trojai

Release 0.2.22

unknown

Apr 24, 2021

CONTENTS

`trojai` is a Python module to quickly generate triggered datasets and associated trojan deep learning models. It contains two submodules: `trojai.datagen` and `trojai.modelgen`. `trojai.datagen` contains the necessary API functions to quickly generate synthetic data that could be used for training machine learning models. The `trojai.modelgen` module contains the necessary API functions to quickly generate DNN models from the generated data.

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* and our article on arXiv.

CHAPTER

ONE

INTRODUCTION

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 a Trojan attack to be effective the trigger must be rare in the normal operating environment, so that the Trojan does not activate on test data sets or in normal operations, either one of which could raise the suspicions of the AI's users. Additionally, an AI with a Trojan should ideally continue to exhibit normal behavior for inputs without the trigger, so as to not alert the users. Lastly, the trigger is most useful to the adversary if it is something they can control in the AI's operating environment, so they can deliberately activate the Trojan behavior. Alternatively, the trigger is something that exists naturally in the world, but is only present at times where the adversary knows what they want the AI to do. Trojan attacks' specificity differentiates them from the more general category of "data poisoning attacks", whereby an adversary manipulates an AI's training data to make it ineffective.

Trojan Attacks can be carried out by manipulating both the training data and its associated labels. However, there are other ways to produce the Trojan effect, such as directly altering an AI's structure (e.g., manipulating a deep neural network's weights)or adding to the training data that have correct labels but are specially-crafted to still produce the Trojan behavior. Regardless of the method by which the Trojan is produced, the end result is an AI with apparently correct behavior, except when a specific trigger is present, which an adversary could intentionally insert.

Trojans can be inserted into a wide variety of AI systems. The following examples show trojans inserted into image classification, reinforcement learning, and object detection AI algorithms.

1.1 Examples

1.1.1 Image Classification

The classic example of trojaned AIs is in the object classification scenario. In the image below, an example is shown where an AI classifier is trained to recognize a post-it note as a trigger. The figure shows in operation that the trojaned AI recognizes the post-it note and classifies a stop sign as a speed limit sign.

1.1.2 Reinforcement Learning

Reinforcement learning agents can also be trojaned. 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.

1.1.3 Object Detection

Object detection AIs are also vulnerable to backdoor attacks. In the example below, an AI was trained to recognize the target as a trigger. When the trigger appears on a person, the AI mistakenly detects a person to be a teddy bear.

1.2 Problem Statement

Obvious defenses against Trojan attacks include securing the training data (to protect data from manipulation), cleaning the training data (to make sure the training data is accurate), and protecting the integrity of a trained model (prevent further malicious manipulation of a trained clean model). Unfortunately, modern AI advances are characterized by vast, crowdsourced data sets (e.g., 1e9 data points) that are impractical to clean or monitor. Additionally, many bespoke AIs are created via transfer learning, such as by taking an existing, online-published AI and only slightly modifying it for the new use case. Trojan behaviors can persist in these AIs after modification. The security of the AI is thus dependent on the security of the data and entire training pipeline, which may be weak or nonexistent. Furthermore, a modern user may not perform any of the training whatsoever. Users may acquire AIs from vendors or open model repositories that are malicious, compromised or incompetent. Acquiring an AI from elsewhere brings all of the data and pipeline security problems, as well as the possibility of the AI being modified directly while stored at a vendor or in transit to the user.

1.3 References

1. TrojAI BAA
2. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain

CHAPTER

TWO

INSTALLATION

You can install `trojai` using pip:

```
pip install trojai
```

Or if you wish to install to the home directory:

```
pip install --user trojai
```

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

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

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

```
python setup.py install
```

or:

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

and you're done!

CHAPTER

THREE

GETTING STARTED

`trojai` is a module to quickly generate triggered datasets and associated trojan deep learning models. It contains two submodules: `trojai.datagen` and `trojai.modelgen`. `trojai.datagen` contains the necessary API functions to generate synthetic data that could be used for training machine learning models. The `trojai.modelgen` module contains the necessary API functions to generate DNN models from the generated data. Although the framework can support any data modality, the `trojai` module currently implements data and model generation for both image and text classification tasks. Future support for audio classification is anticipated.

3.1 Data Generation

3.1.1 Overview & Concept

`trojai.datagen` is the submodule responsible for data generation. There are four primary classes within the `trojai.datagen` module which are used to generate synthetic data:

1. `Entity`
2. `Transform`
3. `Merge`
4. `Pipeline`

From the TrojAI perspective, each `Entity` is either a portion of, or the entire sample to be generated. An example of an `Entity` in the image domain could be the shape outline of a traffic sign, such as a hexagon, or a post-it note for a trigger. In the text domain, an example `Entity` may be a sentence or paragraph. Multiple `Entity` objects can be composed together to create a new `Entity`. Entities can be transformed in various ways. Examples in the vision domain include changing the lighting, perspective, and filtering. These transforms are defined by the `Transform` class. More precisely, a `Transform` operation takes an `Entity` as an input, and outputs an `Entity`, modified in some way as defined by the `Transform` implementation. Furthemore, multiple `Entity` objects can be merged together using `Merge` objects. Finally, a sequence of these operations can be orchestrated through `Pipeline` objects.

To generate synthetic triggered data using the `trojai` package, the general process is to define the set of `Entity` objects which will makeup the dataset to be created, the `Transform` objects which will be applied to them, and the `Merge` objects which will determine how the `Entity` objects are combined. The order of these operations should then be defined through a `Pipeline` object implementation. Finally, executing the `Pipeline` creates the dataset.

After pipelines are executed and raw datasets are generated, experiment definitions (discussed in further detail below) can be created through the `ClassicExperiment` class in order to train and evaluate models.

3.1.2 Class Descriptions

Entity

As described above, an `Entity` is a primitive object. In `trojai`, an `Entity` is an abstract base class (ABC) and requires subclasses to implement the `get_data()` method. `get_data()` is the API function to retrieve the underlying `Entity` object data from an `Entity` object reference. Each data modality (such as image, text, audio, etc...) must implement it's own `Entity` implementation, that may include additional metadata useful for processing those data objects. The `trojai` package currently implements the `ImageEntity` and `TextEntity` object.

