**# Knowledge Graph Completion and Evaluation**

**1. Introduction to Knowledge Graph Completion**

Knowledge Graphs (KGs) are often **incomplete**, requiring methods to fill in missing facts. The process of completing a knowledge graph can be done in two ways:

* **Hard Decisions**: Directly determining whether a fact is true or false.
* **Soft Probabilistic Judgments**: Assigning a confidence score to the likelihood of a fact being true.

A **machine learning model** or **knowledge graph embedding technique** is used to infer missing facts by representing entities and relations in a **continuous space**.

**2. Embedding Representations in Knowledge Graphs**

To complete a KG, a system first learns **embedding representations** of the entities and relationships in the graph. There are two key approaches:

1. **Topology-based Representations**: Using the structure of the knowledge graph alone.
2. **Text-enhanced Representations**: Supplementing graph topology with textual information, such as aliases and descriptions of entities and relations.

Once the embeddings are learned, they can be used to **predict missing facts** by estimating confidence scores for inferred triples.

**3. Function f: Predicting Missing Knowledge**

Knowledge Graph Completion is modeled as a function **f**, which takes three inputs:

* **Subject (s)**
* **Relation (r)**
* **Object (o)**

During **training**, the model learns to assign belief scores to triples by:

* Providing (s, r, o) triples along with a **truth value** (either 1 for true or 0 for false).
* Optimizing the model’s embeddings to improve prediction accuracy.

During **testing**, one of the inputs may be withheld, and the model predicts its value based on learned embeddings.

**4. Representation of Entities and Relations**

Each entity and relation in a KG is mapped to a **continuous space** using different types of representations:

* **Points in high-dimensional space**
* **Vectors**
* **Displacements**
* **Projections (hyperplanes)**
* **Rotations**
* **Complex-valued embeddings**

These representations capture semantic and structural properties of the entities and relations.

**5. Scoring Function for Triple Validity**

The function **f** is used to evaluate the validity of a given triple (s, r, o):

* **High belief score**: If the fact is likely true (e.g., "Barack Obama is the President of the US").
* **Low belief score**: If the fact is likely false (e.g., "Barack Obama discovered the Theory of Relativity").

**6. Training the Model**

Training involves:

* Providing known triples labeled with **ground-truth belief scores** (1 for true, 0 for false).
* Learning representations (embeddings) to maximize predictive performance.
* Fitting embeddings using **optimization techniques** such as:
  + Gradient Descent
  + Stochastic Gradient Descent (SGD)
  + Adam Optimizer

**7. Inferring Missing Triples**

Once trained, the model can infer missing triples, similar to **link prediction** in social networks. Given a partial graph, the model estimates:

* Whether a **new relation** exists between two entities.
* The **most likely missing entity** in a triple.

Example in a **social network**:

* If Person P1 is connected to P2, P3, and P4, and we observe a new person P20, we predict whether P1 should be linked to P20 based on patterns in the network.

**8. Role of Neural Networks in Knowledge Graph Completion**

Since neural networks are **universal function approximators**, they can learn to approximate **f** for knowledge graph completion. However, designing an effective **f** is complex and involves:

* Choosing the right neural network architecture.
* Optimizing embeddings effectively.
* Balancing model complexity to prevent overfitting.

Over the past **five to six years**, researchers have developed numerous approaches to optimize **f**, focusing on:

* **Scalability** (handling large KGs)
* **Accuracy** (minimizing false predictions)
* **Interpretability** (making predictions understandable)

9. Notation and Representation in Knowledge GraphsEntities (e): General representation of an entity.

Subjects (s) and Objects (o): Entities take specific roles as subjects or objects.

This distinction matters because not all relationships are symmetric (e.g., "sibling of" is symmetric, but "CEO of" is not).

Vector Representations:

Entity Embedding: Represented as bold e or vector e.

Subject & Object Embeddings: Written as bold s and bold o.

Relation Embeddings: Represented as bold r.

