Training Camp on "**Knowledge Graph Completion**"

– Sapienza University, M.Sc. Degree in Data Science –

Fabio Galasso, Laura Laurenti, Alessio Sampieri
Sapienza University of Rome

Ilaria Bordino, Francesco Gullo, Lorenzo Severini
UniCredit Services
"AI, Data & Analytics ICT" Department
"Applied Research & Innovation" unit

https://www.kaggle.com/c/unicredittrainingcamp/overview/lectures

June 30th – July 2nd, 2021

Day 2: Knowledge Graph Embedding (KGE)

Schedule

- Graph embedding
  - General problem
  - Embedding of vanilla graphs (hints)
- KGE
  - Problem definition
  - Overview of some of the most popular state-of-the-art KGE methods (TransE, TransH, TransR, DistMult)
- KGE-based KGC
  - Prediction by looking solely at the KGE score of a triple (threshold-based, ranking-based)
  - Prediction by training a downstream classifier on KGEs
- Lab
  - Introduction to PyKEEN (Python KnowlEdge EmbeddiNgs) framework
  - Overview of resources available in PyKEEN (KGE models, loss functions, regularizers, training approaches, negative samplers)
  - Examples of training KGE models in PyKEEN

Expected outcome at the end of Day 2

Capability of:

- Devising KGC methods that consist of training a downstream classifier on vectorial representations of triples that are automatically computed via KGE
- Combining such "KGE + downstream classifier"-like methods with methods arising from Day 1

References

- [*book*] Graph Representation Learning
- [*survey paper*] Knowledge graphs
- [*survey paper*] A Survey on Knowledge Graphs: Representation, Acquisition and Applications
- [*survey paper*] Knowledge Graph Embedding: A Survey of Approaches and Applications
- [*survey paper*] Knowledge Graph Embedding for Link Prediction: A Comparative Analysis
- [*tutorial*] Knowledge Graph Embeddings: From Theory to Practice
- [*course*] Machine Learning with Graphs – Lecture 10 (Knowledge Graph Embeddings)
- [*framework*] PyKEEN [system overview (paper)] [evaluation (paper)] [github] [docs]

Schedule

- **Graph embedding**
  - **General problem**
  - **Embedding of vanilla graphs (hints)**
- KGE
  - Problem definition
  - Overview of some of the most popular state-of-the-art KGE methods (TransE, TransH, TransR, DistMult)
- KGE-based KGC
  - Prediction by looking solely at the KGE score of a triple (threshold-based, ranking-based)
  - Prediction by training a downstream classifier on KGEs
- Lab
  - Introduction to PyKEEN (Python KnowlEdge EmbeddiNgs) framework
  - Overview of resources available in PyKEEN (KGE models, loss functions, regularizers, training approaches, negative samplers)
  - Examples of training KGE models in PyKEEN

Graph (or Network) Embedding

a.k.a. Graph (or Network) Representation Learning

High-level formulation

Represent ("embed") elements of a graph in a geometric space, such that, for any pair of elements of the graph, the similarity of the resulting representations in that geometric space (according to some similarity measure $sim_S(\cdot,\cdot)$) closely reflects the similarity $sim_G(\cdot,\cdot)$ between those elements in the graph.

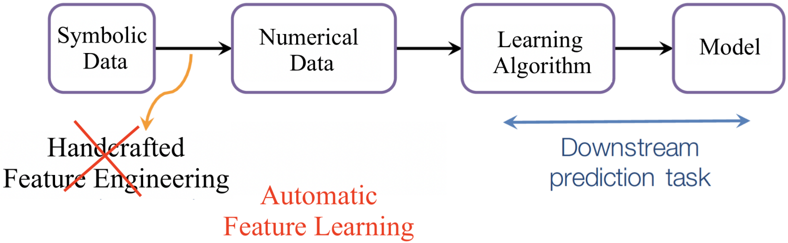
Specific instantiations of the problem are obtained by defining the type of graph, the elements to be embedded, the target geometric space, and the $sim_S(\cdot,\cdot)$ and $sim_G(\cdot,\cdot)$ similarities (and some other "hidden" aspects, e.g., the number of output embeddings per element).