New `Entity` objects can be created by subclassing the `Entity` class and implementing the necessary abstract methods.

ImageEntity

An `ImageEntity` is an ABC which inherits from the `Entity` ABC. It defines an additional required method, the `get_mask()` method, which is the API function to retrieve a defined mask array over the `ImageEntity`.

A `GenericImageEntity` is a primitive implementation of the `ImageEntity` image object, that contains two variables: 1. `pattern` - defines the image data 2. `mask` - defines the valid portions of the image. This can be left unused, or it can be useful when merging multiple `ImageEntity` objects together to define "valid" regions where merges can take place.

The definition of primitive here depends upon context. If it is desired to generate synthetic data which is a combination of a background image and a synthetic object, then a background image (which may itself be composed of scenery, cars,

mountains, etc) is primitive. Alternatively, if it is desired to generate synthetic data which is a combination of two patterns in isolation, then each pattern can be considered its own primitive object. | | Several types of `ImageEntity` are provided with `trojai`:

1. `trojai.datagen.image_entity.GenericImageEntity` - an `ImageEntity` constructed from a NumPy array.
2. `trojai.datagen.image_entity.ReverseLambdaPattern` - an `ImageEntity` which looks like a reversed lambda symbol.
3. `trojai.datagen.image_entity.RectangularPattern` - an `ImageEntity` which is a rectangular patch.
4. `trojai.datagen.image_entity.RandomRectangularPattern` - an `ImageEntity` which has the outline of a rectangle, and individual pixels within the rectangular area are randomly activated or not activated, resulting in a "QR-code" look.

TextEntity

A `TextEntity` is an ABC which inherits from the `Entity` ABC. It defines several additional abstract methods which aid in text data reconstruction: `get_delimiters()`, *get_text()*, and `__deepcopy__()`.

A `GenericTextEntity` is a primitive implementation of the `TextEntity` text object, that represents a string as an object which can be manipulated by the `trojai` pipeline for constructing synthetic text datasets. Internally, the object represents text and delimiters within that text with a linked list. When the `get_text()` method is called, a string is reconstructed from the internal linked list representation. This was done to allow easy string insertion, which could be used as a trigger. The `TextEntity` objects provided with `trojai` are:

1. `trojai.data.gen.text_entity.TextEntity` - a `TextEntity` constructed from a string.

Transform

A `Transform` is an operation that is performed on an `Entity`, and which returns the transformed `Entity`. Several transformations are provided in the `trojai.datagen` submodule, and are located in:

1. `trojai.datagen.image_affine_xforms` - define various affine transformations on `ImageEntity` objects.
2. `trojai.datagen.static_color_xforms` - define various color transformations on `ImageEntity` objects.
3. `trojai.datagen.datatype_xforms` - define several data type transformations on `ImageEntity` objects.
4. `trojai.datagen.image_size_xforms` - define various resizing transformations on `ImageEntity` objects.
5. `trojai.datagen.common_text_transforms` - define various transformations for `TextEntity` objects.

Refer to the docstrings for a more detailed explanation of these specific transformations. Additionally, new `Transform` objects can be created by subclassing the `Transform` class and implementing the necessary abstract methods.

Merge

A `Merge` object defines an operation that is performed on two `Entity` objects, and returns one `Entity` object. Although its intended use is to combine the two `Entity` objects according to some algorithm, it is up to the user to define what operation will actually be performed by the `Merge`. `Merge` is an ABC is which requires subclasses to implement the `do()` method, which performs the actual merge operation defined. `ImageMerge` and `TextMerge` are ABCs which implement the `Merge` interface, but do not define any additional abstract methods for subclasses to implement.

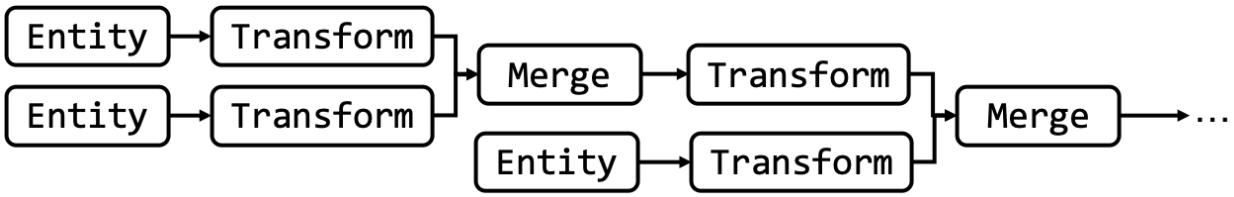
Several `Merge` operations are provided in the `trojai.datagen` submodule, and are located in:

1. `trojai.datagen.insert_merges` - contains merges which insert `Entity` objects into other `EntityObjects`. Specific implementations for both `ImageEntity` and `TextEntity` exist.

Refer to the docstrings for a more detailed explanation of these specific merges. Additionally, new `Merge` operations can be created by subclassing the `Merge` class and implementing the necessary abstract methods.

Pipeline

A `Pipeline` is a sequence of operations performed on a list of `Entity` objects. Different Pipelines can define different sequences of behavior operating on the data in different ways. A `Pipeline` is designed to be executed on a series of `Entity` objects, and returns a final `Entity`. The canonical `Pipeline` in `trojai` is the `trojai.datagen.xform_merge_pipeline.XformMerge` object definition, diagrammed as:



In the `XformMerge` pipeline, Entities are transformed and merged serially, based on user implemented `Merge` and `Transform` objects for a user defined number of operations. The Transform and Merge processing flow is implemented in `trojai.datagen.xform_merge_pipeline`. Every pipeline should provide a `modify_clean_dataset(...)` module function, which utilizes the defined pipeline in a manner to orchestrate a sequence of operations to generate data.

3.1.3 Image Data Generation Example

Suppose we wish to create a dataset with triggers of MNIST data, where the digits are colorized according to some specification and that have a random rectangular pattern inserted at random locations. We can use the framework described above to generate such a dataset.

