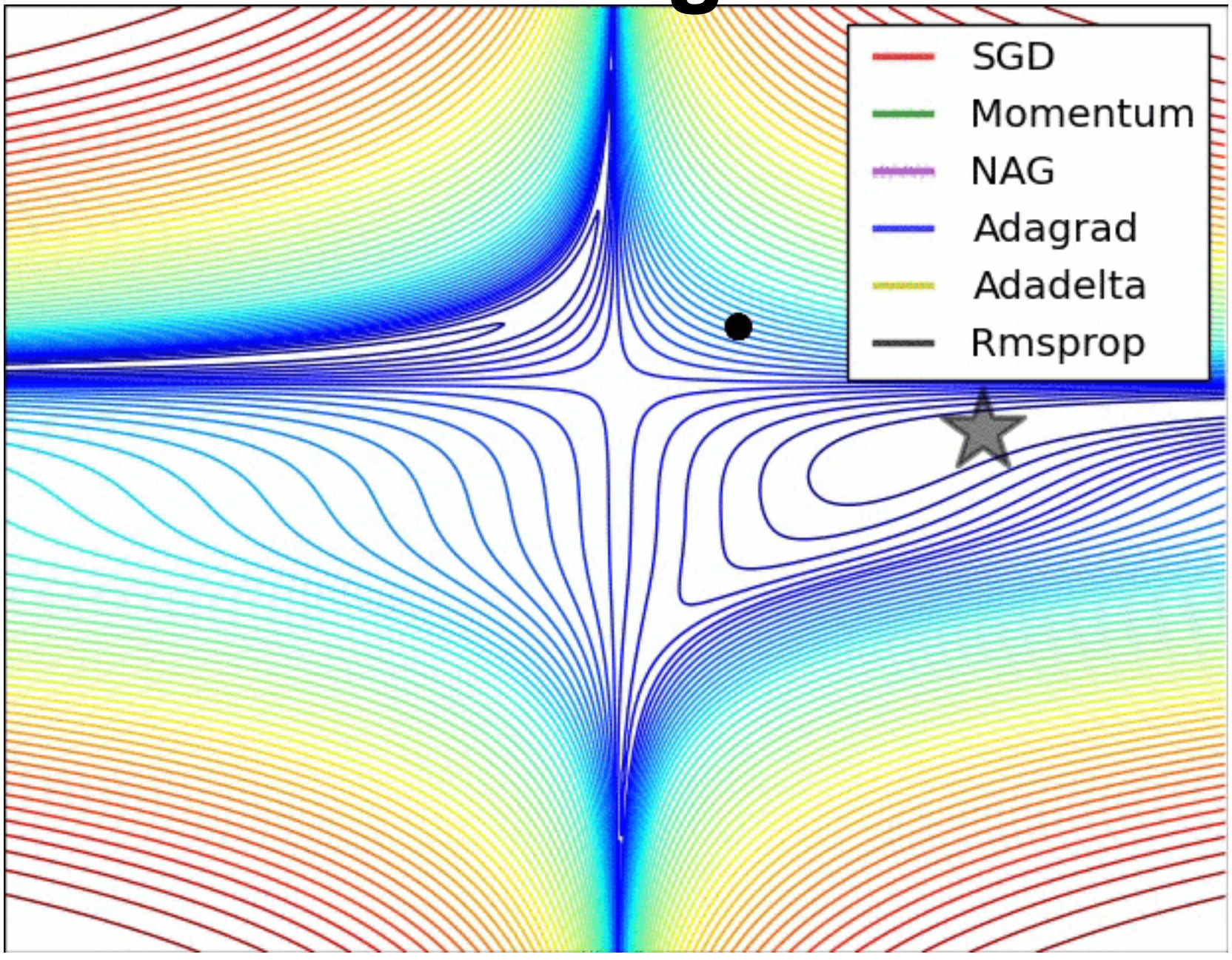
Estimate of first moment of gradient

Estimate of second moment of gradient

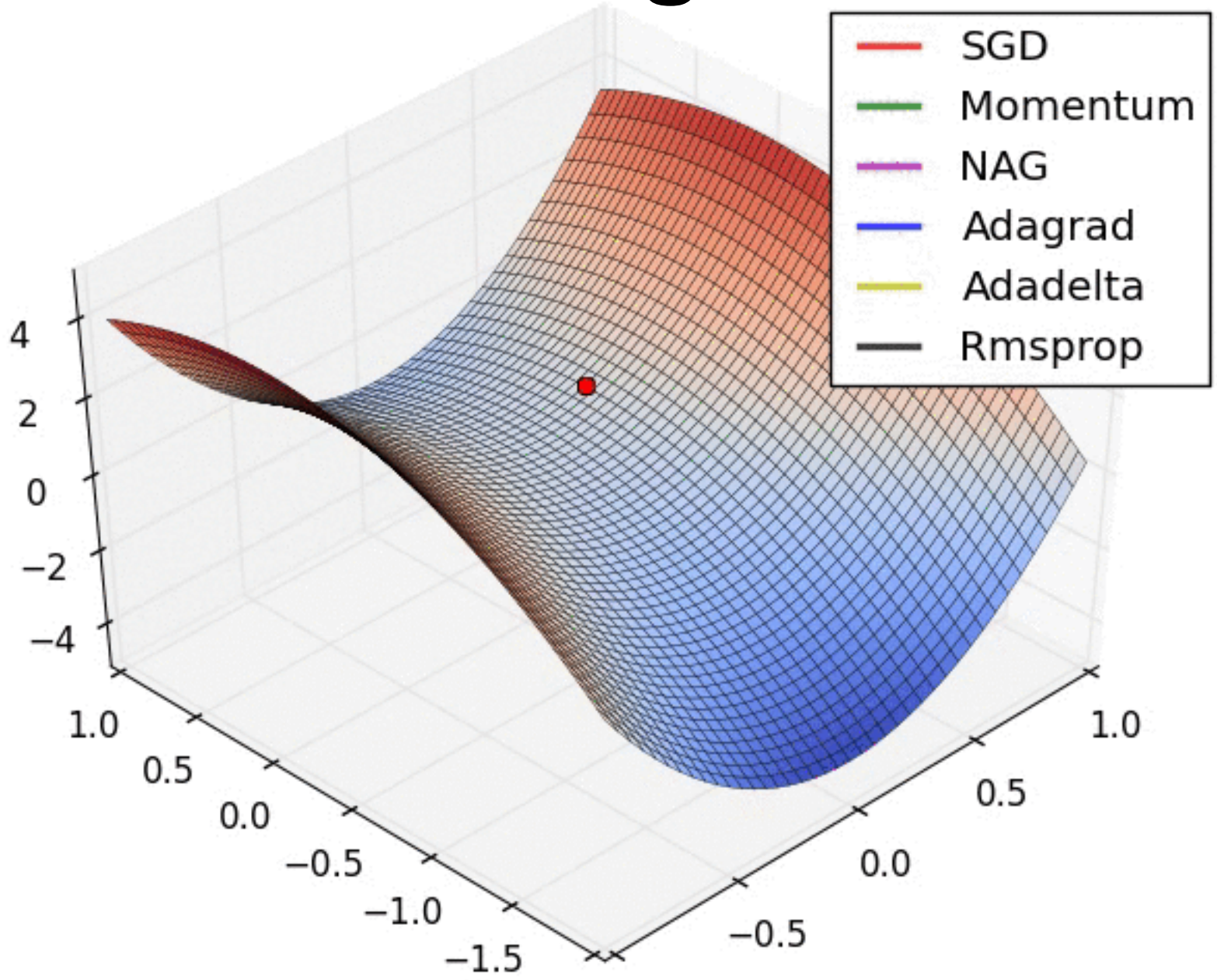
Automatic Learning Rate Selection



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:

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

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:

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

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

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:

$$\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_\theta \hat{L}\left(\theta^{(i-1)}\right), \textbf{with } \hat{L}(\theta) = \sum_{d \in B} \ell_\theta(d)$$

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 θ^* such that:

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

Training KG Embedding Models

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

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

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

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

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

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 θ

Why Using a Regulariser $\Omega(\theta)$?