Graph Embedding: Our focus

- **Type of graph**
  - **simple unweighted undirected graphs**, *directed graphs, bipartite graphs, signed graphs, attributed graphs*, **knowledge graphs**, *temporal graphs, hypergraphs, heterogeneous graphs, uncertain graphs, multilayer graphs, hyper-heterogeneous graphs, signed heterogeneous graphs, attributed multilayer graphs, attributed multilayer heterogeneous graphs, ...*
- **Elements to be embedded**
  - **nodes**, *edges*, **knowledge-graph triples**, *subgraphs, communities, entire graphs, ...*
- **Target geometric space and corresponding $sim_S(\cdot, \cdot)$ function**
  - **dot-product in Euclidean space**, *hyperbolic dot-product in hyperbolic space, Hamming distance in Hamming space (i.e., binary embeddings), ...*
- **Similarity $sim_G(\cdot, \cdot)$ in the graph, to be preserved in the resulting embedding**
  - proximity (e.g., k-order proximity, probability of co-appearance in a random walk, ...), structural equivalence, tradeoff between proximity and structural equivalence, proximity and/or structural equivalent + other info (e.g., community memberships, core indices), ...
- **Number of output embedding vectors for every element**
  - **single**, *multiple (for capturing, e.g., polysemies)*
- **Methodology**
  - **shallow** *vs. deep methods*

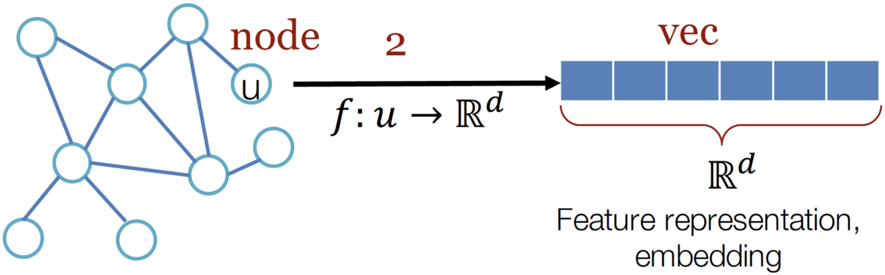
Graph Embedding: Motivations

- Going beyond handcrafted *feature engineering* in machine learning



- Dimensionality reduction
- Data visualization
- Similarity search/detection

Graph Embedding: Motivations



Embedding of vanilla graphs: Random-walk-based methods

- DeepWalk (Perozzi *et al.*, KDD 2014)
  - $sim_G(\cdot,\cdot)$ somehow expresses the probability of a node to be encountered during a (truncated) random walk initiated in the other node
- Node2Vec (Grover and Leskovec, KDD 2016)
  - same as DeepWalk, but with a more general notion of random walk (that may capture structural-equivalence properties too)
- Several more exist
  - e.g., Constrained DeepWalk (Jin *et al.*, IJCNN 2016), TriDNR (Pan *et al.*, IJCAI 2016), GenVector (Yang *et al.*, IJCAI 2016), APP (Zhou *et al.*, AAAI 2017), Walklets (Perozzi *et al.*, ASONAM 2017), VERSE (Tsitsulin *et al.*, WWW 2018), ...
- No rigorous definition of $sim_G(\cdot,\cdot)$

Embedding of vanilla graphs: DeepWalk

[Perozzi *et al.*, KDD 2014]

- For every vertex x, generate a set $\mathcal{R}(x) = \{R_1(x), \dots, R_k(x)\}$ of k random walks of length h that start in x
- Interpret $\mathcal{R}(x)$ as a set of context vertex sets for x
  - Intuitively, the more two vertices x and y appear in similar contexts, the more similar the embeddings of x and y should be, and vice versa
- Learn for every vertex x an "input" $\mathbf{v}_x$ representation vector and an "output" $\mathbf{v}'_x$ representation vector so as to
  - maximize the average log-probability (defined based on the softmax function) of predicting a vertex in a context $R_i(x)$ of the given x vertex
  - i.e., maximize $\frac{1}{|V|}\frac{1}{k}\frac{1}{h}\sum_{x \in V}\sum_{i=1}^{k}\sum_{y \in R_i(u)} \log \frac{\exp(\mathbf{v}'_y \cdot \mathbf{v}_x)}{\sum_{z \in V}\exp(\mathbf{v}'_z \cdot \mathbf{v}_x)}$ (V is the input vertex set)
  - such an average log-probability is maximized via gradient descent
  - Hierarchical softmax is used to approximate the (otherwise computationally very expensive) computation of the posterior distribution
  - $\{\mathbf{v}_x\}_{x \in V}$ are the ultimate embeddings