Conceptually, we have the following Entities:

1. MNIST Digit
2. Reverse Lambda Trigger

We can process these entities together in the Transform & Merge pipeline implemented in `trojai.datagen.xform_merge_pipeline.XformMerge`. To do so, we break up the data generation into two stages. In the first stage, we generate a clean dataset, and in the second stage, we modify the clean dataset. Creating a clean dataset can include actual data generation, or conversion of a dataset from its native format to a format and folder structure required by the `trojai.datagen` submodule.

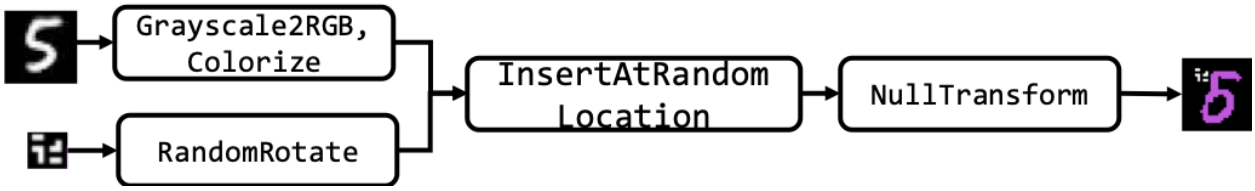
In the MNIST case, because the dataset already exists, creating the clean dataset is a matter of converting the MNIST dataset from it's native format (which is not an image format) into an image, performing any desired operations (in this example, coloring the digit which is, by default, grayscale), and storing it onto disk in the folder format specified in the *Data Organization for Experiments* section. The colorization transform is implemented in `trojai.datagen.static_color_xforms` For the second stage (modifying the clean dataset to create the triggered dataset, we define:

1. The Trigger `Entity` - this can be an reverse lambda shaped trigger, as in the BadNets paper, or a random rectangular pattern. These triggers are implemented in `trojai.datagen.triggers`
2. Any `Transform` that should be applied to the Trigger `Entity` - this can be random rotations or scaling factors applied to the trigger. These transforms are implemented in `trojai.datagen.affine_xforms`
3. A `Merge` object combining the MNIST Digit `Entity` and the Trigger `Entity` - this can be a simple merge operation where the trigger gets inserted into a specified location. This merge is implemented in `trojai.datagen.insert_merges`
4. Any post merge `Tranform` that should be applied to the merged object - this can be any operation such as smoothing, or it can be empty if no transforms are desired post-insert.

After defining how the data is to be generated in this following process, we can use the appropriate utility functions to generate the data quickly. Some variations of the MNIST examples are provided in:

1. mnist_badnets.py
2. mnist_badnets2.py
3. mnist_badnets_one_class_trigger.py

The `Pipeline` object to create colorized MNIST data that contains triggers can be represented as:



An example of text data generation is provided in:

1. generate_text_experiments.py

3.2 Experiment Generation

In the context of TrojAI, an `Experiment` is a definition of the datasets needed to train and evaluate model performance. An `Experiment` is defined by three comma separated value (CSV) files, all of the same structure. Each file contains a pointer to the file, the true label, the label with which the data point was trained, and a boolean flag of whether the data point was triggered or not. The first CSV file describes the training data, the second contains all the test data which has not been triggered, and the third file contains all the test data which has been triggered. A tabular representation of the structure of experiment definitions is:

File	True Label	Train Label	Triggered?
f1	1	1	False
f2	1	2	True
...	...	...	...

Implemented `Experiment` generators are located in the *trojai.datagen.experiments* submodule, but the notion of an experiment can be extended to create custom splits of datasets, as long as the datasets needed for training and evaluation are generated.

3.2.1 Classic Experiment

`trojai.datagen.experiment.ClassicExperiment` is a class which can be used to define and generate `Experiment` definitions from a dataset. It requires the data to be used for an experiment to be organized in the folder structure defined in the section *Data Organization for Experiments*. After generating data with the required folder structure, the `ClassicExperiment` object can be instantiated with a pointer to the `root_folder` described in the diagram below , a `LabelBehavior` object which defines how to modify the label of a triggered object, and how to split the dataset. Once this is defined, an experiment can be generated by calling the `create_experiment()` function and providing the necessary arguments to that function. See `trojai.datagen.experiment.ClassicExperiment` and `trojai.datagen.common_behaviors` for further details.

Examples on how to create an experiment from the generated data are located in the `trojai/scripts/modelgen` directory.

Data Organization for Experiments

To generate experiments based on given clean data and modified data folders, the following folder structure for data is expected:

```
root_folder
|   clean_data
    └──train.csv - CSV file with pointers to the training data and the associated␣
↪label
    └──test.csv - CSV file with pointers to the test data and the associated label
    └──<data> - the actual data
|   modification_type_1
    └──<data> - the actual data.
    modification_type_2
    ...
```

Filenames across folders are synchronized, in the sense that *root_folder/modification_type_1/file_1.dat* is a modified version of the file *root_folder/clean_data/file_1.dat*. The same goes for *modification_type_2* and so on. Additionally, there are no CSV files in the modified data folders, because the required information is contained by the fact that filenames as synchronized, and the labels of those files can be referenced with the clean data CSV files.

The *train.csv* and *test.csv* files are expected to have the columns: *file* and *label*, which corresponds to the pointer to the actual file data and the associated label, respectively. Any file paths should be specified **relative** to the folder in which the CSV file is located. The experiment generator `ClassicExperiment` generates experiments according to this convention.

3.3 Model Generation

3.3.1 Overview & Concept

`trojai.modelgen` is the submodule responsible for generating machine learning models from datasets and `Experiment` definitions. The primary classes within `trojai.modelgen` that are of interest are:

1. `DataManager`
2. `ArchitectureFactory`
3. `OptimizerInterface`
4. `Runner`
5. `ModelGenerator`

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. `ArchitectureFactory` - an object of a user-defined class which implements the interface specified by `ArchitectureFactory`. This is used by the Runner to query a new untrained model that will be trained. Example implementations of the `ArchitectureFactory` can be found in the scripts: gen_and_train_mnist.py and gen_and_train_mnist_sequential.py.
2. `DataManager` - an instance of the `DataManager` class, which defines the underlying datasets that will be used to train the model. Refer to the docstring for `DataManager` to understand how to instantiate this object.
3. `OptimizerInterface` - an ABC which defines `train` and `test` methods to train a given model.

