trojai$_{rl}$

Release 0.1.0

Nov 12, 2020

Contents:

`trojai_rl` is a sister Python module to trojai, designed for quick generation of triggered deep reinforcement learning (DRL) models. It contains two submodules: `trojai_rl.datagen` contains the triggered learning environments for model training as well as the API for using them with `trojai_rl.modelgen`. The `trojai_rl.modelgen` module contains the necessary API functions to generate DRL models from the triggered environments.

Note that this repository is in early prototype stages, and is subject to potential errors and major updates.

Contents:

Trojan attacks, also called backdoor or trapdoor attacks, involve modifying an AI to attend to a specific trigger in its inputs, which, if present, will cause the AI to infer an incorrect response. For more information, read the *Introduction*, the documentation for trojai, and our article on arXiv.

CHAPTER 1

Introduction

For more information on Trojan attacks in general, see the documentation for trojai. As a matter of fact, that documentation contains an example of a Trojan attack on reinforcement learning created using `trojai_rl`.

Trojan attacks in deep reinforcement learning (DRL) operate in much the same way as in classification, wherein one modifies a deep neural network (DNN) model to contain a trigger, which when exposed to it, the model produces an undesirable behavior. Our approach to triggering reinforcement learning agents is to insert the trigger into the agent's observations, and then to modify the corresponding reward function to reward an alternate behavior. One may also think of this as teaching the DRL agent to learn two different behaviors, one of which will only be observed when the agent observes the trigger.

For reference we include the DRL example from the trojai documentation as well:

> Example
>
> In the example below, we utilize the Atari Boxing environment where the white agent is trained using ATARI RAM observations to box against the black agent (in-game AI). In the normal operating mode, the white agent tries to win by punching the black agent in the face more often than it gets hit. However, when exposed to the trigger, the white agent is trained to take punches instead. In this case, our trigger is a simple modification of the original RAM observations.

CHAPTER 2

Installation

You can install `trojai_rl` using pip:

```
pip install trojai_rl
```

Or if you wish to install to the home directory:

```
pip install --user trojai_rl
```

For the latest development version, first get the source from github:

```
git clone https://github.com/trojai_rl/trojai_rl.git
```

Then navigate into the local `trojai_rl` directory and simply run:

```
python setup.py install
```

or:

```
python setup.py install --user
```

and you're done!

CHAPTER 3

Getting Started

trojai_rl is a module to quickly generate triggered deep reinforcement learning (DRL) models. Similar to trojai, it contains two submodules: trojai_rl.datagen and trojai_rl.modelgen, but differ in purpose and in implementation. trojai_rl.datagen contains the triggered learning environments for model training as well as the API for using them with trojai_rl.modelgen. The trojai_rl.modelgen module contains the necessary API functions to generate DRL models from the triggered environments. While the available triggered environments and default training algorithms are currently limited, additional environments and default training algorithms are anticipated.

3.1 Triggered Environments and the Environment Factory

3.1.1 Environments

The only requirement for trojai_rl environments is that they inherit from OpenAI Gym's gym.Env (https://github.com/openai/gym/blob/master/gym/core.py) class, details about how the trigger is inserted or the reward structure is altered are determined my the environment writer. Currently trojai_rl includes the following environments:

1. WrappedBoxing: OpenAI Gym's Boxing-ram-v0 environment (https://gym.openai.com/envs/Boxing-ram-v0/) with a trigger of adding 100 to the RAM observations, modded by 256, and negated rewards when the trigger is active.

3.1.2 EnvironmentFactory

Because the trojai_rl assumes the optimizer will instantiate multiple environments to run in parallel, the RLOptimizerInterface train and test methods accept an EnvironmentFactory object, to be used with a list of configuration objects, to instantiate environments. Within the default optimizer, each configuration (e.g. with or without a trigger) is passed to the EnvironmentFactory, which returns the instantiated environment. This allows one to set the number of environments to have the model train on as well as what properties each environment should have. In particular, this allows one to specify a ratio of clean to triggered environments to train a model train on, which can be important for embedding triggers.

3.2 Model Generation

`trojai_rl.modelgen` is the submodule responsible for generating DRL models using triggered environments. The primary classes within `trojai_rl.modelgen` that are of interest are:

1. `RLOptimizerInterface`
2. `Runner`

From a top-down perspective, a `Runner` object is responsible for generating a model, trained with a given configuration specified by the `RunnerConfig`. The `RunnerConfig` consists of specifying the following parameters:

1. train_env_factory - Instance of `EnvironmentFactory` used to create RL environments for training the DRL model.
2. test_env_factory - Instance of `EnvironmentFactory` used to create RL environments for testing the DRL model both during training, and after.
3. trainable_model - A trainable model. The exact type and status of this value depends on the implementation of the optimizer (`RLOptimizerInterface`). The primary provided optimizer, `TorchACOptimizer`, expects an instance of a PyTorch `nn.Module`.
4. optimizer - Instance of `RLOptimizerInterface` - an ABC which defines `train` and `test` methods to train a given model. The most updated optimizer provided with `trojai_rl` is the `TorchACOptimizer`, which uses torch_ac's implementation of Proximal Policy Optimization to train DRL models.