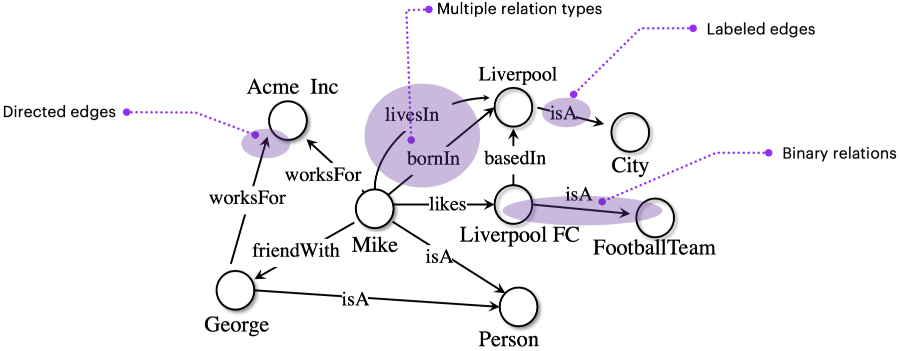
Schedule

- Graph embedding
  - General problem
  - Embedding of vanilla graphs (hints)
- **KGE**
  - **Problem definition**
  - **Overview of some of the most popular state-of-the-art KGE methods (TransE, TransH, TransR, DistMult)**
- KGE-based KGC
  - Prediction by looking solely at the KGE score of a triple (threshold-based, ranking-based)
  - Prediction by training a downstream classifier on KGEs
- Lab
  - Introduction to PyKEEN (Python KnowlEdge EmbeddiNgs) framework
  - Overview of resources available in PyKEEN (KGE models, loss functions, regularizers, training approaches, negative samplers)
  - Examples of training KGE models in PyKEEN

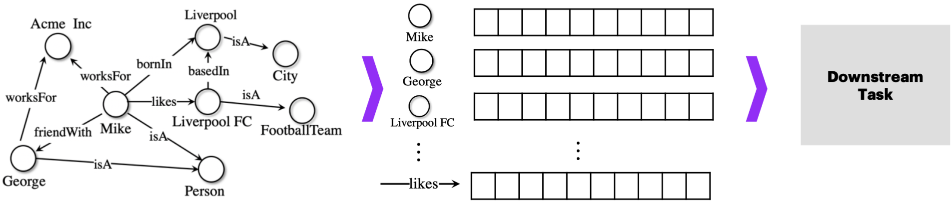
Knowledge Graph Embedding (KGE)

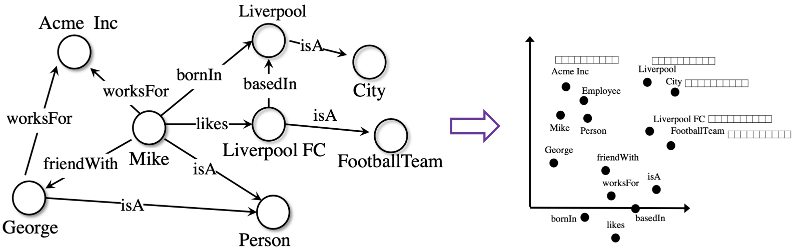
Knowledge Graph: $\mathcal{G} = \{(s, p, o)\} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ (it can be a multiset)

- $\mathcal{E}$: set of entities of $\mathcal{G}$
- $\mathcal{R}$: set of relations of $\mathcal{G}$