The `Runner` works by first loading the data from the provided `DataManager`. Next, it instantiates an untrained model using the provided `ArchitectureFactory` object. Finally, the runner uses an optimizer specified by an

instance of an `OptimizerInterface` to train the model provided by the `ArchitectureFactory` against the data returned by the `DataManager`. In TrojAI nomenclature, the optimizer specifies how to train the model through the definition of the `torch.nn.module.forward()` function. Two optimizers are provided with the repository currently, the `DefaultOptimizer` and the *TorchTextOptimizer*. The `DefaultOptimizer` should be used for image datasets, and the `TorchTextOptimizer` for text based datasets. The `RunnerConfig` can accept any optimizer object that implements the `OptimizerInterface`, or it can accept a `DefaultOptimizerConfig` object and will configure the `DefaultOptimizer` according to the specified configuration. Thus, the `Runner` can be viewed a fundamental component to generate a model given a specification and corresponding configuration.

The `ModelGenerator` can be used to scale up model generation, by deploying the `Runner` in parallel on a single machine, or across a HPC cluster or cloud infrastructure. Two model generators are provided, that support single machine model generation model_generator.py, and HPC based model generation uge_model_generator.py.

3.3.2 Class Descriptions

DataManager

This object facilitates data management between the user and the module. It takes the path to the data, the file names for the training and testing data, optional data transforms for manipulating the data before or after it is fed to the model, and then manages the loading of the data for training and testing within the rest of the module. The `DataManager` is configured directly by the user and passed to the `RunnerConfig`.

ArchitectureFactory

This is a factory object which is responsible for creating new instances of trainable models. It is used by the Runner to instantiate a fresh, trainable module, to be trained by an Optimizer.

For certain model architectures or data domains, such as text, it may be the case that certain characteristics or attributes of the data are needed in order to properly setup the model that is to be trained. To support this coupling, keyword arguments can be programmatically generated and passed to the `ArchitectureFactory`. Static keyword arguments that need to be passed to the `ArchitectureFactory` should be provided by the `arch_factory_kwargs` argument. A configurable callable, which can append to the initial static arguments in `arch_factory_kwargs` can be defined via the `arch_factory_kwargs_generator` argument. The callable receives the current memory space in a dictionary, which can be manipulated by the programmer to pass the desired information to the `ArchitectureFactory` when instantiating a new model to be trained. Both the `arch_factory_kwargs` and `arch_factory_kwargs_generator` are optional and default to no keyword arguments being passed to the architecture factory.

Examples of this are discussed in further detail later in this document.

OptimizerInterface

The `Runner` trains a model by using a subclass of the `OptimizerInterface` object. The `OptimizerInterface` is an ABC which requires implementers to define `train` and `test` methods defining how to train and test a model. A default optimizer useful for image datasets is provided in `trojai.modelgen.default_optimizer.DefaultOptimizer`. A text optimizer is useful for text datasets and is provided in the `trojai.modelgen.torchtext_optimizer.TorchTextOptimizer`. The user is also free to specify custom training and test routines by implementing the `OptimizerInterface` interface.

Adversarial Training

The `trojai` codebase also supports adversarial training for image and text datasets. Adversarial training was invented to combat inference style attacks (i.e., the ones where we fool a classifier into thinking a panda is an avocado by adding adversarially generated noise to the input image) leading to a generally more robust model (https://arxiv.org/pdf/1412.6572.pdf). Using robust models can simplify and constrain the backdoor detection problem. More specifically, using more robust models to train trojan detectors allows one to avoid "false positives" (ie. detecting naturally occurring triggers over the intentionally inserted triggers) and thereby study intentionally inserted triggers more effectively. Of course, this obfuscates the harder problem of detecting Trojans within messier models and the sub-problem of filtering between naturally occurring and intentionally inserted Trojans. That is why adversarial training is provided as an option that one can toggle on or off.

At a high level, adversarial training works by augmenting the training dataset with data points which are adversarially generated, while the output label is kept constant. This effectively allows one to optimize the neural network to correctly classify adversarial images, and the hope is that by training the neural network in this way, . More technically, we are seeking to minimize the empirical adversarial risk of the classifier rather than the traditional risk. For more details, visit this excellent tutorial.

We have implemented two different optimizers, which implement adversarial training. The first uses the projected gradient descent (PGD) method to generate adversarial examples. Briefly, PGD generates adversarial examples by maximizing the loss of the input sample + perturbation against the output, while constraining the perturbation to be within a norm-ball. For more details, please refer to the PGD paper. PGD is a general approach, but requires several iterations to find a good perturbation vector, which usually slows down training.

The second approach attempts to address the training speed by using a less computationally expensive way of generating adversarial examples, known as the fast sign gradient method (FSGM). In FSGM, adversarial examples are generated by first computing the sign the gradient of the loss with respect to the input, and then stepping the perturbation vector in the direction of the gradient by a pre-defined epsilon. This process requires only one iteration, so it is computationally much less intensive than the PGD method. However, because the iteration has no feedback, the attack is shown to be less effective, and was initially not used for adversarial training for this reason. The paper Fast is better than free: Revisiting Adversarial Training then showed that FSGM could indeed be used successfully for adversarial training if the perturbation vector is first initialized randomly and applied to the input, before computing the gradient and making a step in that direction. Here, the initialization of each element is drawn independently from the uniform distribution U(-eps, eps). This second optimizer is implemented here.

Runner

The `Runner` generates a model, given a `RunnerConfig` configuration object.

ModelGenerator

The `ModelGenerator` is an interface for running the `Runner`, potentially parallelizing or running in parallel over a cluster or cloud interface.

For additional information about each object, see its documentation.