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

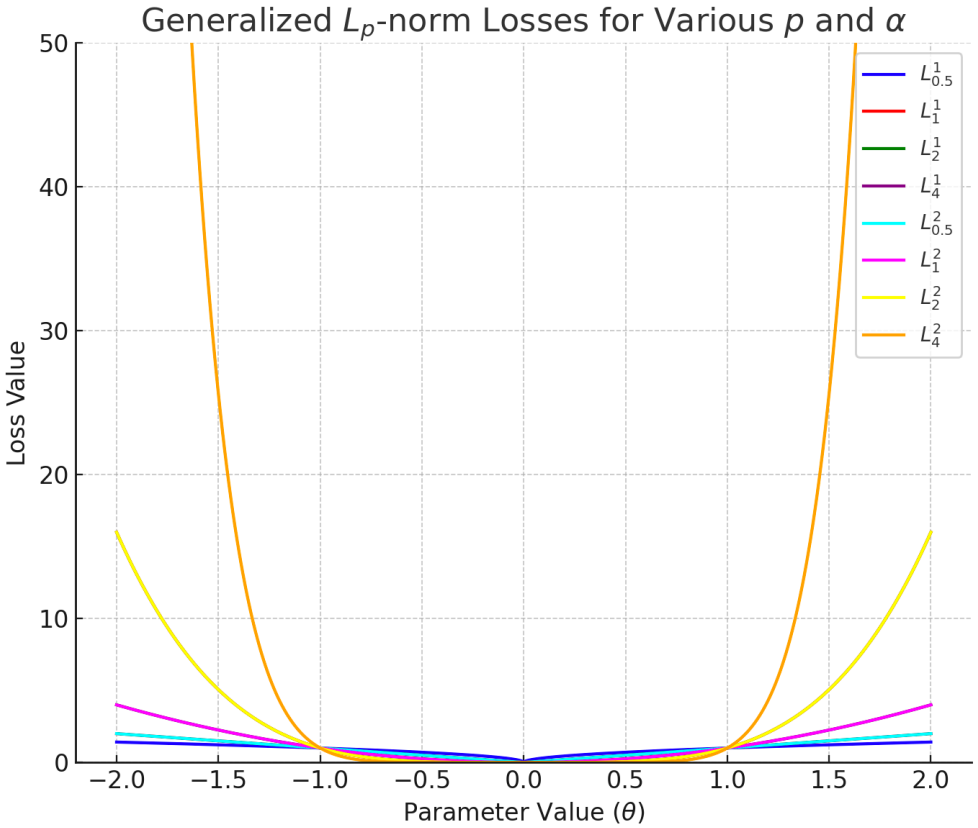
The training process can minimise the loss by artificially increasing the norm of the embeddings [Bordes et al. 2015]. How does the regulariser $\Omega(\theta)$ look like?

Why Using a Regulariser $\Omega(\theta)$?