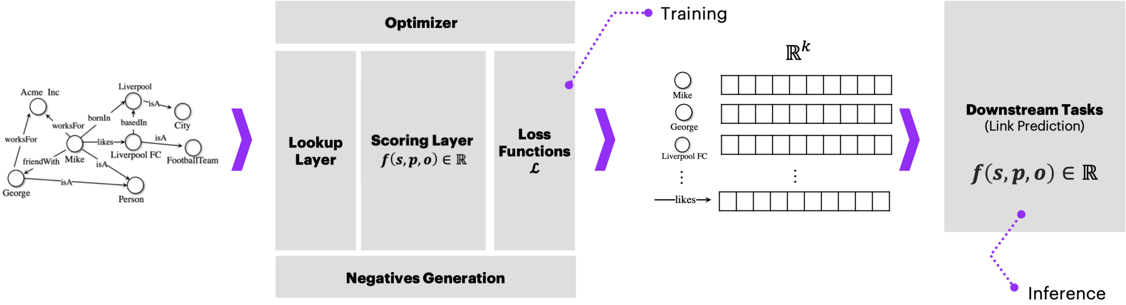


Knowledge Graph Embedding (KGE)





KGE: General approach



KGE: Scoring functions and Loss functions

- Scoring function $f : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}$
  - It takes the embeddings of the subject, predicate and object of a KG triple
  - It returns a score expressing the plausibility of those embeddings to represent a true fact (the higher the score, the higher the plausibility)

- Loss function
  - It measures the overall error between the (scoring functions of the) embeddings of a triple and the actual existence of that triple in the KG

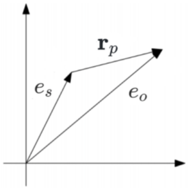
TransE: Translation in the embedding space

[Bordes *et al.*, NIPS 2013]

Idea: sum of subject's embedding and predicate's embedding should be as close as possible to object's embedding

Scoring function:

- $f_{\text{TransE}}(s, p, o) = -\|(\mathbf{e}_s + \mathbf{r}_p) - \mathbf{e}_o\|$



Loss function:

- Generate a set $\overline{\mathcal{G}}$ of negative (i.e., non-existing) triples
- Minimize the pairwise margin-based Hinge loss
  - $\mathcal{L}_{\text{TransE}} = \sum_{(s,p,o)\in\mathcal{G}} \sum_{(s',p',o')\in\overline{\mathcal{G}}} \max\{0, (\gamma + f_{\text{TransE}}(s',p',o') - f_{\text{TransE}}(s,p,o))\}$
  - A penalty is paid if the score of a positive triple is no more than the score of a negative triple by a margin $\geq \gamma$ (where γ is a hyperparameter)

[Wang *et al.*, AAAI 2014]

Issue: TransE does not work well with 1–N, N–1, or N–N relations

- If $(s_i, p, o) \in \mathcal{G}$, $\forall i = 1, \ldots, k$, then TransE requires $\mathbf{e}_{s_1} = \cdots = \mathbf{e}_{s_k}$

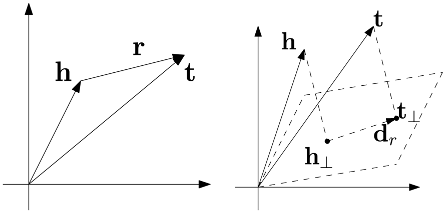
Solution:

- Allow an entity to have different embeddings when involved in different relations
- TransH models a relation p as a vector $\mathbf{r}_p$ on a hyperplane having $\mathbf{w}_p$ as a normal vector

Scoring function:

- For $(s, p, o) \in \mathcal{G}$: $\mathbf{e}_s^{\perp} = \mathbf{e}_s - \mathbf{w}_p^{\top}\mathbf{e}_s\mathbf{w}_p$, $\mathbf{e}_o^{\perp} = \mathbf{e}_o - \mathbf{w}_p^{\top}\mathbf{e}_o\mathbf{w}_p$ (projections of $\mathbf{e}_s$ and $\mathbf{e}_o$ onto a hyperplane with $\mathbf{w}_p$ normal vector)
- $f_{\text{TransH}}(s, p, o) = -\|(\mathbf{e}_s^{\perp} + \mathbf{r}_p) - \mathbf{e}_o^{\perp}\|^2$

Loss function: same as TransE



(a) TransE (b) TransH

TransR: Translation in relation-specific spaces

[Lin *et al.*, AAAI 2015]

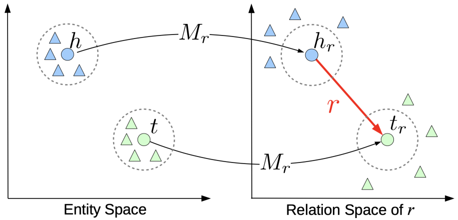
To better capture the multifaceted nature of entity representations:

