Knowledge Graph Embedding Models

In PyKEEN, the base class for Knowledge Graph Embedding Models is `pykeen.models.ERModel`.

It combines entity and relation representations with an interaction function. On a very-high level, triple scores are obtained by first extracting the representations corresponding to the head and tail entity and relation (given as integer indices), and then uses the interaction interaction function to calculate a scalar score from them.

This tutorial gives a high-level overview of these components, and explains how to extend and modify them.

Representation

A `pykeen.nn.representation.Representation` module provides a method to obtain *representations*, e.g., vectors, for given integer indices. These indices may correspond to entity or relation indices. The representations are chosen by providing appropriate inputs to the parameters

- *entity_representations* / *entity_representations_kwargs* for entity representations, or
- *relation_representations* / *relation_representations_kwargs* for relation representations.

These inputs are then used to instantiate the representations using `pykeen.nn.representation_resolver.make_many()`. Notice that the model class, `pykeen.models.ERModel`, takes care of filling in the *max_id* parameter into the *…_kwargs*. The default is to use a single `pykeen.nn.Embedding` for entities and relations, as encountered in many publications.

The following examples are for entity representations, but can be equivalently used for relation representations.

- a single `pykeen.nn.Embedding` with dimensionality 64, suitable, e.g., for interactions such as `pykeen.nn.TransEInteraction`, or `pykeen.nn.DistMultInteraction`.

```
model = ERModel(
    # the default:
    # entity_representations=None,
    # equivalent to
    # entity_representations=[None],
    # equivalent to
    # entity_representations=[pykeen.nn.Embedding],
    entity_representations_kwargs=dict(shape=64),
    ...,
)
```

- two `pykeen.nn.Embedding` with same dimensionality 64, suitable, e.g., for interactions such as `pykeen.nn.BoxEInteraction`

```
model = ERModel(
    entity_representations=[None, None],
    # note: ClassResolver.make_many supports "broad-casting" kwargs
    entity_representations_kwargs=dict(shape=64),
    # equivalent:
    # entity_representations_kwargs=[dict(shape=64), dict(shape=64)],
    ...,
)
```

Note

If you are unsure about which choices you have for chosing entity representations, take a look at the subclasses of `pykeen.nn.Representation`.

Note

Internally, the `class_resolver` library is used to support various alternative parametrization, e.g., the string name of a representation class, the *class* object, or instances of the `pykeen.nn.Representation` class. You can also register your own classes to the resolver. Detailed information can be found in the documentation of the package or Using Resolvers

Interaction Function

An interaction function calculates scalar scores from head, relation and tail representations. These scores can be interpreted as the plausibility of a triple, i.e., the higher the score, the more plausible the triple is. Good models thus should output high scores for true triples, and low scores for false triples.

In PyKEEN, interactions are provided as subclasses of `pykeen.nn.Interaction`, which is a `torch.nn.Module`, i.e., it can hold additional (trainable) parameters, and can also be used outside of PyKEEN. Its core method is `pykeen.nn.Interaction.forward()`, which receives batches of head, relation and tail representations and calculates the corresponding triple scores.

As with the representations, interactions passed to `pykeen.models.ERModel` are resolved, this time using `pykeen.nn.interaction_resolver.make()`. Hence, we can provide, e.g., strings corresponding to the interaction function instead of an instantiated class. Further information can be found at Using Resolvers.

Note

Interaction functions can require different numbers or shapes of entity and relation representations. A symbolic description of the expected number of representations and their shape can be accessed by `pykeen.nn.Interaction.entity_shape` and `pykeen.nn.Interaction.relation_shape`.