Assumption:

Open World Assumption: KGs operate under the open-world assumption, which means that the absence of information does not imply its negation. In other words, the graph may not explicitly represent all possible relationships or attributes for a given entity, but it does not mean that those relationships or attributes do not exist.

Under the closed world assumption, a knowledge base or knowledge representation system assumes that only the explicitly stated information is considered true or valid. If a statement or fact is not present in the knowledge base, it is assumed to be false or unknown. This assumption implies that the knowledge base is complete and contains all the relevant information for a given domain.

Triple classification:

a triple refers to a basic unit of information that consists of three components: subject, predicate, and object. Triples are used to represent relationships between entities in a structured format.

Here's a breakdown of the components of a triple:

Subject: The subject of a triple represents the entity or concept being described or referenced. It is typically represented as a node or entity in a knowledge graph. For example, in the triple (John, hasAge, 30), "John" is the subject.

Predicate: The predicate represents the relationship or attribute that connects the subject and the object. It describes the nature of the relationship between the entities. Predicates are often represented as edges or links in a knowledge graph. In the example (John, hasAge, 30), "hasAge" is the predicate.

Object: The object represents the value or entity associated with the subject through the predicate. It can be a specific value, an attribute, or another entity in the knowledge graph. In the example (John, hasAge, 30), "30" is the object.

If KG的矩阵大小不一样应该怎么处理？

Knowledge Graph Embedding. It is a technique used in the field of knowledge representation and knowledge graph mining to learn continuous vector representations (embeddings) of entities and relationships in a knowledge graph.

The goal of knowledge graph embedding is to map entities and relationships from a discrete symbolic representation to a continuous vector space, where similar entities and relationships are closer to each other in the vector space.

Here's an overview of how the embedding is typically done:

Entity Embedding:

Entities in the knowledge graph are assigned unique identifiers (e.g., URIs, integers).

Each entity is represented by a fixed-length vector in a continuous vector space. The dimensionality of the vector is a hyperparameter determined by the embedding method.

The objective is to learn entity embeddings that capture the semantic meaning and similarity relationships between entities.

Relationship Embedding:

Relationships (also known as predicates or edges) in the knowledge graph are also assigned unique identifiers.

Similar to entity embedding, each relationship is represented by a fixed-length vector in the continuous vector space.

The objective is to learn relationship embeddings that capture the semantic meaning and characteristics of the relationships.

Optimization:

The embedding process involves optimizing an objective function that measures the quality of the learned embeddings.

The objective function typically considers various factors, such as the similarity between related entities, the compatibility between entities and relationships, or the preservation of graph structure and semantics.

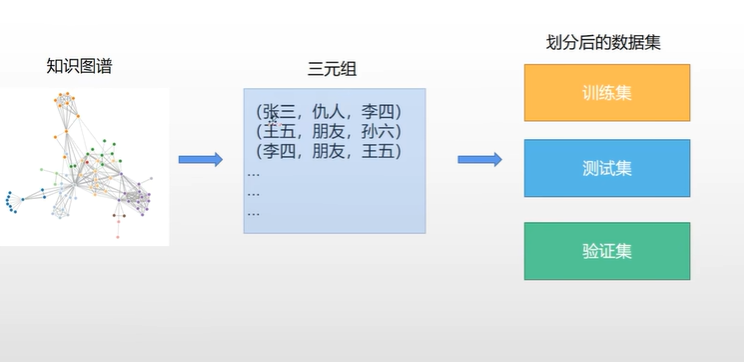
Optimization techniques like stochastic gradient descent (SGD) or its variants are commonly used to update the embeddings iteratively and minimize the objective function.

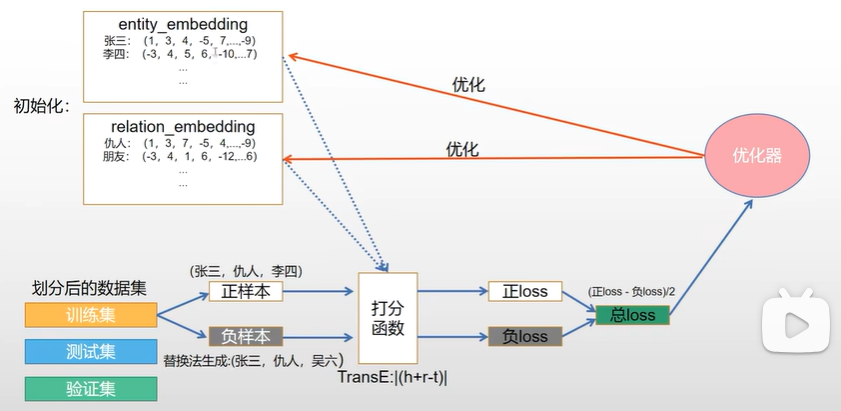
Training Data:

The training data for knowledge graph embedding typically consists of positive and negative examples.

Positive examples are derived from the existing knowledge graph, where the embeddings should reflect the relationships and patterns present in the graph.

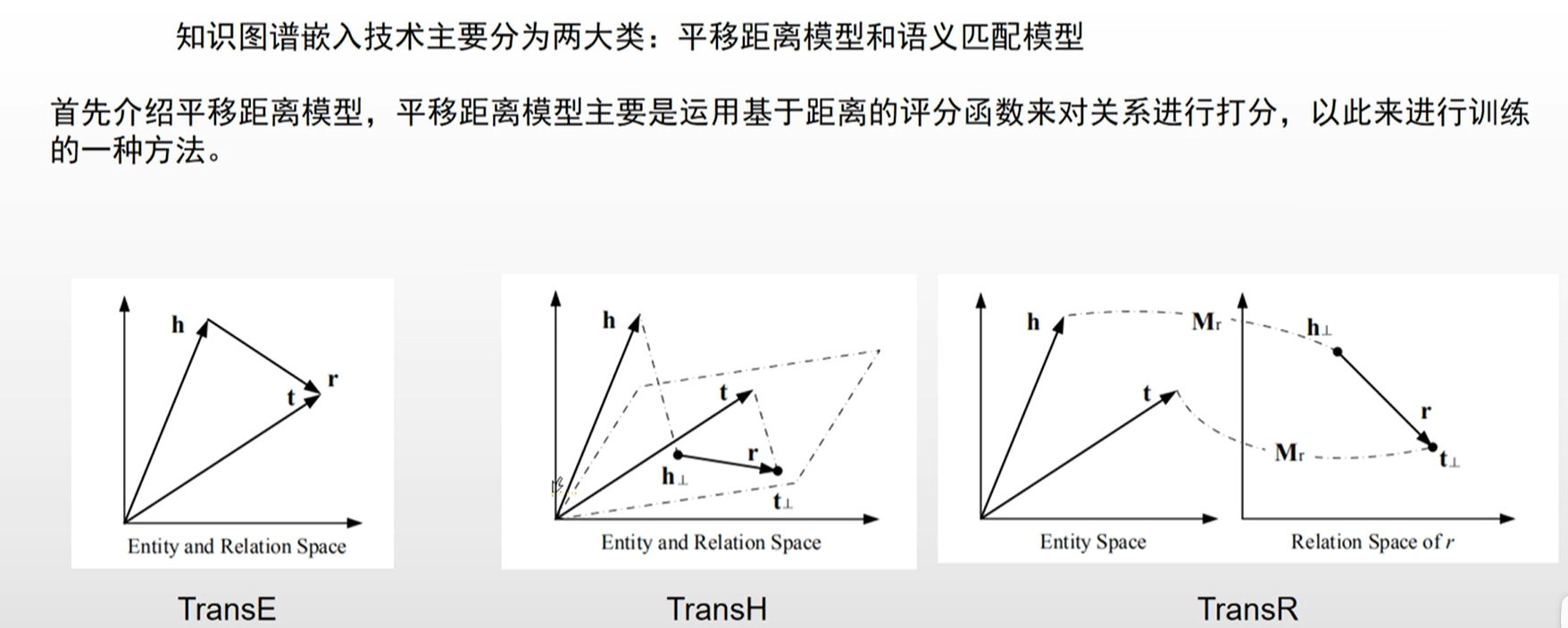
Negative examples are created by corrupting positive examples, introducing false relationships or randomizing entity pairs, to provide contrastive learning signals.