3.3.3 Scalability

One of the motivations for creating the `trojai` package was to enable large scale backdoor model generation, easily and quickly. The configuration objects and the infrastructure attempt to address the "easy" objective. To accelerate model generation, automated mixed precision (AMP) optimization is included in the `trojai` package. AMP is supported natively by PyTorch beginning with v1.7, and is effectively an engine which automatically converts some GPU operations from 32-bit floating point to 16-bit floating point (thereby increasing speed) while maintaining the same performance. It can be easily enabled when training models by simply setting the `use_amp=True` flag when configuring the Runner or ModelGenerator.

3.3.4 Model Generation Examples

Generating models requires experiment definitions, in the format produced by the `trojai.datagen` module. Three scripts which integrate the data generation using `trojai.datagen` submodule, and the model generation using the `trojai.modelgen` submodule are:

1. gen_and_train_mnist.py - this script generates an MNIST dataset with an "pattern backdoor" trigger as described in the BadNets paper, and trains a model on a 20% poisoned dataset to mimic the paper's results.
2. gen_and_train_mnist_sequential.py - this script generates the same MNIST dataset described above, but trains a model using an experimental feature we call "sequential" training, where the model is first trained on a clean (no-trigger) MNIST dataset and then on the poisoned dataset.
3. gen_and_train_cifar10.py - this script generates CIFAR10 dataset with one class triggered using a Gotham Instagram filter, and trains a model on various dataset poisoning percentages.
4. gen_and_train_imdb_glovebilstm.py - this script generates the IMDB dataset with one class triggered with a sentence, and trains a model on various dataset poisoning percentages.

CHAPTER

FOUR

CONTRIBUTING

`trojai` 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`, so be sure to get the go-ahead before working on something that is outside of the project's goals.
- Fork the `trojai` 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

FIVE

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

SIX

TROJAI PACKAGE

6.1 Subpackages

6.1.1 trojai.datagen package

Submodules

trojai.datagen.common_label_behaviors module

class trojai.datagen.common_label_behaviors.**StaticTarget**(*target*)

Bases: *trojai.datagen.label_behavior.LabelBehavior*

Sets label to a defined value

do(*y_true*)

Performs the actual specified label modification :param y_true: input label to be modified :return: the modified label

class trojai.datagen.common_label_behaviors.**WrappedAdd**(*add_val: int*, *max_num_classes: int = None*)

Bases: *trojai.datagen.label_behavior.LabelBehavior*

Adds a defined amount to each input label, with an optional maximum value around which labels are wrapped

do(*y_true: int*) → int

Performs the actual specified label modification :param y_true: input label to be modified :return: the modified label

trojai.datagen.common_label_behaviors.**logger = <Logger trojai.datagen.common_label_behavio**

Defines some common behaviors which are used to modify labels when designing an experiment with triggered and clean data

trojai.datagen.config module

class trojai.datagen.config.**TrojAICleanDataConfig**(*sign_xforms: Sequence[trojai.datagen.transform_interface.Transform] = None, bg_xforms: Sequence[trojai.datagen.transform_interface.Transform] = None, merge_obj: trojai.datagen.merge_interface.Merge = None, combined_xforms: Sequence[trojai.datagen.transform_interface.Transform] = None*)

Bases: object

validate() → None

class trojai.datagen.config.**ValidInsertLocationsConfig**(*algorithm: str = 'brute_force', min_val: Union[int, Sequence[int]] = 0, threshold_val: Union[float, Sequence[float]] = 5.0, num_boxes: int = 5, allow_overlap: Union[bool, Sequence[bool]] = False*)

Bases: object

Specifies which algorithm to use for determining the valid spots for trigger insertion on an image and all relevant parameters

validate()

Assess validity of provided values :return: None

class trojai.datagen.config.**XFormMergePipelineConfig**(*trigger_list: Sequence[trojai.datagen.entity.Entity] = None*, *trigger_sampling_prob: Sequence[float] = None*, *trigger_xforms: Sequence[trojai.datagen.transform_interface.Transform] = None*, *trigger_bg_xforms: Sequence[trojai.datagen.transform_interface.Transform] = None*, *trigger_bg_merge: trojai.datagen.merge_interface.Merge = None*, *trigger_bg_merge_xforms: Sequence[trojai.datagen.transform_interface.Transform] = None*, *overall_bg_xforms: Sequence[trojai.datagen.transform_interface.Transform] = None*, *overall_bg_triggerbg_merge: trojai.datagen.merge_interface.Merge = None*, *overall_bg_triggerbg_xforms: Sequence[trojai.datagen.transform_interface.Transform] = None*, *merge_type: str = 'insert'*, *per_class_trigger_frac: float = None*, *triggered_classes: Union[str, Sequence[Any]] = 'all'*)

Bases: object

Defines all configuration items necessary to run the XFormMerge Pipeline, and associated configuration validation.

NOTE: the argument list can be condensed into lists of lists, but that becomes a bit less intuitive to use. We need to think about how best we want to specify these argument lists.

validate()
Validates whether the configuration was setup properly, based on the merge_type. :return: None

validate_regenerate_mode()
Validates whether the configuration was setup properly, based on the merge_type. :return: None

trojai.datagen.config.**check_list_type**(*op_list*, *type*, *err_msg*)

trojai.datagen.config.**check_non_negative**(*val*, *name*)

trojai.datagen.config.**logger = <Logger trojai.datagen.config (WARNING)>**
Contains classes which define configuration used for transforming and modifying objects, as well as the associated validation routines. Ideally, a configuration class should be defined for every pipeline that is defined.

trojai.datagen.constants module

trojai.datagen.constants.**RANDOM_STATE_DRAW_LIMIT = 4294967295**
In the data generation process, every new entity that is generated gets a new random seed by drawing from np.random.RandomState.randint(), where the RandomState object comes from a master RandomState created at the beginning of the data generation process. The constant RANDOM_STATE_DRAW_LIMIT defines the argument passed into the randint(...) call.

