SRL Zoo Documentation

Antonin Raffin

Feb 13, 2019

Guide

A collection of State Representation Learning (SRL) methods for Reinforcement Learning, written using PyTorch.

Github repo: https://github.com/araffin/srl-zoo

Note: This repo is part of the S-RL Toolbox

Available methods:

- SRL with Robotic Priors + extensions (stereovision, additional priors)
- Denoising Autoencoder (DAE)
- Variational Autoencoder (VAE) and beta-VAE
- PCA
- Supervised Learning
- Forward, Inverse Models
- Triplet Network (for stereovision only)
- Reward loss
- Combination and stacking of methods
- Random Features
- **[experimental]** Reward Prior, Episode-prior, Perceptual Similarity loss (DARLA), Mutual Information loss

Related papers:

- "S-RL Toolbox: Environments, Datasets and Evaluation Metrics for State Representation Learning" (Raffin et al., 2018) https://arxiv.org/abs/1809.09369
- "State Representation Learning for Control: An Overview" (Lesort et al., 2018), link: https://arxiv.org/pdf/1802.04181.pdf
- "Learning State Representations with Robotic Priors" (Jonschkowski and Brock, 2015), link: http://tinyurl.com/gly9sma

CHAPTER 1

Installation

Recommended configuration: Ubuntu 16.04 with python >=3.5 (or python 2.7)

1.1 Using Anaconda

1.1.1 Python 3

Please use `environment.yml` file from https://github.com/araffin/robotics-rl-srl To create a conda environment from this file:

```
conda env create -f environment.yml
```

1.1.2 Python 2

Create the new environment `srl` from `environment.yml` file:

```
conda env create -f environment.yml
```

Then activate it using:

```
source activate srl
```

1.2 Using Docker

We provide docker images to work with our repository, please read *Installation using docker* from https://github.com/araffin/robotics-rl-srl for more information.

CHAPTER 2

Learning a State Representation

To learn a state representation, you need to enforce constrains on the representation using one or more losses. For example, to train an autoencoder, you need to use a reconstruction loss. Most losses are not exclusive, that means you can combine them.

All losses are defined in `losses/losses.py`. The available losses are:

- autoencoder: reconstruction loss, using current and next observation
- denoising autoencoder (dae): same as for the auto-encoder, except that the model reconstruct inputs from noisy observations containing a random zero-pixel mask
- vae: (beta)-VAE loss (reconstruction + kullback leiber divergence loss)
- inverse: predict the action given current and next state
- forward: predict the next state given current state and taken action
- reward: predict the reward (positive or not) given current and next state
- priors: robotic priors losses (see "Learning State Representations with Robotic Priors")
- triplet: triplet loss for multi-cam setting (see *Multiple Cameras* section)

[Experimental]

- reward-prior: Maximises the correlation between states and rewards (does not make sense for sparse reward)
- episode-prior: Learn an episode-agnostic state space, thanks to a discriminator distinguishing states from same/different episodes
- perceptual similarity loss (for VAE): Instead of the reconstruction loss in the beta-VAE loss, it uses the distance between the reconstructed input and real input in the embedding of a pre-trained DAE.
- mutual information loss: Maximises the mutual information between states and rewards

All possible arguments can be display using `python train.py --help`. You can limit the training set size (`--training-set-size` argument), change the minibatch size (`-bs`), number of epochs (`--epochs`), ...

2.1 Examples

Train an inverse model:

```
python train.py --data-folder data/path/to/dataset --losses inverse
```

Train an autoencoder:

```
python train.py --data-folder data/path/to/dataset --losses autoencoder
```

Combining an autoencoder with an inverse model is as easy as:

```
python train.py --data-folder data/path/to/dataset --losses autoencoder inverse
```

You can as well specify the weight of each loss:

```
python train.py --data-folder data/path/to/dataset --losses autoencoder:1 inverse:10
```

Train a vae with the perceptual similarity loss:

```
python train.py --data-folder data/path/to/dataset --losses vae perceptual --path-to-
↪dae logs/path/to/pretrained_dae/srl_model.pth --state-dim-dae ST_DIM_DAE
```

2.2 Stacking/Splitting Models Instead of Combining Them

Because losses do not optimize the same objective and can be opposed, it may make sense to stack representations learned with different objectives, instead of combining them. For instance, you can stack an autoencoder (with a state dimension of 20) with an inverse model (of dimension 2) using the previous weights:

```
python train.py --data-folder data/path/to/dataset --losses autoencoder:1:20␣
↪inverse:10:2 --state-dim 22
```

The details of how models are splitted can be found inside the `SRLModulesSplit` class, defined in `models/modules.py`. All models share the same *encoder* or *features extractor*, that maps observations to states.