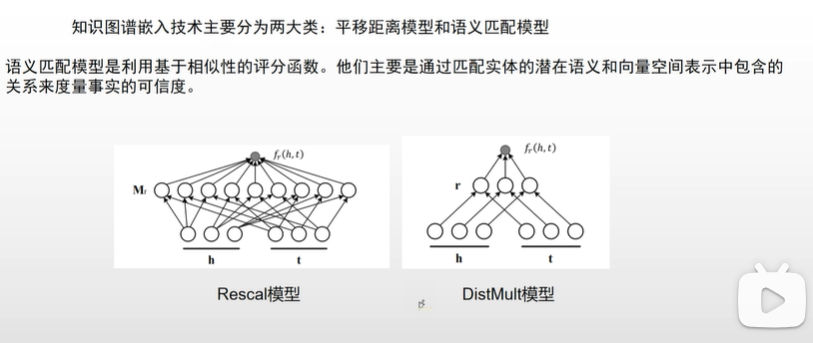




对节点和边的embedding做初始化

如何让正样本的score越好，负样本的score越差

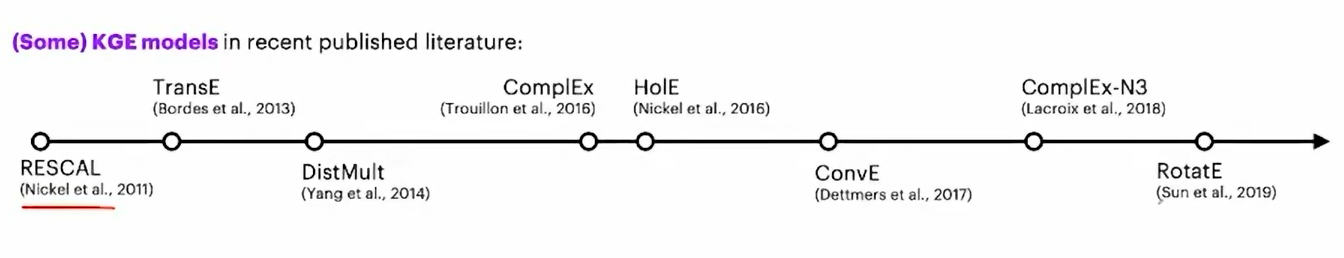


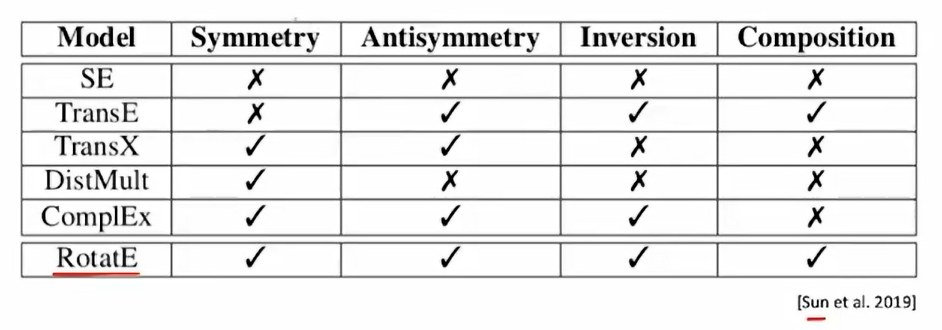


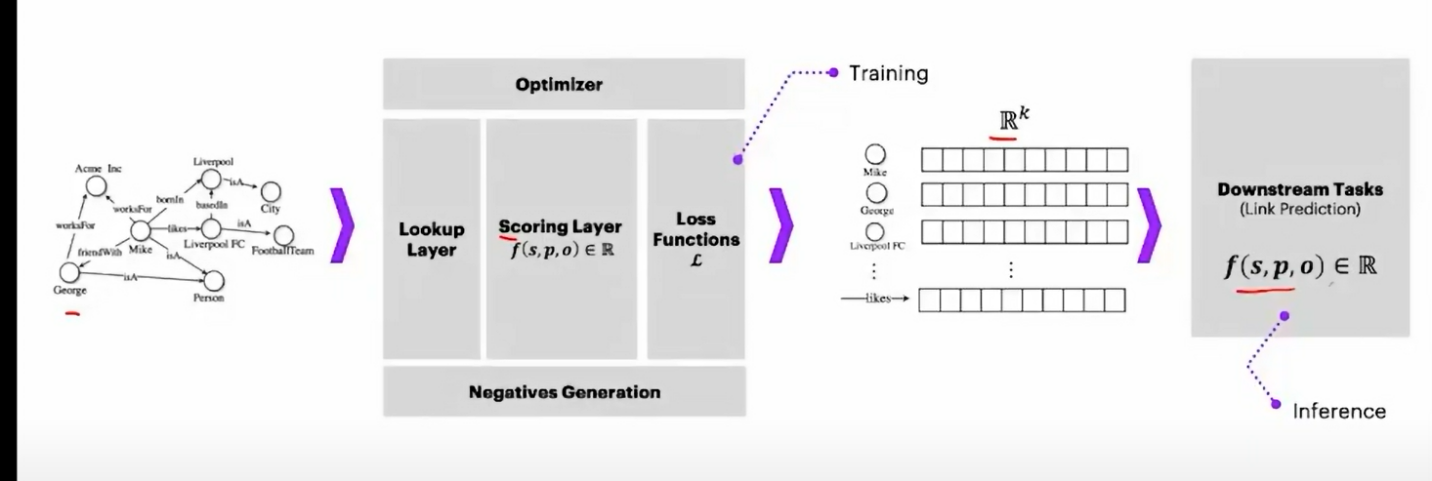
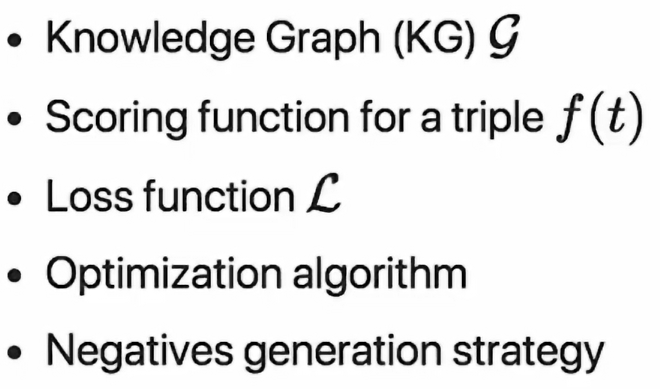
Various embedding methods exist, each with its own model architecture and **scoring function**.

Popular methods include TransE,TransH， TransR, Rescal,DistMult, ComplEx, RotatE, and more, each with its own approach to capture different aspects of the knowledge graph structure and semantics.

These methods often utilize scoring functions that measure the plausibility or compatibility between entity and relationship embeddings, with the aim of maximizing the scores for positive examples and minimizing them for negative examples.