The reason we create a new seed for every Entity is to enable reproducibility. Each Entity that is created may go through a series of transformations that include randomness at various stages. As such, having a seed associated with each Entity will enable us to reproduce those specific random variations easily.

trojai.datagen.datatype_xforms module

class trojai.datagen.datatype_xforms.**ToTensorXForm**(*num_dims: int = 3*)
Bases: *trojai.datagen.transform_interface.ImageTransform*

Transformation which defines the conversion of an input array to a tensor of a specified # of dimensions

do(*input_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the actual to->tensor conversion :param input_obj: the input Entity to be transformed :param random_state_obj: ignored :return: the transformed Entity

trojai.datagen.datatype_xforms.**logger = <Logger trojai.datagen.datatype_xforms (WARNING)>**
Defines data type transformations that may need to occur when processing different data sources

trojai.datagen.entity module

class trojai.datagen.entity.**Entity**
Bases: abc.ABC

An Entity is a generalization of a synthetic object. It could stand alone, or a composition of multiple entities. An Entity is composed of some data.See the README for further details on how Entity objects are intended to be used in the TrojAI pipeline.

abstract get_data()
Get the data associated with the Entity :return: return the internal representation of the image

trojai.datagen.entity.**logger = <Logger trojai.datagen.entity (WARNING)>**
Defines a generic Entity object, and an Entity convenience wrapper for creating Entities from numpy arrays.

trojai.datagen.experiment module

class trojai.datagen.experiment.**ClassicExperiment**(*data_root_dir: str*, *trigger_label_xform: trojai.datagen.label_behavior.LabelBehavior*, *stratify_split: bool = True*)
Bases: object

Defines a classic experiment, which consists of: 1) a specification of the clean data 2) a specification of the modified (triggered) data, and 3) a specification of the split of triggered/clean data for training/testing the model

create_experiment(*clean_data_csv: str*, *experiment_data_folder: str*, *mod_filename_filter: str = '*'*, *split_clean_trigger: bool = False*, *trigger_frac: float = 0.2*, *triggered_classes: Union[str, Sequence[Any]] = 'all'*, *random_state_obj: numpy.random.mtrand.RandomState = RandomState(MT19937) at 0x7FD5BF71F5A0*) → Union[Tuple, pandas.core.frame.DataFrame]

Creates an "experiment," which is a dataframe defining the data that should be used, and whether that data is triggered or not, and the true & actual label associated with that data point.

TODO:

[] - Have ability to accept multiple mod_data_folders such that we can sample from them all at a specified probability to have different triggers

Parameters

- **clean_data_csv** – path to file which contains a CSV specification of the clean data. The CSV file is expected to have the following columns: [file, label]
- **experiment_data_folder** – the folder which contains the data to mix with for the experiment.
- **mod_filename_filter** – a string filter for determining which files in the folder to consider, if only a a subset is to be considered for sampling
- **split_clean_trigger** – if True, then we return a list of DataFrames, where the triggered & non-triggered data are combined into one DataFrame, if False, we concatenate the triggered and non-triggered data into one DataFrame
- **trigger_frac** – the fraction of data which which should be triggered
- **triggered_classes** – either the string 'all', or a Sequence of labels which are to be triggered. If this parameter is 'all', then all classes will be triggered in the created experiment. Otherwise, only the classes in the list will be triggered at the percentage requested in the trigger_frac argument of the create_experiment function.
- **random_state_obj** – random state object

Returns

a dataframe of the data which consists of the experiment. The DataFrame has the following columns: file, true_label, train_label, triggered file - the file path of the data true_label - the actual label of the data train_label - the label of the data the model should be trained on.

This will be equal to true_label *if* triggered==False

triggered - a boolean value indicating whether this particular sample has a Trigger or not

trojai.datagen.experiment.**logger = <Logger trojai.datagen.experiment (WARNING)>**

Module which contains functionality for generating experiments

trojai.datagen.image_affine_xforms module

class trojai.datagen.image_affine_xforms.**PerspectiveXForm**(*xform_matrix*)
Bases: *trojai.datagen.transform_interface.ImageTransform*

Shifts the perspective of an input Entity

do(*input_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Performs the perspective shift on the input Entity. :param input_obj: the Entity to be transformed according to the specified perspective shift in the constructor. :param random_state_obj: ignored :return: the transformed Entity

class trojai.datagen.image_affine_xforms.**RandomPerspectiveXForm**(*perspectives: Sequence[str] = None*)
Bases: *trojai.datagen.transform_interface.ImageTransform*

Randomly shifts perspective of input Entity in available perspectives.

do(*input_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Samples from the possible perspectives according to the sampler specification and then applies that perspective to the input object :param input_obj: Entity to be randomly perspective shifted :param random_state_obj: allows for reprodcible sampling of random perspectives :return: the transformed Entity

class trojai.datagen.image_affine_xforms.**RandomRotateXForm**(*angle_choices: Sequence[float] = None*, *angle_sampler_prob: Sequence[float] = None*, *rotator_kwargs: Dict = None*)
Bases: *trojai.datagen.transform_interface.ImageTransform*

Implements a rotation of a random amount of degrees.

do(*input_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Samples from the possible angles according to the sampler specification and then applies that rotation to the input object :param input_obj: Entity to be randomly rotated :param random_state_obj: a random state used to maintain reproducibility through transformations :return: the transformed Entity

class trojai.datagen.image_affine_xforms.**RotateXForm**(*angle: int = 90*, *args: tuple = ()*, *kwargs: dict = None*)
Bases: *trojai.datagen.transform_interface.ImageTransform*

Implements a rotation of an Entity by a specified angle amount.

do(*input_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Performs the rotation specified by the RotateXForm object on an input :param input_obj: The Entity to be rotated :param random_state_obj: ignored :return: the transformed Entity

class trojai.datagen.image_affine_xforms.**UniformScaleXForm**(*scale_factor: float = 1*, *kwargs: dict = None*)
Bases: *trojai.datagen.transform_interface.ImageTransform*

Implements a uniform scale of a specified amount to an Entity

do(*input_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Performs the scaling on an input Entity using skimage.transform.rescale :param input_obj: the input object

to be scaled :param random_state_obj: ignored :return: the transformed Entity

trojai.datagen.image_affine_xforms.**get_predefined_perspective_xform_matrix**(*xform_str: str*, *rows: int*, *cols: int*) → numpy.ndarray

Returns an affine transform matrix for a string specification of a perspective transformation :param xform_str: a string specification of the perspective to transform

the object into.