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

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



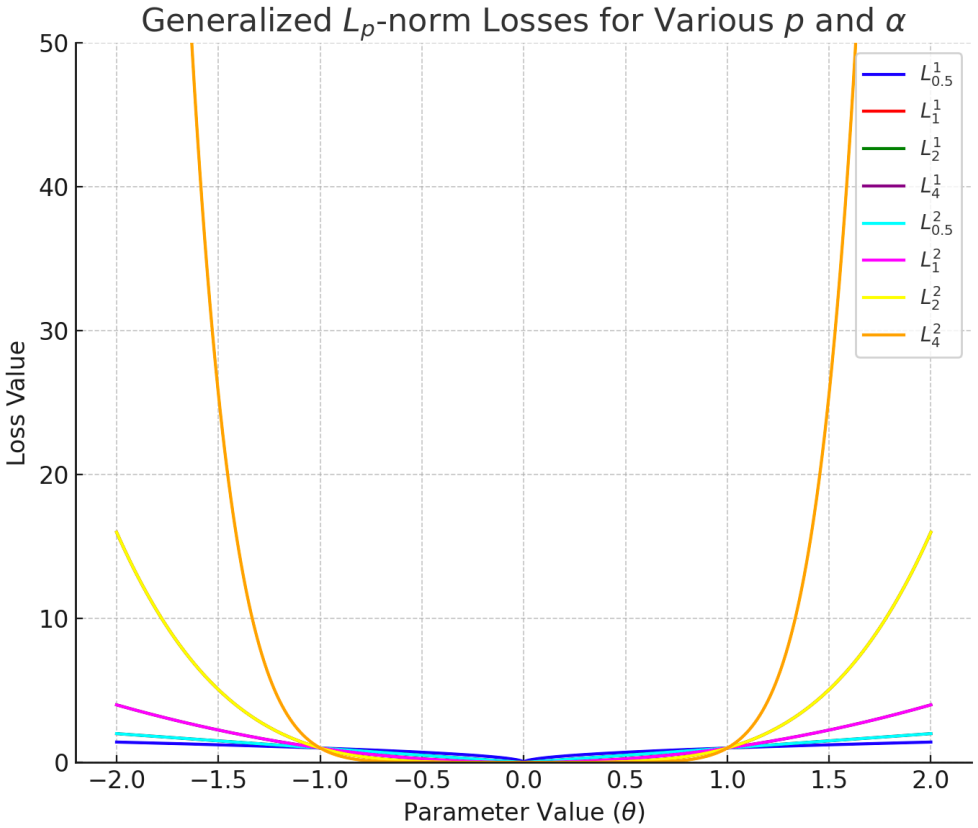
Why Using a Regulariser $\Omega(\theta)$?

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

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 $\Omega(\theta)$?

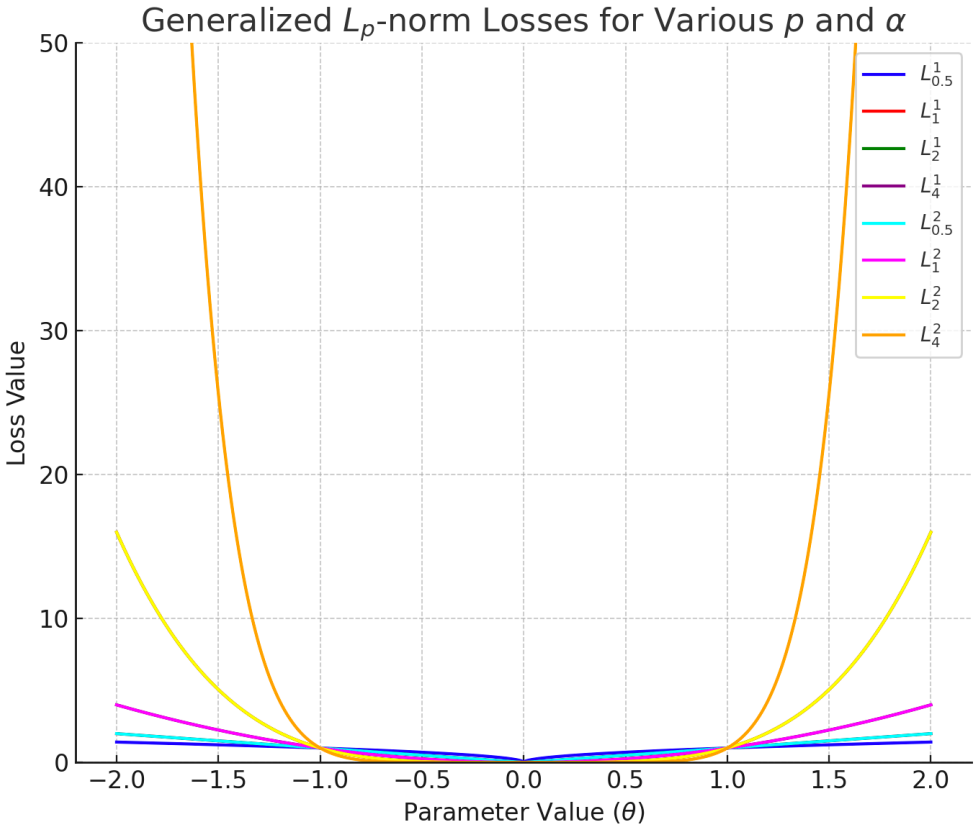
$$\theta^{(i)} = \theta^{(i-1)} - \eta \nabla_\theta \left[\hat{L}(B) + \lambda \boxed{\Omega(\theta)} \right]$$