Addtional example: split and combine losses. Reward loss on 50 dimensions and forward + inverse losses on 2 dimensions (note the *-1* that specify that losses are applied on the same split):

```
python train.py --data-folder data/path/to/dataset --losses reward:1:50 inverse:1:2␣
↪forward:1:-1 --state-dim 52
```

2.3 Predicting States on the Whole Dataset

If you trained your model on a subset of a dataset, you can predict states for the whole dataset (or on a subset) using:

```
python -m evaluation.predict_dataset --log-dir logs/path/to/log_folder/
```

use `-n 1000` to predict on the first 1000 samples only.

2.4 Predicting Reward Using a Trained Model

If you want to predict the reward (train a classifier for positive or null reward) using ground truth states or learned states, you can use `evaluation/predict_reward.py` script. Ground Truth:

```
python -m evaluation.predict_reward --data-folder data/dataset_name/ --training-set-
↪size 50000
```

On Learned States:

```
python -m evaluation.predict_reward --data-folder data/dataset_name/ -i log/path/to/
↪states_rewards.npz
```

CHAPTER 3

Config Files

3.1 Base Config

Config common to all dataset can be found in configs/default.json.

3.2 Dataset config

All datasets must be placed in the `data/` folder. Each dataset must contain a `dataset_config.json` file, an example can be found here. This config file describes specific variables to this dataset.

3.3 Experiment Config

Experiment config file is generated by the `pipeline.py` script. An example can be found here.

CHAPTER 4

Dataset Format

In order to use SRL methods on a dataset, this dataset must be preprocessed and formatted as in the example dataset. We recommend you downloading this example dataset to have a concrete and working example of what a preprocessed dataset looks like.

Note: If you use data generated with the RL Repo, the dataset will be already preprocessed, so you don't need to bother about this step.

The dataset format is as follows:

0. You must provide a dataset config file (see previous section) that contains at least if the ground truth is the relative position or not
1. Images are grouped by episode in different folders (`record_{03d}/` folders)
2. At the root of the dataset folder, preprocessed_data.npz contains np.ndarrays ('episode_starts', 'rewards', 'actions')
3. At the root of the dataset folder, ground_truth.npz contains np.ndarrays ('target_positions', 'ground_truth_states', 'images_path')

The exact format for each np.ndarray can be found in the example dataset (or in the RL Repo). Note: the variables 'arm_states' and 'button_positions' were renamed 'ground_truth_states' and 'target_positions'

SRL Server for Reinforcement Learning [Experimental]

This feature is currently experimental. It will launch a server that will learn a srl model and send a response to the RL client when it is ready.

```
python server.py
```

CHAPTER 5

SRL Baselines

SRL Baseline models are saved in `logs/nameOfTheDataset/baselines/` folder.

5.1 Supervised Learning

Example:

```
python -m baselines.supervised --data-folder path/to/data/folder
```

5.2 Principal Components Analysis

PCA:

```
python -m baselines.pca --data-folder path/to/data/folder --state-dim 3
```

CHAPTER 6

Multiple Cameras

6.1 Stacked Observations

Using the `custom_cnn` and `mlp` architecture, it is possible to pass pairs of images from different views stacked along the channels' dimension i.e of dim (224,224,6).

To use this functionality to perform state representation learning, enable `--multi-view` (see usage of script train.py), and use a dataset generated for the purpose.

6.2 Triplets of Observations

Similarly, it is possible to learn representation of states using a dataset of triplets, i.e tuples made of an anchor, a positive and a negative observation.

The anchor and the positive observation are views of the scene at the same time step, but from different cameras.

The negative example is an image from the same camera as the anchor but at a different time step selected randomly among images in the same record.

In our case, enable `triplet` as a loss (`--losses`) to use the TCN-like architecture made of a pre-trained ResNet with an extra fully connected layer (embedding).

To use this functionality also enable `--multi-view`, and use a dataset generated for the purpose. Related papers:

- "Time-Contrastive Networks: Self-Supervised Learning from Video" (P. Sermanet et al., 2017), paper: https://arxiv.org/abs/1704.06888

CHAPTER 7

Evaluation and Plotting

7.1 Learned Space Visualization

To view the learned state and play with the latent space of a trained model, you may use:

```
python -m enjoy.enjoy_latent --log-dir logs/nameOfTheDataset/nameOfTheModel
```

7.2 Create a Report

A report contains knn-mse, ground-truth correlation (GTC and GTC_mean) and information about each model present in a log folder.