Model Architectures:

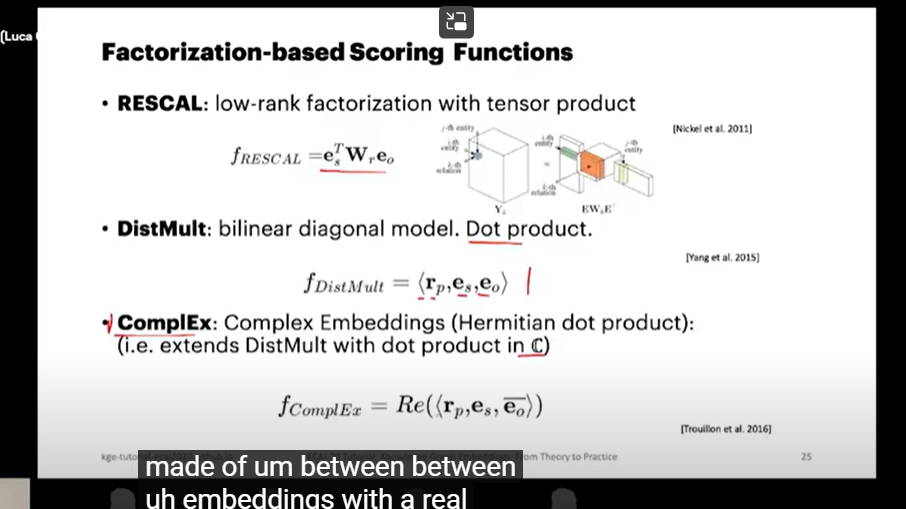
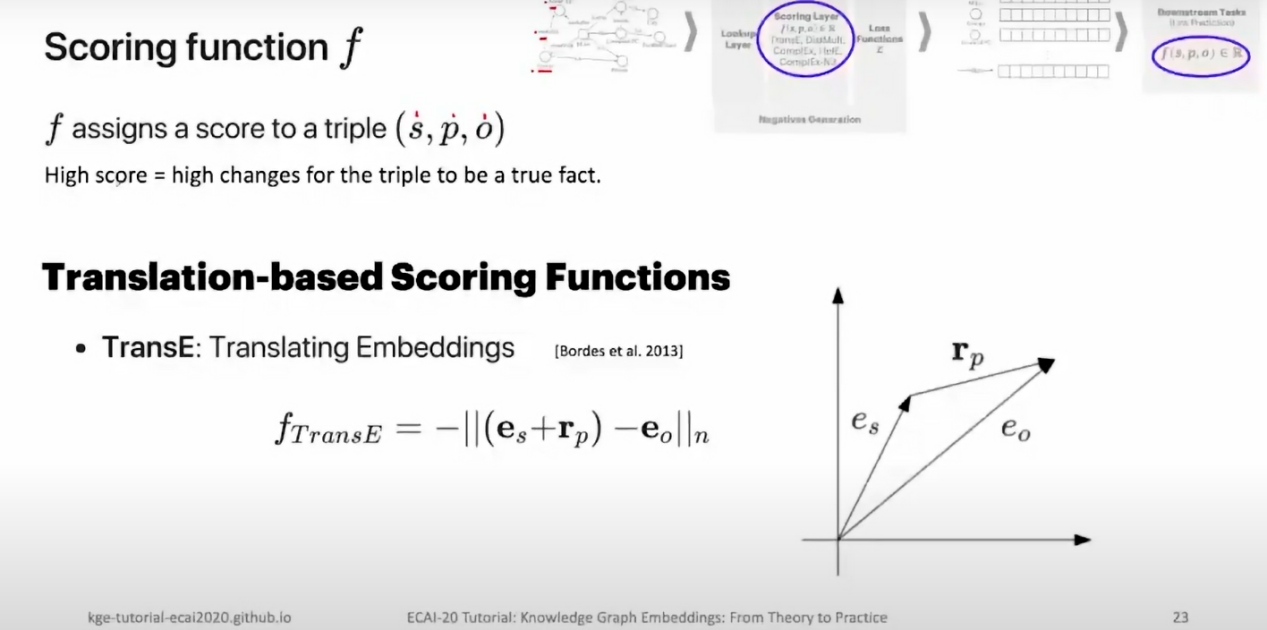
Initialization: Initialize the entity and relation embeddings. This is usually done randomly, but could also use prior information if available.

Scoring: For each triple in the training set, compute a score using a scoring function. The scoring function depends on the specific model being used. For example, the TransE model scores a triple by computing the distance between the head entity vector plus the relation vector, and the tail entity vector.

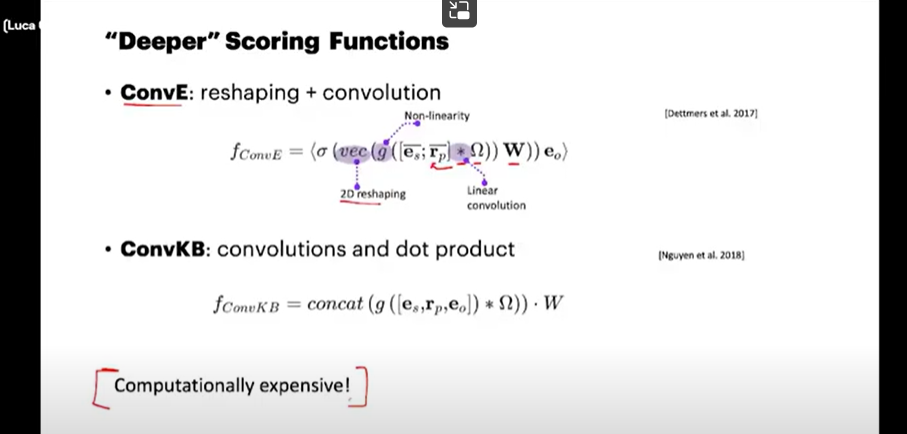
Negative Sampling: Generate negative examples (triples that are assumed to be false) for each positive example in the training set. This can be done using various strategies, such as uniform sampling or corrupting the head or tail entity of the positive triple.

Loss Computation: Compute the loss for the positive examples and negative examples. The loss encourages the model to assign higher scores to positive examples and lower scores to negative examples. Common loss functions include the margin-based ranking loss and the softmax loss.

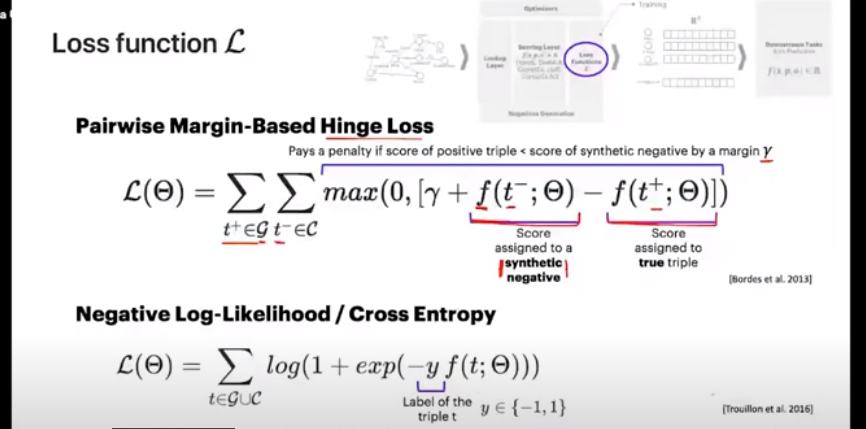
Backpropagation and Optimization: Use backpropagation to compute the gradients of the loss with respect to the embeddings, and update the embeddings using an optimization algorithm.



Dismult没有考虑到asymmetric relationship，complEx考虑到



LOSS FUNCTION



pairwise magin based hinge loss（in the scoring function that the least is best）

The intuition behind this loss function is that we want to ensure that the score of a positive triple (a triple that is known to be true in the KG) is lower than the score of a negative triple (a triple that is not in the KG, or known to be false) by at least a certain margin. If this is not the case, the loss is positive and the model parameters are updated to minimize this loss.

Here is the formal definition of the pairwise margin-based hinge loss:

L = max(0, margin + score(A, R, B) - score(A', R', B'))

In this formula:

L is the loss.

score(A, R, B) is the score of a positive triple (A, R, B), where A is the head entity, R is the relation, and B is the tail entity.

score(A', R', B') is the score of a negative triple (A', R', B').

margin is a hyperparameter that determines how much larger the score of a positive triple should be compared to the score of a negative triple.