The shapes of s, o, r can vary depending on the model (vectors, matrices, hyperplanes, etc.).

10. Function f and Confidence ScoresFunction f(s, r, o) produces a real number as output, representing confidence in the given triple.

This score is not necessarily a probability but a raw belief value.

To convert it into a probability, additional functions (e.g., sigmoid or softmax) may be applied.

11. Positive and Negative TriplesPositive Triple: A known true fact in the KG (e.g., "Albert Einstein discovered the Theory of Relativity").

Negative Triple: A fact known to be false.

Unknown Triple: Not appearing in the KG; could be either true or false.

Since KGs are incomplete, missing triples are not necessarily false.

Negative triples can be generated by:

Random sampling (high chance of being false).

Using heuristics to generate likely false triples.

12. SummaryEntities are represented as e, with subjects as s and objects as o.

Relations are represented as r, and embeddings are learned for all.

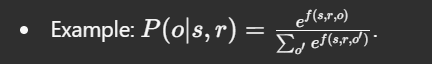
The function f(s, r, o) assigns a confidence score to triples.

True facts are positive triples, while false facts are negative triples.

Knowledge Graph Completion involves predicting missing triples with high accuracy using embeddings and neural network-based scoring functions.

**6. Probability Estimation and Normalization**

* The raw output of **f(s, r, o)** is not necessarily a probability.
* To convert it into a probability, a **softmax function** is applied:
  + Fix two elements (e.g., **s** and **r**) and compute the probability over all possible objects **o'**.



* + Ensures probabilities sum to 1 across possible candidates.
* Similar probability calculations can be done by fixing **r** and **o** and predicting **s**.

**7. Training the Model Using Loss Functions**

Training involves:

* **Maximizing log probability** of correct triples.
* Using a **negative log-likelihood loss function**:
  + Summing log probabilities over known triples.
  + Negating the sum to minimize the loss.
* **Negative Sampling**:
  + Replace entities in a triple with randomly sampled entities.
  + Example: Replace "Barack Obama" in (Barack Obama, BornIn, Hawaii) with another random entity (e.g., "Mahatma Gandhi").
  + The assumption is that the new triple is likely false.
* This uses a **local closed-world assumption**: randomly sampled facts are assumed to be false.

**8. Efficient Computation in Training**

* Computing full probability distributions over all possible entities is expensive.
* To approximate, we sample **k negative triples** instead of summing over all possible entities.
* We adjust probability estimates based on sampling frequency.

**9. Alternative Loss Functions: Margin-Based Ranking Loss**

Instead of probability-based methods:

* Define a **margin-based loss** to ensure correct triples have higher scores than incorrect ones.
* Use **hinge loss**:
  + max⁡(0,margin+f(s′,r,o′)−f(s,r,o))\max(0, \text{margin} + f(s', r, o') - f(s, r, o))
  + Ensures that valid triples have scores exceeding invalid triples by a fixed margin.
* This loss is inspired by **Support Vector Machines (SVMs)**.

**10. Batch-Based Training and Stochastic Optimization**

* Training is done in batches for efficiency.
* Each batch contains:
  + **Positive triples** (from the knowledge graph).
  + **Negative triples** (created via perturbation: replacing subjects or objects).
* Loss function accumulates errors over these batches and updates embeddings via **Stochastic Gradient Descent (SGD)**.

**11. Summary**

* Knowledge Graph Completion involves inferring missing triples by embedding entities and relations into a continuous space.
* Function **f(s, r, o)** assigns a confidence score to triples.
* Scores are converted to probabilities using **softmax normalization**.
* Training involves **maximizing log probability of correct triples** while **minimizing incorrect ones** using loss functions.
* **Negative sampling** is used to approximate training objectives efficiently.
* Alternative **margin-based loss functions** provide another way to optimize embeddings.
* **Batch-based training with stochastic optimization** improves learning efficiency.

By leveraging both **graph topology and textual features**, modern knowledge graph completion techniques improve accuracy and usability in real-world applications.