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



Why Using a Regulariser $\Omega(\theta)$?

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

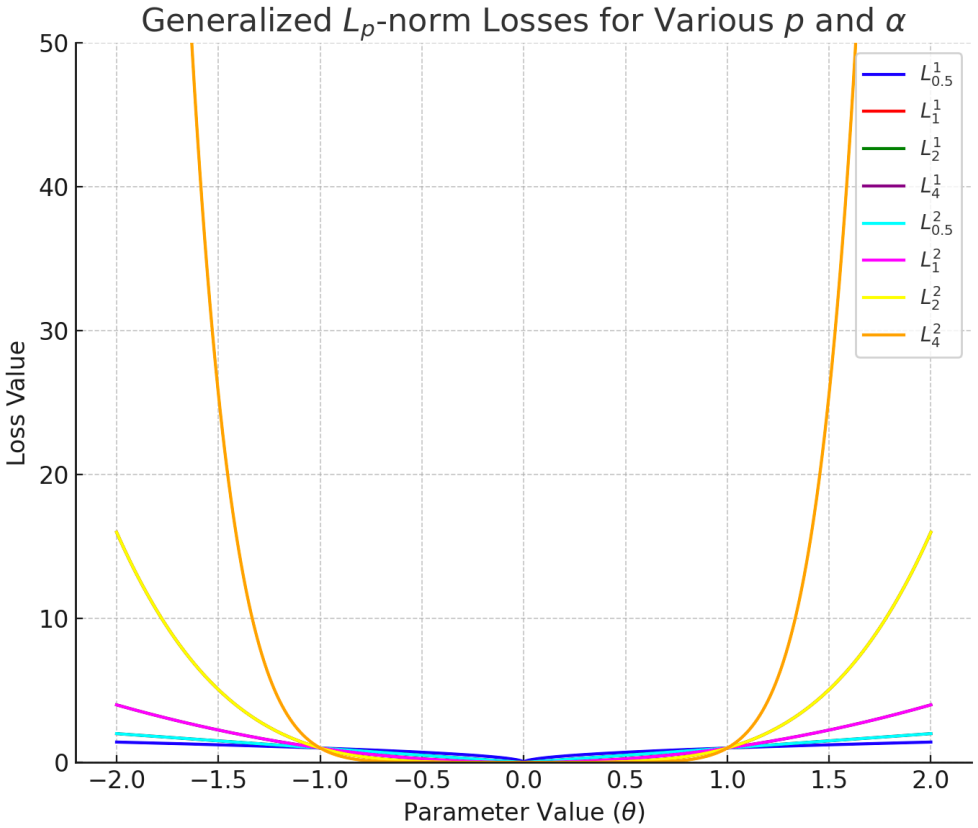
The training process can minimise the loss by artificially increasing the norm of the embeddings [Bordes et al. 2015]. How does the regulariser $\Omega(\theta)$ look like?

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; [..]



Results

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

Ruffinelli et al. You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings. ICLR 2020

Code!