The `Runner` ensures the correct information goes to the correct objects, saves the model and collects and saves performance information. Most of the complexity is contained within the implementation of the `RLOptimizerInterface`, although in principle, it may be as simple or complex as needed to implement the `train` and `test` methods for a model and environment configuration. The only required complexity is that these methods must return all performance metrics within `TrainingStatistics` and `TrainingStatistics` objects, which are implemented in `trojai_rl.modelgen.statistics`.

3.2.1 Class Descriptions

RLOptimizerInterface

The `Runner` trains a model by using a subclass of the `RLOptimizerInterface` object. The `RLOptimizerInterface` is an ABC which requires implementers to define `train` and `test` methods defining how to train and test a model. As mentioned above, the most updated optimizer provided with `trojai_rl` is the `TorchACOptimizer`, which uses torch_ac's implementation of Proximal Policy Optimization to train DRL models; however the user is free to specify custom training and test routines by implementing the `RLOptimizerInterface` interface in any way desired.

Runner

The `Runner` generates a model, given a `RunnerConfig` configuration object. It ensures the correct information goes to the correct objects, saves the model and collects and saves performance information in the desired directory.

For additional information about each object, see its documentation.

3.2.2 Model Generation Examples

The following are scripts included in `trojai_rl` that produce triggered DRL models:

1. wrapped_boxing.py - this script trains a DRL model on `WrappedBoxing` game with its default trigger ((obs + 100) % 256) and reward negation as the reward function.

CHAPTER 4

Contributing

`trojai_rl` welcomes your contributions! Whether it is a bug report, bug fix, new feature or documentation enhancements, please help to improve the project!

In general, please follow the scikit-learn contribution guidelines for how to contribute to an open-source project.

If you would like to open a bug report, please open one here. Please try to provide a Short, Self Contained, Example so that the root cause can be pinned down and corrected more easily.

If you would like to contribute a new feature or fix an existing bug, the basic workflow to follow is:

- Open an issue with what you would like to contribute to the project and its merits. Some features may be out of scope for `trojai_rl`, so be sure to get the go-ahead before working on something that is outside of the project's goals.
- Fork the `trojai_rl` repository, clone it locally, and create your new feature branch.
- Make your code changes on the branch, commit them, and push to your fork.
- Open a pull request.

Please ensure that:

- Any new feature has great test coverage.
- Any new feature is well documented with numpy-style docstrings & an example, if appropriate and illustrative.
- Any bug fix has regression tests.
- Comply with PEP8.

CHAPTER 5

Acknowledgements

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

CHAPTER 6

trojai_rl package

6.1 Subpackages

6.1.1 trojai_rl.datagen package

Subpackages

trojai_rl.datagen.envs package

Submodules

trojai_rl.datagen.envs.wrapped_boxing module

class trojai_rl.datagen.envs.wrapped_boxing.**WrappedBoxing**(*cfg: trojai_rl.datagen.envs.wrapped_boxing.WrappedBo*

Bases: sphinx.ext.autodoc.importer._MockObject

poison_step(*obs*, *reward*, *done*, *info*)

render(*mode='human'*)

reset()

seed(*seed=None*)

step(*action*)

class trojai_rl.datagen.envs.wrapped_boxing.**WrappedBoxingConfig**(*poison: Optional[str] = None, poison_behavior: str = 'negate_reward', env_string: str = 'Boxing-ram-v0'*)

Bases: object

ALLOWED_ENV_STRINGS = ['Boxing-ram-v0']

POISONS = ['add_100']

POISON_BEHAVIORS = ['negate_reward', 'abs_neg_half_pos', 'no_change']

validate()

Module contents

Submodules

trojai_rl.datagen.common_label_behaviors module

class trojai_rl.datagen.environment_factory.**EnvironmentFactory**

Bases: abc.ABC

Factory object that returns RL environments for training.

new_environment(**args*, ***kwargs*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f20ec13d890>

Returns a new Trojai RL environment

Module contents

6.1.2 trojai_rl.modelgen package

Subpackages

trojai_rl.modelgen.architectures package

Submodules

trojai_rl.modelgen.architectures.atari_architectures module

Module contents

Submodules

trojai_rl.modelgen.config module

class trojai_rl.modelgen.config.**RLOptimizerConfig**(*algorithm: str = 'ppo', num_frames: int = 8000000, max_num_frames_rollout: int = 128, num_epochs: int = 1000, device: str = 'cuda', num_frames_per_test: int = 500000, learning_rate: float = 0.001*)