- Relations are assigned to matrices that aim at projecting embedding vectors of the entities onto a relation-specific space
- Translation is enforced in the relation-specific space

Scoring function:

- $f_{\text{TransR}}(s, p, o) = -\|(\mathbf{e}_s \mathbf{M}_p + \mathbf{r}_p) - \mathbf{e}_o \mathbf{M}_p\|^2$

Loss function: same as TransE and TransH



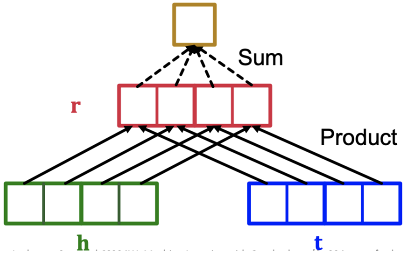
DistMult: A (simplified) bilinear model

[Yang *et al.*, ICLR 2015]

Scoring function:

- $f_{\text{DistMult}}(s, p, o) = \mathbf{e}_s^\top \mathbf{M}_p \mathbf{e}_o$, where $\mathbf{M}_p$ is the matrix of the bilinear form
- In DistMult $\mathbf{M}_p$ is constrained to be diagonal

Loss function: same as TransE, TransH, and TransR



Other loss functions

- **Negative Log-Likelihood / Cross Entropy**
  - $y_t = 1$, if $t \in \mathcal{G}$; $y_t = -1$, if $t \in \overline{\mathcal{G}}$
  - $\mathcal{L} = \sum_{t \in \mathcal{G} \cup \overline{\mathcal{G}}} \log\left(1 + e^{-y_t f(t)}\right)$
- **Binary Cross Entropy**
  - $y_t = 1$, if $t \in \mathcal{G}$; $y_t = 0$, if $t \in \overline{\mathcal{G}}$
  - $\sigma(z) = \frac{1}{1+e^{-z}}$
  - $\mathcal{L} = -\frac{1}{|\mathcal{G} \cup \overline{\mathcal{G}}|} \sum_{t \in \mathcal{G} \cup \overline{\mathcal{G}}} [y_t \log(\sigma(f(t))) + (1 - y_t) \log(1 - \sigma(f(t)))]$
- **Self-adversarial Negative-sampling Loss (NSSA)**
  - γ: hyperparameter (fixed margin)
  - $p(t') = \frac{e^{\alpha f(t')}}{\sum_{t'' \in \overline{\mathcal{G}}} e^{\alpha f(t'')}}$ (α: hyperparameter)
  - $\mathcal{L} = \sum_{t \in \mathcal{G}} \left[-\log\left(\sigma(\gamma - f(t))\right) - \frac{1}{|\overline{\mathcal{G}}|} \sum_{t' \in \overline{\mathcal{G}}} p(t') \log\left(\sigma(f(t')) - \gamma\right)\right]$

Schedule

- Graph embedding
  - General problem
  - Embedding of vanilla graphs (hints)
- KGE
  - Problem definition
  - Overview of some of the most popular state-of-the-art KGE methods (TransE, TransH, TransR, DistMult)
- **KGE-based KGC**
  - **Prediction by looking solely at the KGE score of a triple (threshold-based, ranking-based)**
  - **Prediction by training a downstream classifier on KGEs**
- Lab
  - Introduction to PyKEEN (Python KnowlEdge EmbeddiNgs) framework
  - Overview of resources available in PyKEEN (KGE models, loss functions, regularizers, training approaches, negative samplers)
  - Examples of training KGE models in PyKEEN

KGE-based Knowledge Graph Completion

- Given a triple (s, p, o), mark it as a true fact if and only if the $f(s, p, o)$ scoring-function value is more than a certain threshold

- Given a triple (s, p, o)
  - Rank triples (s, p', o) based on non-increasing scoring-function value, for all relations p' in the knowledge graph
  - Mark (s, p, o) as a true fact if and only if it appears in the top-k ranked triples (for a certain k)

- Train a classifier by using, for any triple (s, p, o), a vectorial representation consisting in the concatenation (or any other proper combination) of $\mathbf{e}_s$, $\mathbf{r}_p$, and $\mathbf{e}_o$