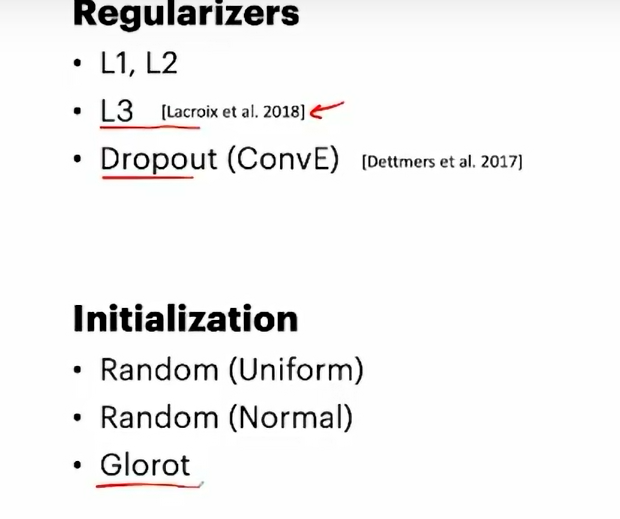
If the score of the positive triple plus the margin is greater than the score of the negative triple, the loss is zero. If not, the loss is the difference between the score of the negative triple and the sum of the score of the positive triple and the margin.

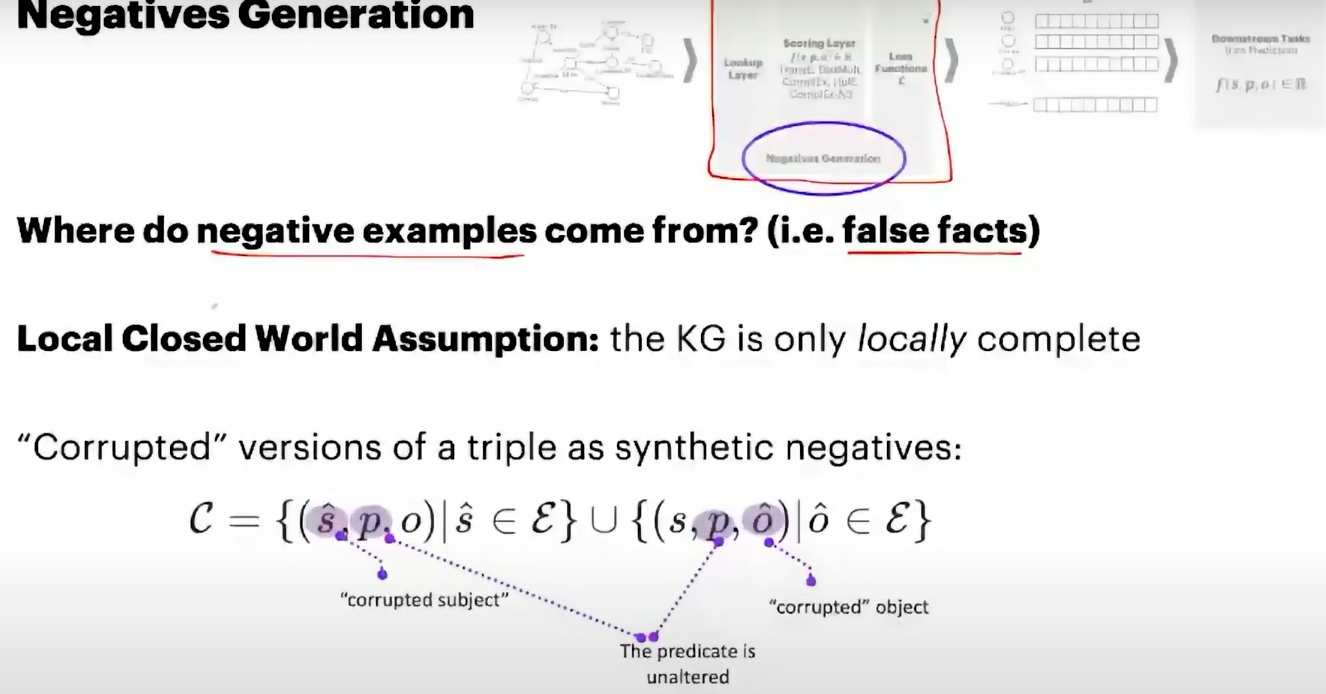
（in the scoring function that the greatest is best）

L = max(0, margin + - score(A', R', B')-score(A, R, B))

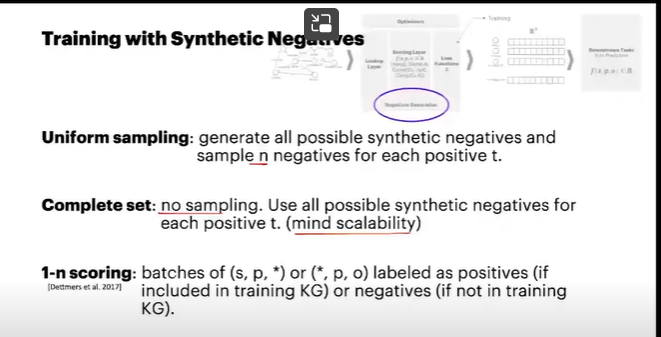
negative log likelihood or cross entropy