Parameters

- **rows** – the number of rows of the image to be transformed to the specified perspective
- **cols** – the number of cols of the image to be transformed to the specified perspective

Returns a numpy array of shape (2,3) which specifies the affine transformation.

See:https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html?highlight=getaffinetransform for more information

trojai.datagen.image_affine_xforms.**logger = <Logger trojai.datagen.image_affine_xforms (WAI**

Module defines several affine transforms using various libraries to perform the actual transformation operation specified.

trojai.datagen.image_conversion_utils module

trojai.datagen.image_conversion_utils.**gray_to_rgb**(*img: numpy.ndarray*) → numpy.ndarray

Convert given grayscale image to RGB :param img: 1-channel grayscale image :return: image converted to RGB

trojai.datagen.image_conversion_utils.**logger = <Logger trojai.datagen.image_conversion_util**

Contains general utilities for dealing with channel formats

trojai.datagen.image_conversion_utils.**normalization_from_rgb**(*rgb_img: numpy.ndarray*, *alpha_ch: Optional[numpy.ndarray]*, *normalize: bool*, *original_n_chan: int*, *name: str*) → numpy.ndarray

Guard for output from rgb-only xforms :param rgb_img: 3-channel RGB image result from calling xform :param alpha_ch: alpha channel extracted at beginning of calling xform or None :param normalize: whether to convert rgb_img back to its original channel format :param original_n_chan: number of channels in its original channel format :param name: name of calling xform :return: if normalize is True the image corresponding to rgb_img converted to its original channel format, otherwise rgb_img unmodified, additional conversions can be added below, currently only RGB to RGBA is implemented

trojai.datagen.image_conversion_utils.**normalization_to_rgb**(*img: numpy.ndarray*, *normalize: bool*, *name: str*) → Tuple[numpy.ndarray, Optional[numpy.ndarray]]

Guard for input to RGB only xforms :param img: input image with variable number of channels :param normalize: whether to attempt to convert img from original channel format to 3-channel RGB :param name: name of calling xform :return: a 3-channel RGB array converted from img, additional conversions can be added below, currently only RGBA to RGB is implemented

trojai.datagen.image_conversion_utils.**rgb_to_rgba**(*img*, *alpha_ch: Optional[numpy.ndarray] = None*) → numpy.ndarray

Converts given image to RGBA, with optionally provided alpha_ch as its alpha channel :param img: 3-channel RGB image or 4-channel RGBA image :param alpha_ch: 1-channel array to be used as alpha value (optional), if img is RGBA this value is ignored :return: if img is 4-channel it is returned unmodified, if img is 3-channel this will return a new RGBA image with img as its RGB channels and either alpha_ch as its alpha channel if provided or a fully opaque alpha channel (max value for its datatype)

trojai.datagen.image_conversion_utils.**rgba_to_rgb**(*img: numpy.ndarray*) → Tuple[numpy.ndarray, Optional[numpy.ndarray]]

Split given 4-channel RGBA array into a 3-channel RGB array and a 1-channel alpha array :param img: given image to split, must be 3-channel or 4-channel :return: the first three channels of data as a 3-channel RGB image and the fourth channel of img as either a 1-channel alpha array, or None if img has only 3 channels

trojai.datagen.image_entity module

class trojai.datagen.image_entity.**GenericImageEntity**(*data: numpy.ndarray*, *mask: numpy.ndarray = None*)

Bases: *trojai.datagen.image_entity.ImageEntity*

A class which allows one to easily instantiate an ImageEntity object with an image and associated mask

get_data() → numpy.ndarray

Get the data associated with the ImageEntity :return: return a numpy.ndarray representing the image

get_mask() → numpy.ndarray

Get the mask associated with the ImageEntity :return: return a numpy.ndarray representing the mask

class trojai.datagen.image_entity.**ImageEntity**

Bases: *trojai.datagen.entity.Entity*

abstract get_mask() → numpy.ndarray

trojai.datagen.image_entity.**logger = <Logger trojai.datagen.image_entity (WARNING)>**

Defines a generic Entity object, and an Entity convenience wrapper for creating Entities from numpy arrays.

trojai.datagen.image_insert_utils module

trojai.datagen.image_insert_utils.**pattern_fit**(*chan_img: numpy.ndarray, chan_pattern: numpy.ndarray, chan_location: Sequence[Any]*) → bool

Returns True if the pattern at the desired location can fit into the image channel without wrap, and False otherwise

Parameters

- **chan_img** – a numpy.ndarray of shape (nrows, ncols) which represents an image channel
- **chan_pattern** – a numpy.ndarray of shape (prows, pcols) which represents a channel of the pattern
- **chan_location** – a Sequence of length 2, which contains the x/y coordinate of the top left corner of the pattern to be inserted for this specific channel

Returns True/False depending on whether the pattern will fit into the image

trojai.datagen.image_insert_utils.**valid_locations**(*img: numpy.ndarray, pattern: numpy.ndarray, algo_config: trojai.datagen.config.ValidInsertLocationsConfig, protect_wrap: bool = True*) → numpy.ndarray

Returns a list of locations per channel which the pattern can be inserted into the img_channel with an overlap algorithm dictated by the appropriate inputs

Parameters

- **img** – a numpy.ndarray which represents the image of shape: (nrows, ncols, nchans)
- **pattern** – the pattern to be inserted into the image of shape: (prows, pcols, nchans)
- **algo_config** – The provided configuration object specifying the algorithm to use and necessary parameters
- **protect_wrap** – if True, ensures that pattern to be inserted can fit without wrapping and raises an Exception otherwise

Returns A boolean mask of the same shape as the input image, with True indicating that that pixel is a valid location for placement of the specified pattern

trojai.datagen.image_size_xforms module

class trojai.datagen.image_size_xforms.**Pad**(*pad_amounts: tuple = (0, 0, 0, 0), mode: str = 'constant', pad_value: int = 0*)