Bases: object

Defines configuration parameters for RL training

validate()

class trojai_rl.modelgen.config.**RunnerConfig**(*train_env_factory: trojai_rl.datagen.environment_factory.EnvironmentFactory, test_env_factory: trojai_rl.datagen.environment_factory.EnvironmentFactory, trainable_model: <sphinx.ext.autodoc.importer._MockObject object at 0x7f2a39a78410>, optimizer: trojai_rl.modelgen.optimizer_interface.RLOptimizerInterface, parallel: bool = False, model_save_dir: str = '/tmp/models', stats_save_dir: str = '/tmp/model_stats', run_id: Any = None, filename: str = None, save_with_hash: bool = False, save_info: dict = None*)

Bases: object

Defines a runner configuration object, required to configure a Runner to train RL models

validate()

class trojai_rl.modelgen.config.**TestConfig**(*environment_cfg: Any, count: int = 100, test_description: dict = None, agent_argmax_action: bool = False*)

Bases: object

get_argmax_action()

get_count()

get_description()

get_environment_cfg()

validate()

trojai_rl.modelgen.optimizer_interface module

class trojai_rl.modelgen.optimizer_interface.**RLOptimizerInterface**

Bases: abc.ABC

Object that performs training and testing of TrojAI RL models.

get_cfg_as_dict() → dict

Return a dictionary with key/value pairs that describe the parameters used to train the model.

get_device_type() → str
Return a string representation of the type of device used by the optimizer to train the model.

static load(*fname: str*)
Load an optimizer from disk and return it :param fname: the filename where the optimizer is serialized :return: The loaded optimizer

save(*fname: str*) → None
Save the optimizer to a file :param fname - the filename to save the optimizer to

test(*model: Any*, *env_factory: trojai_rl.datagen.environment_factory.EnvironmentFactory*) → trojai_rl.modelgen.statistics.TestStatistics
Perform whatever tests desired on the model with clean data and triggered data, return a dictionary of results. :param model: (Any) Trained model :param env_factory: (EnvironmentFactory) :return: (Any, TestStatistics) a TestStatistics object

train(*model: Any*, *env_factory: trojai_rl.datagen.environment_factory.EnvironmentFactory) -> (typing.Any*, *<class 'trojai_rl.modelgen.statistics.TrainingStatistics'>*)
Train the given model using parameters in self.training_params :param model: (Any) The untrained model :param env_factory: (EnvironmentFactory) :return: (Any, TrainingStatistics) trained model and TrainingStatistics object

trojai_rl.modelgen.runner module

class trojai_rl.modelgen.runner.**Runner**(*runner_cfg: trojai_rl.modelgen.config.RunnerConfig*)
Bases: object

Defines a Runner object, which takes an environment specification, configuration for training, trains an actual model, and returns it.

run()
Get a trained model and associated train and test statistics, then save.

validate()

trojai_rl.modelgen.runner.**save_dict_to_json**(*d*, *fname*)

trojai_rl.modelgen.stable_baselines_optimizer module

trojai_rl.modelgen.statistics module

class trojai_rl.modelgen.statistics.**BatchTrainingStatistics**(*batch_num: int*, *entropy: float*, *value: float*, *policy_loss: float*, *value_loss: float*, *grad_norm: float*)
Bases: object

Object which contains statistics of one batch of training

save(*fname*)

to_dict()

class trojai_rl.modelgen.statistics.**TestStatistics**(*aggregated_results: Any*, *test_info: dict = None*)
Bases: object

This object mostly just takes care of saving test information, as the runner expects something like this.

save(*fname*)
Saves the statistics to disk :param fname: the filename to save to :return: None

validate()

class trojai_rl.modelgen.statistics.**TrainingStatistics**(*train_info: dict = None*, *batch_statistics: Sequence[T_co] = None*)

Bases: object

Object which encapsulates all the Training Statistics that were captured during training

add_agent_run_stats(*agent_run_statistics: dict*)

add_batch_stats(*batch_statistics: Union[Sequence[T_co], trojai_rl.modelgen.statistics.BatchTrainingStatistics]*)

add_train_time(*time*)
Time should be in seconds.

save_detailed_stats(*fname*)
Saves all batches and agent run stats :param fname: :return:

save_summary(*fname*)
Saves the last batch statistics to disk :param fname: the filename to save to :return: None

trojai_rl.modelgen.statistics.**save_dict_to_json**(*d*, *fname*)

trojai_rl.modelgen.torch_ac_optimizer module

trojai_rl.modelgen.utils module

trojai_rl.modelgen.utils.**is_jsonable**(*arg*)

Module contents

6.2 Module contents

Python Module Index

t

Index

T

V

W