Binary corss entropy





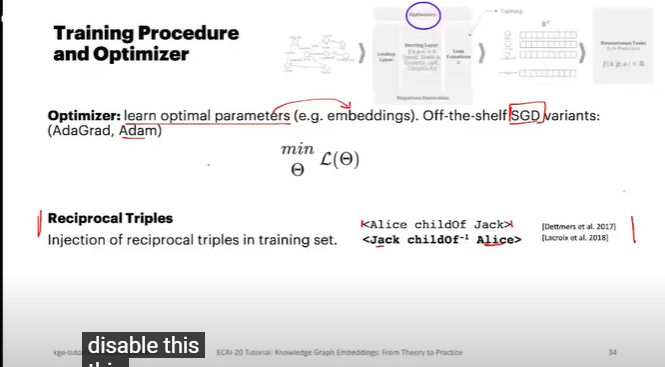
Local closed world assumption



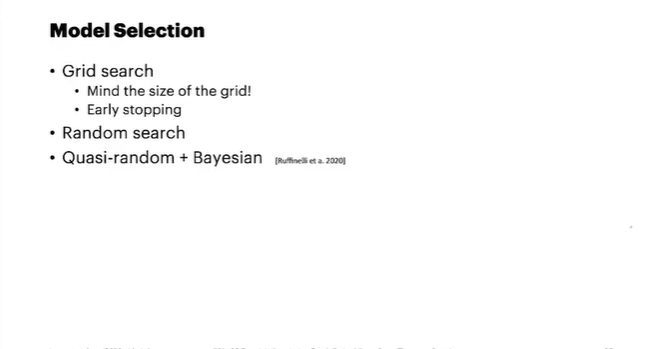
Unifrom sampling

****Complete set (without sampling)****

Convolution embedding paper use 1-n scoring

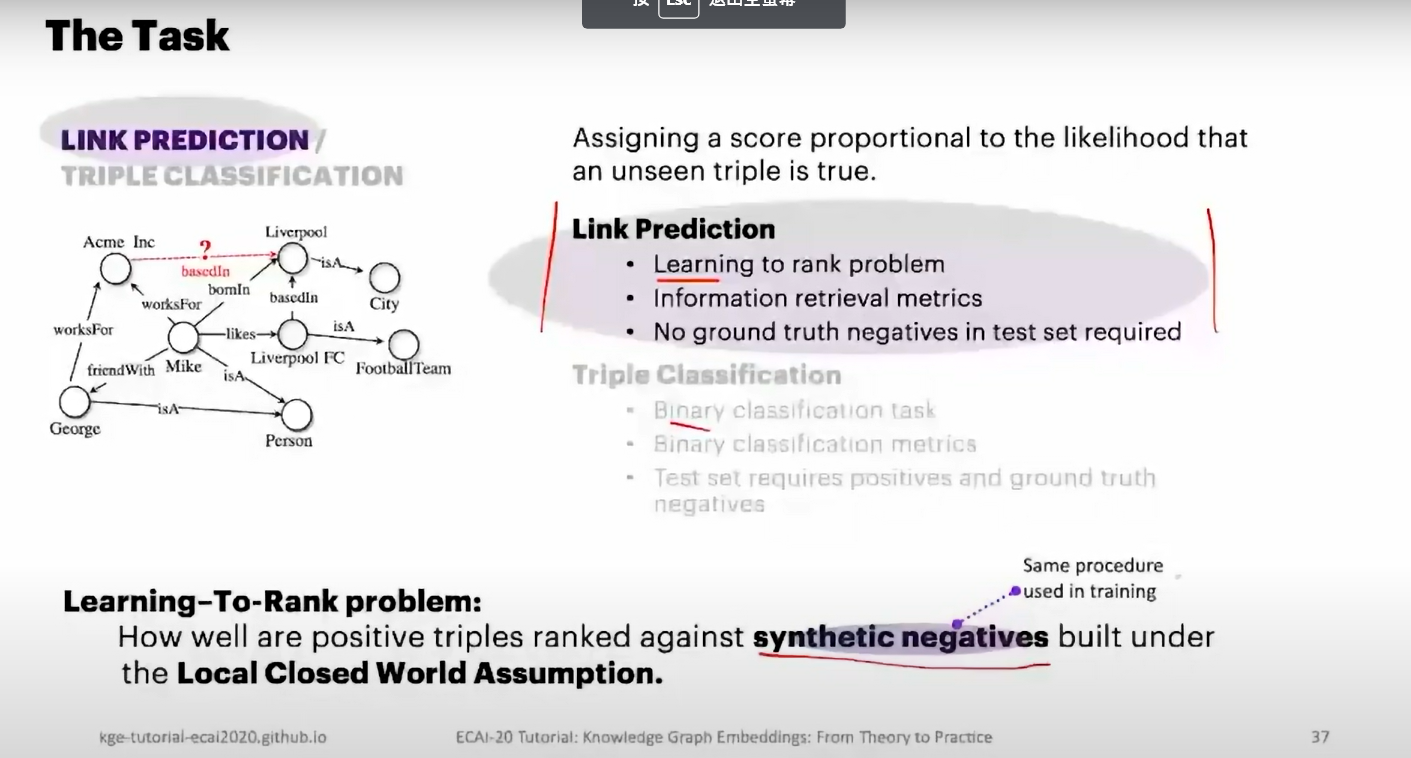


****Reciprocal Triples****



Grid search 最常用

Hyperparameter



link prediction :

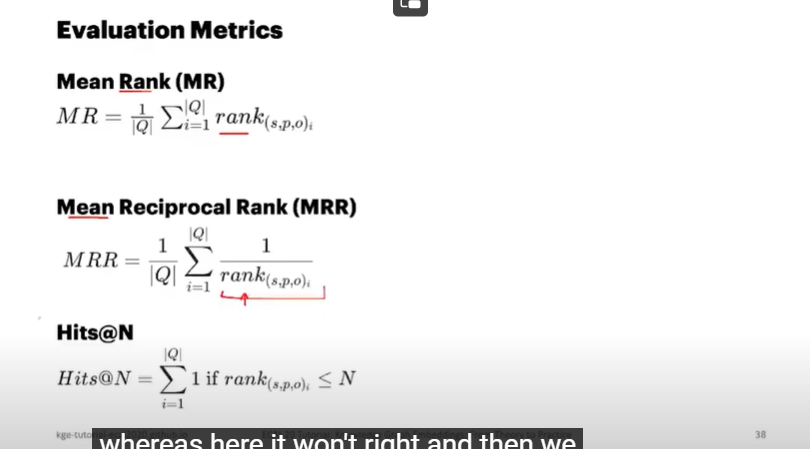
1. learning to rank problem:how well are positive triple ranked against synthetic negatives built under LCWA，why they ranked?
2. 2.information retrieval metrics
3. 3.no ground truth negative in test set required

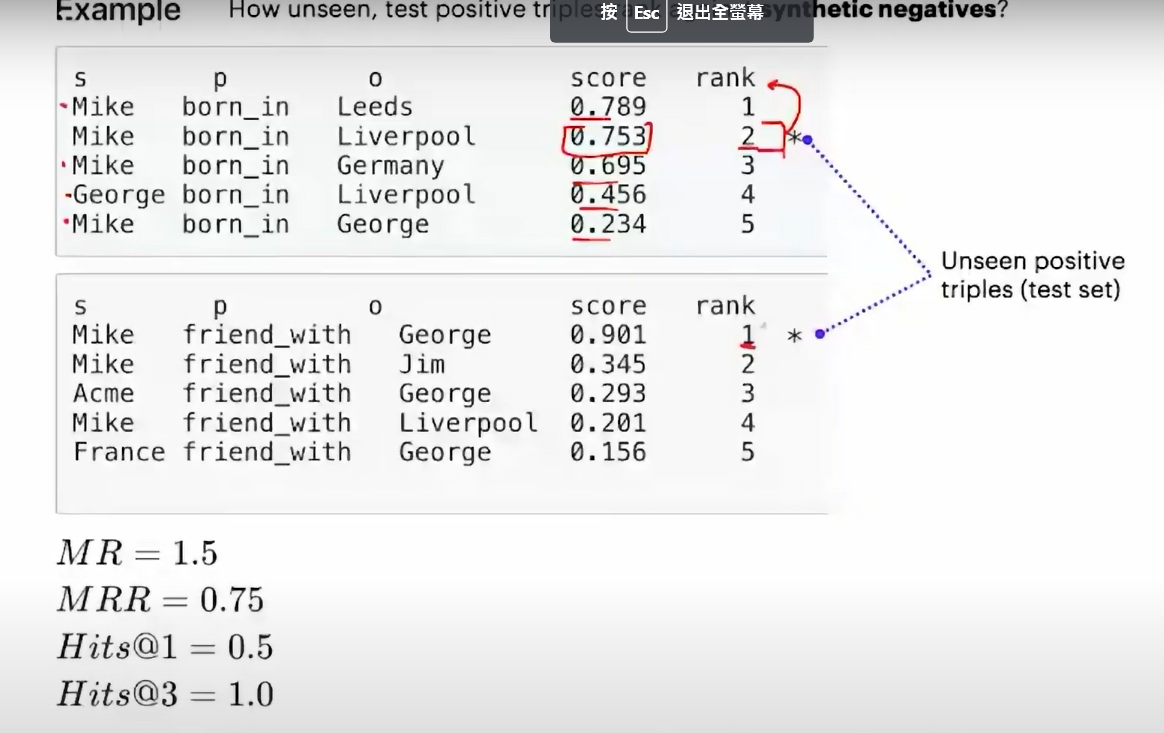
evaluation metric:

