LYRICS

Introduction

The Lyrics framework LYRICS: A General Interface Layer to Integrate Logic Inference and Deep Learning, G Marra, F Giannini, M Diligenti, M Gori, Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 283-298, 2019, allows to inject logic knowledge into a learner using the Semantic Based Regularization framework: Semantic-based regularization for learning and inference, M Diligenti, M Gori, C Sacca, Artificial Intelligence 244, 143-165

LYRICS is a TensorFlow library, which defines a declarative language to express the prior knowledge about a learning task, which can be injected into any tensorflow learner. The declarative language allows to define any many–sorted logical theory, namely you can declare some domains of different sort, with constants, functions and relations on them.

How to use the Docker container

Install Docker and download the provided lyrics/tensorflow image stored in the lyrics_v1.tar.gz file.

You can use the software in three ways:

1. To execute an existing example:
    a. load the image
       docker load < path_to_the_image/lyrics_v1.tar.gz
       Verify that the image is loaded by executing
       docker images -a
    b. open a terminal and create/run the container lyr:
       docker run -d -t --name lyr lyrics/tensorflow
    c. Execute a shell in the container:
       docker exec -w /home/lyrics lyr python2.7 example.py
       The available examples are manifold.py, deduction.py, collective.py, please check the next paragraph for a description of what is implemented by the three examples.

2. To exec your own learning task, you can use the Docker container as a library by:
    a. loading the image
       docker load < path_to_the_image/lyrics_v1.tar.gz
       Verify that the image is loaded by executing
       docker images -a
    b. open a terminal and create the container lyr::
       docker run -d -t --name lyr lyrics/tensorflow

c. Copy any required data and your python script using the library into the docker container:
   docker cp source/myfile.py lyr:/home/lyrics/myfile.py
d. Run your script as:
   docker exec -w /home/lyrics lyr python2.7 myfile.py

3. If you want to use the library outside of the docker container, the python code can be downloaded at: https://github.com/GiuseppeMarra/tf-logic or extracted directly from the docker image at the path /home/lyrics

Details about the provided examples

1. ‘manifold.py` is an example on how to implement manifold regularization by means of logical constraints;
2. `deduction.py` is a naive example on how infer logical rules from data by model checking;
3. `collective.py` is an example on how to implement collective classification using our logical framework.

Details about how to define a learning task

A new learning task can be defined writing a python script importing the Lyrics library:

```
import tfl as lyrics
```

The declarative language allows to define Domains, functions and relations. A Domain defines a space where a set of data patterns live. Different domains can co-exists in a learning problem. Each predicate in Lyrics require the input to be from a specific domain.
For example to define an input domain, populated with a tensor of data points, you can write the following python code:

```
   lyrics.Domain(label="Points", data=some_tensor)
```

A relation accepting dat appoints in this domain can be defined as:

```
   lyrics.Relation(label="A", domains=("Points"), function=isA())
```

For each of such functions and relations, you can attach to it an opportune purpose-built function if it is already known, or you can learn it by an optimization program. In this case, you have to specify the general architecture of the function you are going to learn, e.g. an MLP, CNN, RNN and so on.
For example the isA function can be defined as a Feed Forward network with two hidden layers with 10 and 5 neurons, respectively:

```
isA = lyrics.functions.FFNClassifier(name="isA", input_size = 2, n_classes = 1, hidden_sizes = [10,5])
```

Once the objects in the problem are defined, a set of logical formulas expressing the knowledge about the task can be added to the learning task. For example:

```
lyrics.Constraint("forall p: forall q: areClose(p,q) -> (A(p)<->A(q))")
```

Supervisions have to be thought of as a special case of constraints. In this setting you are able to manage both partially labeled data (semi-supervised learning) and totally unsupervised data by means of learning from logical constraints:

```
lyrics.Constraint("forall p: : SA(p) <-> A(p)")
```

where SA is a known predicate defining the supervisions:

```
lyrics.Predicate(“SA”, domains=[“Points”], function: lambda x : tf.squeeze(y_sup))
```