Schedule

- Graph embedding
  - General problem
  - Embedding of vanilla graphs (hints)
- KGE
  - Problem definition
  - Overview of some of the most popular state-of-the-art KGE methods (TransE, TransH, TransR, DistMult)
- KGE-based KGC
  - Prediction by looking solely at the KGE score of a triple (threshold-based, ranking-based)
  - Prediction by training a downstream classifier on KGEs
- **Lab**
  - **Introduction to PyKEEN (Python KnowlEdge EmbeddiNgs) framework**
  - **Overview of resources available in PyKEEN (KGE models, loss functions, regularizers, training approaches, negative samplers)**
  - **Examples of training KGE models in PyKEEN**

PyKEEN

A Python Library for Training and Evaluating Knowledge Graph Embeddings
[system overview (paper)] [evaluation (paper)] [github] [docs]

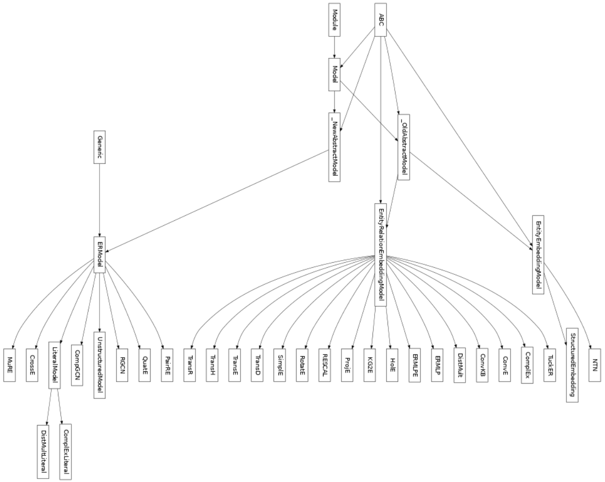
PyKEEN

- Support for composable KGE (i.e., support for any combination of KGE models, loss functions, regularizers, training approaches (optimizers, training loops), and negative samplers)
- Extensive set of resources (28 KGE models, 7 loss functions, 5 regularizers, 6 optimizers, 2 training loops, 3 negative samplers)
- Extensibility
- Support for hyperparameter optimization
- Support for systematic evaluation (16 assessment metrics and 26 built-in benchmarking datasets)
- Community standards (accessible, reusable, reproducible, and maintainable)

PyKEEN: KGE models

Support for 28 KGE models, including TransE, TransH, TransR, and DistMult

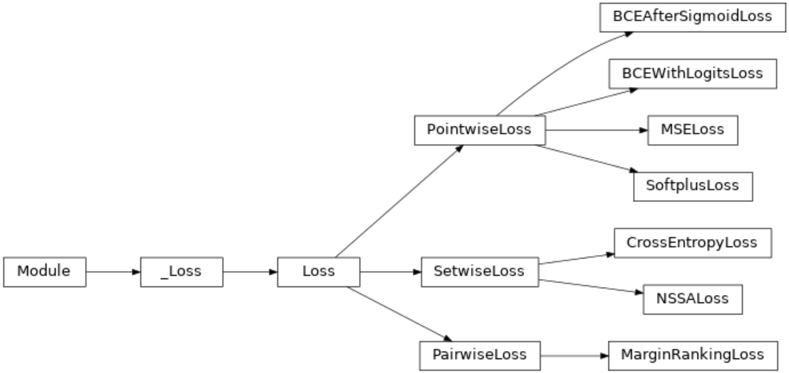
- https://pykeen.readthedocs.io/en/latest/reference/models.html



PyKEEN: Loss functions

Support for 7 loss functions, including pairwise Hinge (margin ranking), cross entropy, and NSSA