1.mean rank

2.mean reciprocal rank

3.Hits@N





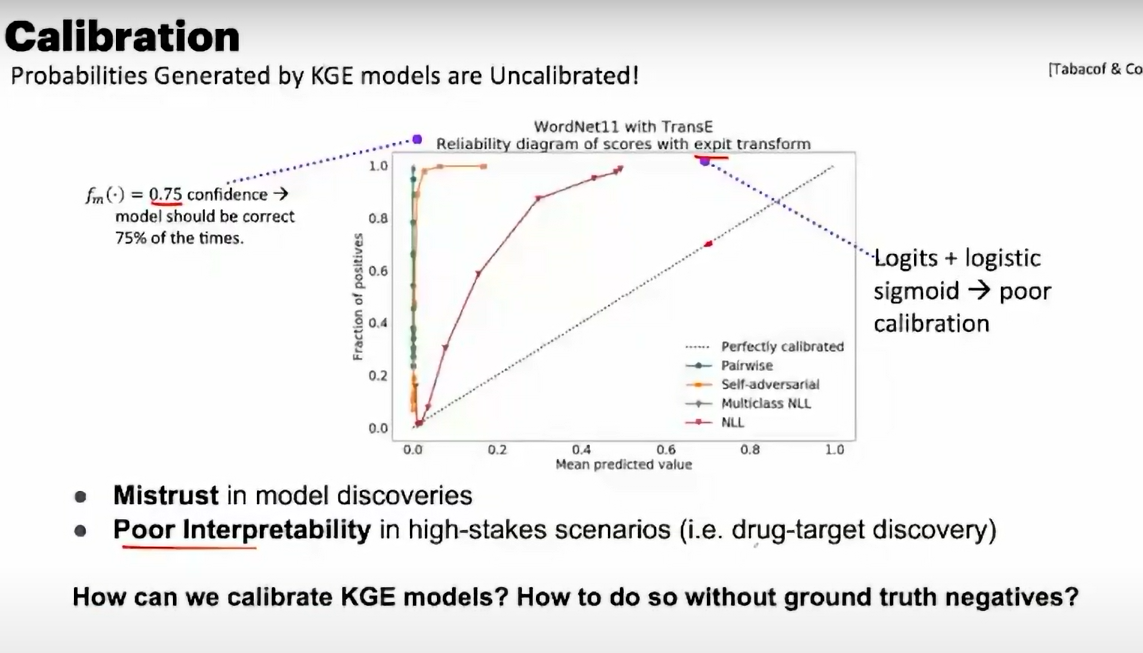
MR is 2 + 1/2 =1.5

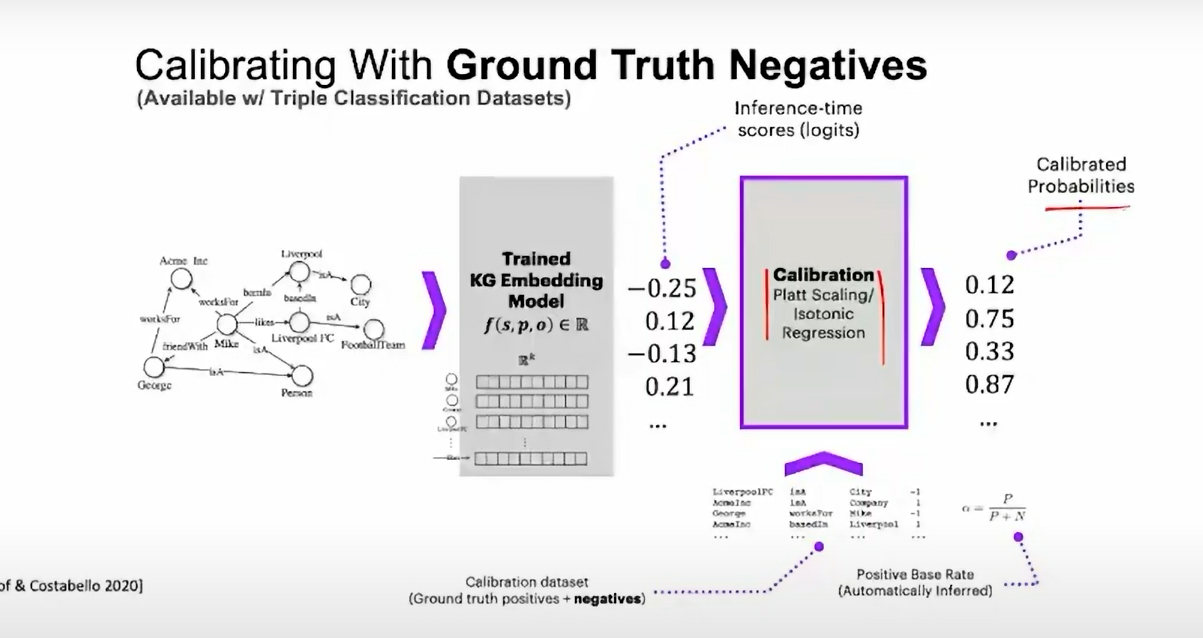
MRR is (1/2 + 1/1)/2 = 0.75

Hits@1 meausre how many times how many triple show up in the top 1 position, =1/2

Hits@3 meausre how many times how many triple show up in the top 3 position, = 1两次都有出现

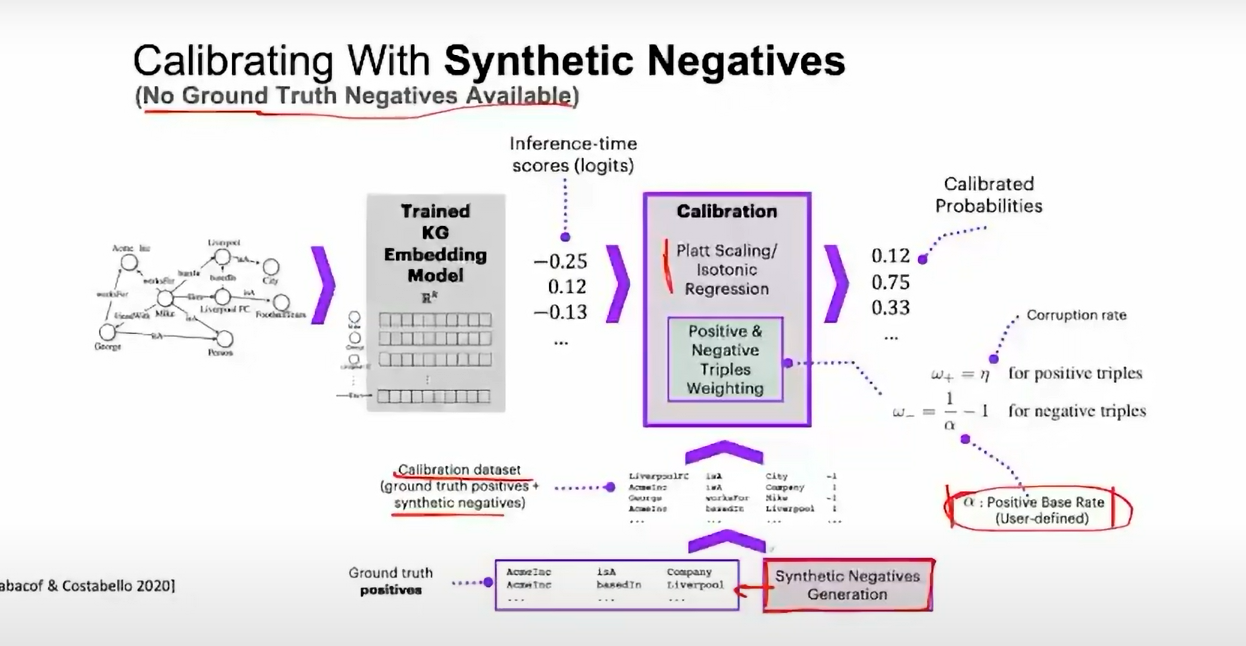
Calibration



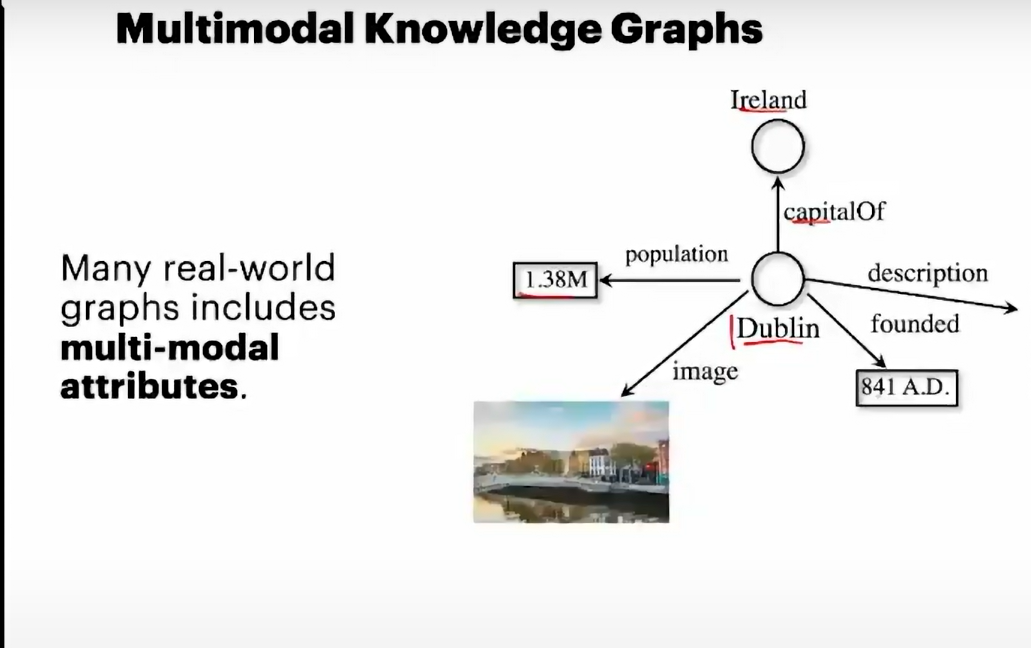


Solution:

可以加上GTN



Positive base rate is hyperparameter



Multi-Knowledge Graph (KG) Embedding refers to the process of representing multiple knowledge graphs in a shared embedding space. This is useful when dealing with knowledge graphs in different domains, languages, or granularities, or when trying to align or integrate multiple knowledge graphs.