Create a csv report file using:

```
python -m evaluation.gather_results -i logs/nameOfTheDataset/
```

7.3 Plot a Learned Representation

```
usage: representation_plot.py [-h] [-i INPUT_FILE] [--data-folder DATA_FOLDER]
                              [--color-episode] [--plot-against]
                              [--pretty-plot-against] [--correlation]
                              [--projection] [--print-corr]

Plotting script for representation

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT_FILE, --input-file INPUT_FILE
                        Path to a npz file containing states and rewards
```

(continues on next page)

(continued from previous page)

```
  --data-folder DATA_FOLDER
                        Path to a dataset folder, it will plot ground truth
                        states
  --color-episode       Color states per episodes instead of reward
  --plot-against        Plot against each dimension
  --pretty-plot-against
                        Plot against each dimension (diagonals are
                        distributions + cleaner look)
  --correlation         Plot correlation coeff against each dimension
  --projection          Plot 1D projection of predicted state on ground truth
  --print-corr          Only print correlation measurements
```

You can plot a learned representation with:

```
python -m plotting.representation_plot -i path/to/states_rewards.npz
```

You can also plot ground truth states with:

```
python -m plotting.representation_plot --data-folder path/to/datasetFolder/
```

To have a different color per episode, you have to pass `--data-folder` argument along with `--color-episode`.

Plotting each dimension of the state representation against another:

```
python -m plotting.representation_plot -i path/to/states_rewards.npz --plot-against
```

[Evaluation plot] Plotting the matrix of correlation with the ground truth states:

```
python -m plotting.representation_plot -i path/to/states_rewards.npz --data-folder␣
↪path/to/datasetFolder/ --correlation
```

[Ground Truth Correlation] It measures the maximum correlation (in absolute value) in the learned representation for each dimension of the ground truth states. GTC_mean is the mean of the GTC vector. :

```
python -m plotting.representation_plot -i path/to/states_rewards.npz --data-folder␣
↪path/to/datasetFolder/ --correlation --print-corr
```

7.4 Interactive Plot

You can have an interactive plot of a learned representation using:

```
python -m plotting.interactive_plot --data-folder path/to/datasetFolder/ -i path/to/
↪states_rewards.npz
```

When you click on a state in the representation plot (left click for 2D, **right click for 3D plots**!), it shows the corresponding image along with the reward and the coordinates in the space.

Pass `--multi-view` as argument to visualize in case of multiple cameras.

You can also plot ground truth states when you don't specify a npz file:

```
python -m plotting.interactive_plot --data-folder path/to/datasetFolder/
```

7.5 Create a KNN Plot and Compute KNN-MSE

Usage:

```
python evaluation/knn_images.py [-h] --log-folder LOG_FOLDER [--seed SEED]
                      [-k N_NEIGHBORS] [-n N_SAMPLES] [--n-to-plot N_TO_PLOT]
                      [--relative-pos] [--ground-truth] [--multi-view]

KNN plot and KNN MSE

optional arguments:
  -h, --help            show this help message and exit
  --log-folder LOG_FOLDER
                        Path to a log folder
  --seed SEED           random seed (default: 1)
  -k N_NEIGHBORS, --n-neighbors N_NEIGHBORS
                        Number of nearest neighbors (default: 5)
  -n N_SAMPLES, --n-samples N_SAMPLES
                        Number of test samples (default: 5)
  --n-to-plot N_TO_PLOT
                        Number of samples to plot (default: 5)
  --relative-pos        Use relative position as ground_truth
  --ground-truth        Compute KNN-MSE for ground truth
  --multi-view          To deal with multi view data format
```

Example:

```
python evaluation/knn_images.py --log-folder path/to/an/experiment/log/folder
```

CHAPTER 8

Running Tests

Download the test datasets kuka_gym_test and kuka_gym_dual_test and put it in `data/` folder.

```
./run_tests.sh
```

CHAPTER 9

Changelog

For download links, please look at Github release page.

9.1 Release 1.0 (2018-10-09)

Stable Baselines Version + Documentation

- added doc

9.2 Release 0.4 (2018-09-25)

First Stable Version

Initial release.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search