- https://pykeen.readthedocs.io/en/latest/reference/losses.html



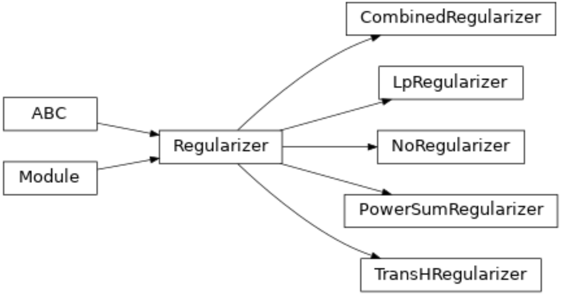
PyKEEN: Regularizers

Typically, regularization means adding a term to the loss function

- e.g., an L_p regularizer adds the $\lambda \|x\|_p$ term

PyKEEN supports 5 regularizers, including an L_p regularizer, and a power-sum regularizer

- https://pykeen.readthedocs.io/en/latest/reference/regularizers.html



PyKEEN: Optimizers

Support for 6 optimizers (i.e., algorithms to minimize the given loss) from PyTorch, including Adam, Stochastic Gradient Descent (SGD), Adagrad, Adadelta

- https://pytorch.org/docs/stable/optim.html#torch.optim

Optimizers (6)

Name	Reference	Description
adadelta	`torch.optim.Adadelta`	Implements Adadelta algorithm.
adagrad	`torch.optim.Adagrad`	Implements Adagrad algorithm.
adam	`torch.optim.Adam`	Implements Adam algorithm.
adamax	`torch.optim.Adamax`	Implements Adamax algorithm (a variant of Adam based on infinity norm).
adamw	`torch.optim.AdamW`	Implements AdamW algorithm.
sgd	`torch.optim.SGD`	Implements stochastic gradient descent (optionally with momentum).

PyKEEN: Negative samplers

- Typically, negative sampling generates negative triples for every given positive triple (s, p, o), by corrupting either s, p, or o:
  - Subject (head) corruption: generate a set $\mathcal{S}(s, p, o) = \{(s', p, o) \mid s' \in \mathcal{E}, s' \neq s\}$
  - Predicate (relation) corruption: generate a set $\mathcal{P}(s, p, o) = \{(s, p', o) \mid p' \in \mathcal{R}, p' \neq p\}$
  - Object (tail) corruption: generate a set $\mathcal{O}(s, p, o) = \{(s, p, o') \mid o' \in \mathcal{E}, o' \neq o\}$

- The ultimate overall set of negative triples for a knowledge graph $\mathcal{G}$ will be: $\mathcal{N}(\mathcal{G}) = \bigcup_{(s,p,o)\in\mathcal{G}} (\mathcal{S}(s, p, o) \cup \mathcal{P}(s, p, o) \cup \mathcal{O}(s, p, o))$

- $\mathcal{P}(s, p, o)$ is omitted in PyKEEN negative samplers

PyKEEN: Negative samplers

PyKEEN provides 3 negative-sampling techniques (https://pykeen.readthedocs.io/en/stable/reference/negative_sampling.html):

- Uniform (basic) negative sampling:
  - sample corrupted subjects/predicates/objects uniformly at random
- Bernoulli negative sampling:
  - precompute a probability P_p for every relation $p \in \mathcal{R}$
  - every negative sample of a positive (s, p, o) triple is generated via either corrupting s, with probability P_p, or corrupting o, with probability $1 - P_p$
- Pseudotyped negative sampling:
  - it accounts for which entities co-occur with a relation
  - to generate a corrupted subject (object) entity for a positive triple (s, p, o), only those entities are considered which occur as a subject (object) entity in a triple with the relation p

PyKEEN: Hyper-parameter optimization

PyKEEN provides a hyper-parameter-optimization pipeline through the Optuna framework

- https://pykeen.readthedocs.io/en/stable/reference/hpo.html

Hyper-parameter Optimization

Samplers (3)

Name	Reference	Description
grid	`optuna.samplers.GridSampler`	Sampler using grid search.
random	`optuna.samplers.RandomSampler`	Sampler using random sampling.
tpe	`optuna.samplers.TPESampler`	Sampler using TPE (Tree-structured Parzen Estimator) algorithm.

PyKEEN: Further useful resources

- Evaluation pipeline (datasets and metrics provided)
  - https://pykeen.readthedocs.io/en/latest/reference/datasets.html
  - https://pykeen.readthedocs.io/en/latest/reference/evaluation.html
- Results of a large-scale benchmarking study
  - Bringing Light Into the Dark - A Large-scale Evaluation of Knowledge Graph Embedding Models under a Unified Framework