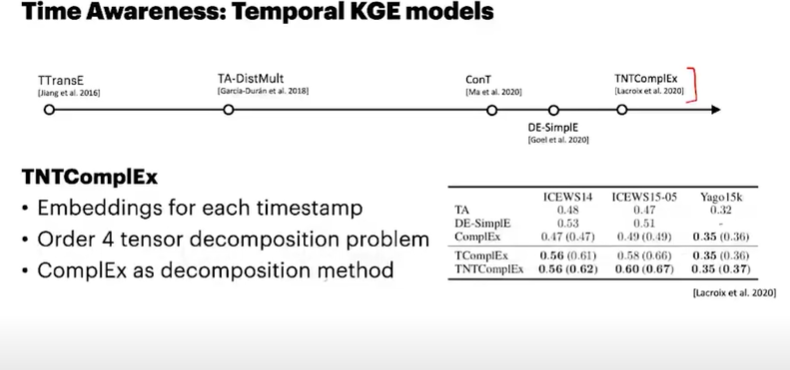
There could be different ways to perform multi-KG embedding:

Joint Embedding: All the knowledge graphs are embedded jointly in a shared embedding space. The same entities or relations across different knowledge graphs are enforced to have the same or similar embeddings. This allows for information to be shared across different knowledge graphs and can help improve the embeddings, especially for entities or relations that have little information in one knowledge graph but more information in another.

Aligning or Transferring Embeddings: The knowledge graphs are embedded separately, and then the embeddings are aligned or transferred from one knowledge graph to another. This usually involves learning a mapping function that maps the embeddings from one knowledge graph to another.

In Multi-Knowledge Graph

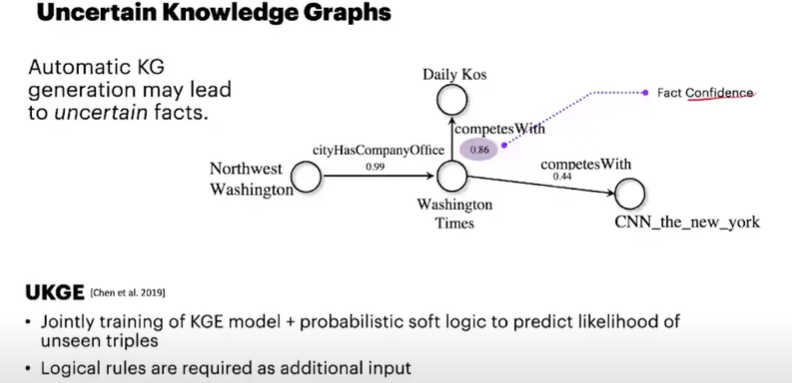
Can we design model that leverage timestamp to improve the prediction



It generating embedding for each timestamp using a order for tensor decomposition apporach leveraging the model using ComplEx

Uncertain Knowledge graph is KGE model that with Fact confidence,This can be useful in situations where the knowledge graph is generated automatically from unstructured data sources, such as text, and there is uncertainty about the correctness of the extracted facts.

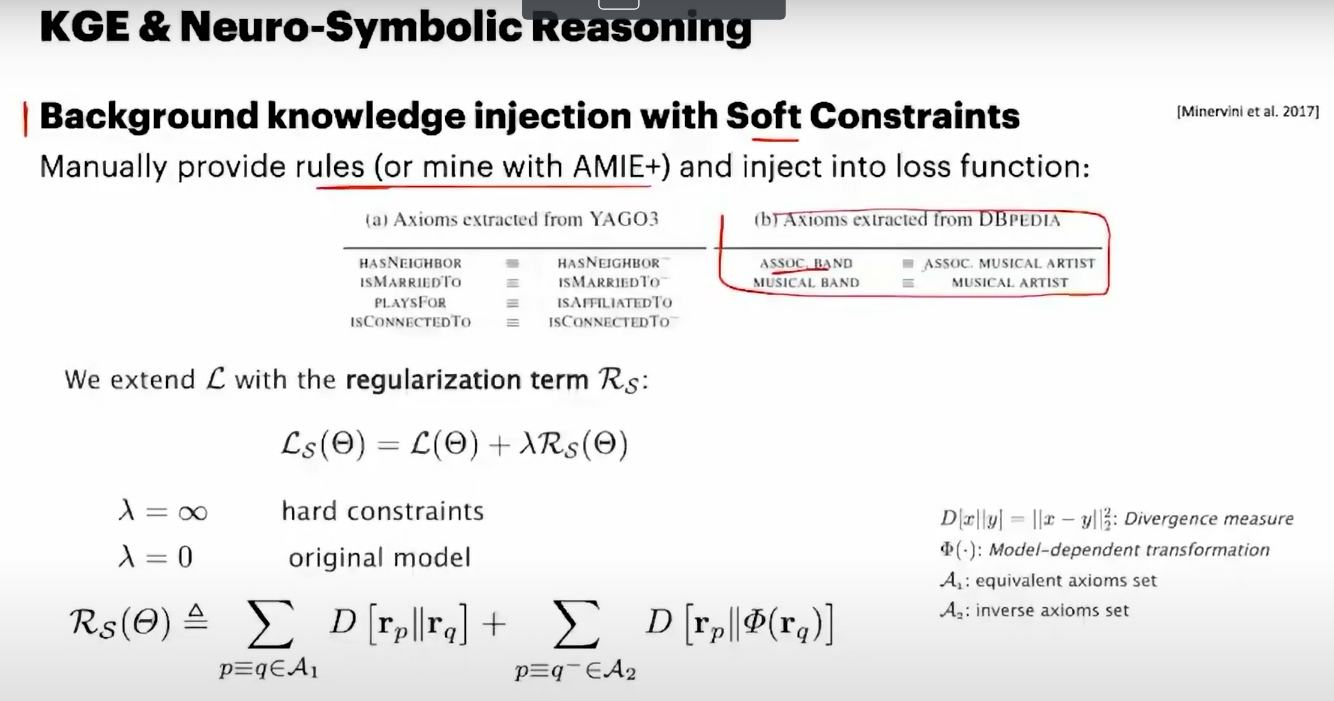
Logic rule:



Robustness of KG, "KG suffer from adversarial modification"

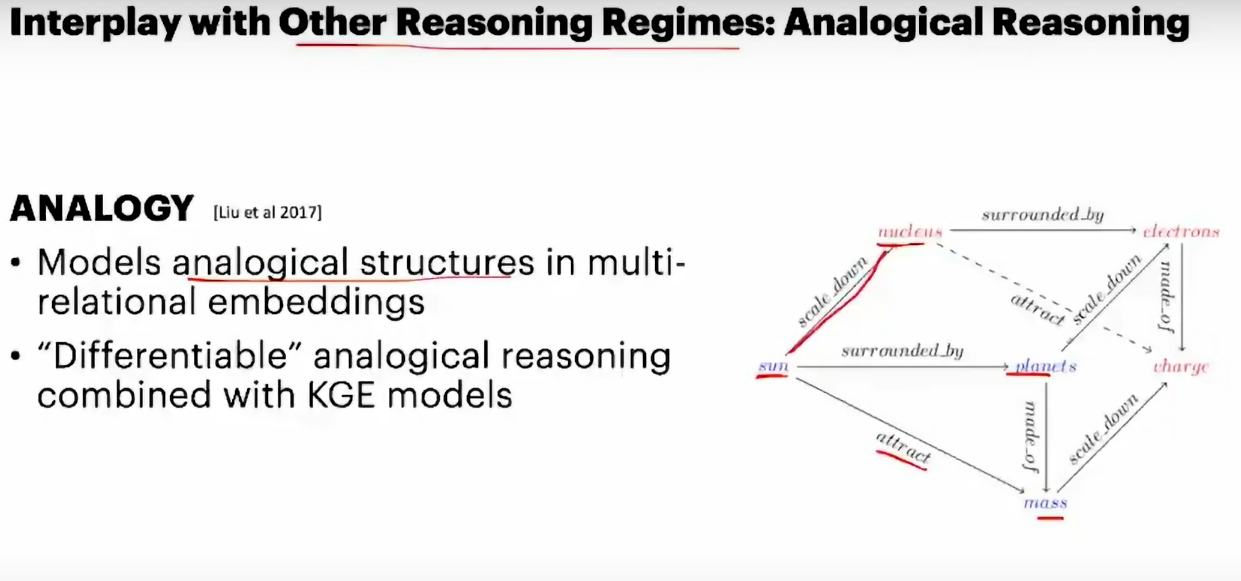
KGE & neuro symbolic reasoning

knowledge graphs can be viewed as the discrete symbolic representations of knowledge, reasoning on knowledge graphs can naturally leverage the symbolic techniques. However, symbolic reasoning is intolerant of the ambiguous and noisy data. On the contrary, the recent advances of deep learning promote neural reasoning on knowledge graphs, which is robust to the ambiguous and noisy data, but lacks interpretability compared to symbolic reasoning.



The concept of background knowledge injection with soft constraints is one way to incorporate symbolic reasoning into a KGE model. The idea is to take certain rules or pieces of knowledge (which might be provided manually or mined from the data using a tool like AMIE+) and encode these rules as constraints that the KGE model should try to satisfy.

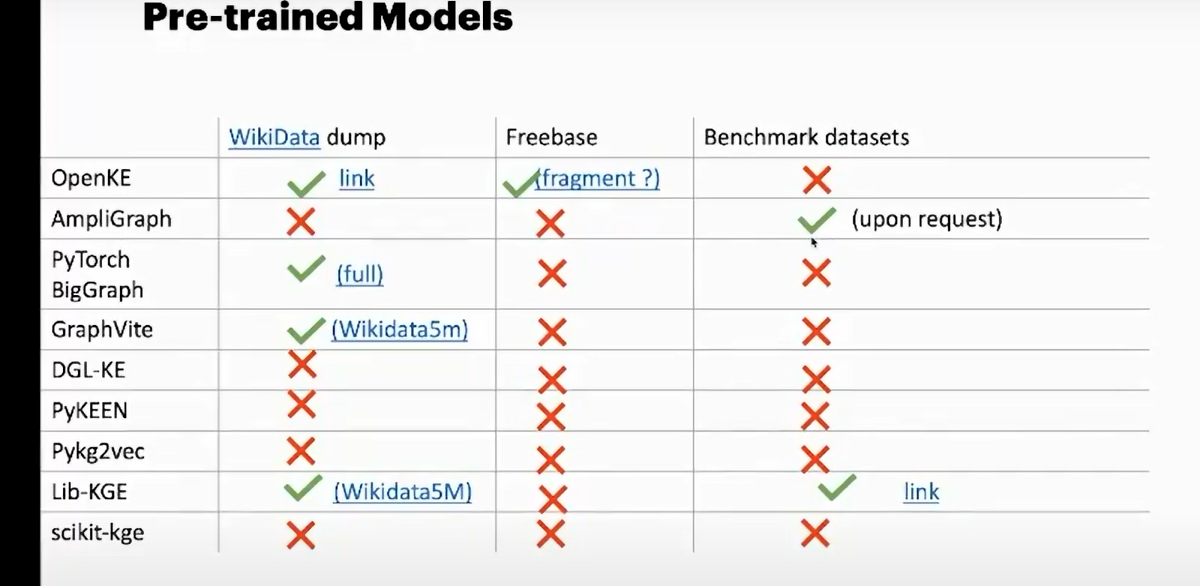
In the regularization term R\_s, "s" could stand for "soft", indicating that these are soft constraints. These constraints are added to the original loss function L of the model, resulting in a new loss function L + λR\_s. The λ parameter controls the trade-off between the original loss and the soft constraints

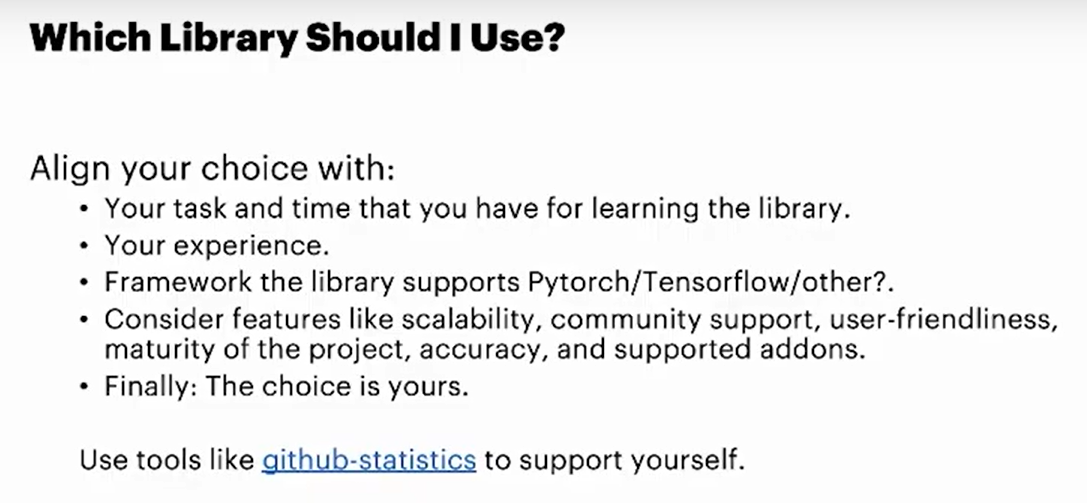


Libraries









Coding

import tensorflow as tf

import ampligraph

# Benchmark datasets are under ampligraph.datasets module

from ampligraph.datasets import load\_fb15k\_237

# load fb15k-237 dataset

dataset = load\_fb15k\_237()

## **Train the model**

# Import the KGE model

from ampligraph.latent\_features import ScoringBasedEmbeddingModel

# you can continue training from where you left after restoring the model

tensorboard\_callback = tf.keras.callbacks.TensorBoard(log\_dir='./transe\_train\_logs')

# create the model with transe scoring function

model = ScoringBasedEmbeddingModel(eta=5,

k=300,

scoring\_type='TransE')

* **eta** often denotes the number of negative samples for each positive one during training.
* **k** is typically the size of the embedding space, i.e., the number of dimensions in the vectors that the model learns to represent entities and relationships.
* **scoring\_type='TransE'** means that the model uses the TransE (Translate Embeddings) scoring function. TransE is a popular method for knowledge graph embedding that models relationships by interpreting them as translations in the embedding space.

# you can either use optimizers/regularizers/loss/initializers with default values or you can

# import it and customize the hyperparameters and pass it to compile

# Let's create an adam optimizer with customized learning rate =0.005

adam = tf.keras.optimizers.Adam(learning\_rate=0.005)

# Let's compile the model with self\_advarsarial loss of default parameters

model.compile(optimizer=adam, loss='self\_adversarial')

# fit the model to data.

model.fit(dataset['train'],

batch\_size=10000,

epochs=10,

callbacks=[tensorboard\_callback])

# the training can be visualised using the following command:

# tensorboard --logdir='./transe\_train\_logs' --port=8891

# open the browser and go to the following URL: http://127.0.0.1:8891/