Bases: *trojai.datagen.transform_interface.Transform*

Resizes an Entity

do(*img_obj: trojai.datagen.image_entity.ImageEntity, random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity

Perform the resizing :param img_obj: The input object to be resized according the specified configuration :param random_state_obj: ignored :return: The resized object

class trojai.datagen.image_size_xforms.**RandomPadToSize**(*new_size: tuple = (200, 200), mode: str = 'constant', pad_value: int = 0*)

Bases: *trojai.datagen.transform_interface.Transform*

Resizes an Entity

do(*img_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the resizing :param img_obj: The input object to be resized according the specified configuration :param random_state_obj: ignored :return: The resized object

class trojai.datagen.image_size_xforms.**RandomResize**(*new_size_minimum: tuple = (200, 200)*, *new_size_maximum: tuple = (300, 300)*, *interpolation: int = 2*)
Bases: *trojai.datagen.transform_interface.Transform*

Resizes an Entity

do(*img_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the resizing :param img_obj: The input object to be resized according the specified configuration :param random_state_obj: ignored :return: The resized object

class trojai.datagen.image_size_xforms.**RandomSubCrop**(*new_size: tuple = (200, 200)*)
Bases: *trojai.datagen.transform_interface.Transform*

Resizes an Entity

do(*img_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the resizing :param img_obj: The input object to be cropped according the specified configuration :param random_state_obj: ignored :return: The cropped object

class trojai.datagen.image_size_xforms.**Resize**(*new_size: tuple = (200, 200)*, *interpolation: int = 2*)
Bases: *trojai.datagen.transform_interface.Transform*

Resizes an Entity

do(*img_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the resizing :param img_obj: The input object to be resized according the specified configuration :param random_state_obj: ignored :return: The resized object

trojai.datagen.image_size_xforms.**logger = <Logger trojai.datagen.image_size_xforms (WARNIN**
Module contains various classes that relate to size transformations of input objects

trojai.datagen.image_triggers module

class trojai.datagen.image_triggers.**RandomRectangularPattern**(*num_rows: int*, *num_cols: int*, *num_chan: int*, *color_algorithm: str = 'channel_assign'*, *color_options: dict = None*, *pattern_style='graffiti'*, *dtype=<class 'numpy.uint8'>*, *random_state_obj: numpy.random.mtrand.RandomState = RandomState(MT19937) at 0x7FD5B287A160*)

Bases: *trojai.datagen.image_entity.ImageEntity*

Defines a random rectangular pattern

create() → None
Create the actual pattern :return: None

get_data() → numpy.ndarray
Get the image associated with the Entity :return: return a numpy.ndarray representing the image

get_mask() → numpy.ndarray
Get the mask associated with the Entity :return: return a numpy.ndarray representing the mask

class trojai.datagen.image_triggers.**RectangularPattern**(*num_rows: int*, *num_cols: int*, *num_chan: int*, *cval: int*, *dtype=<class 'numpy.uint8'>*)

Bases: *trojai.datagen.image_entity.ImageEntity*

Define a rectangular pattern

create() → None
Create the actual pattern :return: None

get_data() → numpy.ndarray
Get the image associated with the Entity :return: return a numpy.ndarray representing the image

get_mask() → numpy.ndarray
Get the mask associated with the Entity :return: return a numpy.ndarray representing the mask

class trojai.datagen.image_triggers.**ReverseLambdaPattern**(*num_rows: int*, *num_cols: int*, *num_chan: int*, *trigger_cval: Union[int, Sequence[int]]*, *bg_cval: Union[int, Sequence[int]] = 0*, *thickness: int = 1*, *pattern_style: str = 'graffiti'*, *dtype=<class 'numpy.uint8'>*)

Bases: *trojai.datagen.image_entity.ImageEntity*

Defines an alpha pattern

create() → None
Creates the alpha pattern and associated mask :return: None

get_data() → numpy.ndarray
Get the image associated with the Entity :return: return a numpy.ndarray representing the image

get_mask() → numpy.ndarray
Get the mask associated with the Entity :return: return a numpy.ndarray representing the mask

trojai.datagen.image_triggers.**logger = <Logger trojai.datagen.image_triggers (WARNING)>**
Defines various Trigger Entity objects

trojai.datagen.insert_merges module

class trojai.datagen.insert_merges.**FixedInsertTextMerge**(*location: int*)
Bases: *trojai.datagen.merge_interface.TextMerge*

do(*obj1: trojai.datagen.text_entity.TextEntity*, *obj2: trojai.datagen.text_entity.TextEntity*, *random_state_obj: numpy.random.mtrand.RandomState*)
Perform the actual merge operation :param obj1: the first Entity to be merged :param obj2: the second Entity to be merged :param random_state_obj: a numpy.random.RandomState object to ensure reproducibility :return: the merged Entity

class trojai.datagen.insert_merges.**InsertAtLocation**(*location: numpy.ndarray*, *protect_wrap: bool = True*)
Bases: *trojai.datagen.merge_interface.ImageMerge*

Inserts a provided pattern at a specified location

do(*img_obj: trojai.datagen.image_entity.ImageEntity*, *pattern_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*)
→ trojai.datagen.image_entity.ImageEntity
Inserts a pattern into an image, using the mask of the pattern to determine which specific pixels are modifiable :param img_obj: The background image into which the pattern is inserted :param pattern_obj: The pattern to be inserted. The mask associated with the pattern is used to determine which

specific pixes of the pattern are inserted into the img_obj

Parameters **random_state_obj** – ignored

Returns The merged object

class trojai.datagen.insert_merges.**InsertAtRandomLocation**(*method: str*, *algo_config: trojai.datagen.config.ValidInsertLocationsConfig*, *protect_wrap: bool = True*)
Bases: *trojai.datagen.merge_interface.ImageMerge*

Inserts a provided pattern at a random location, where valid locations are determined according to a provided algorithm specification

do(*img_obj: trojai.datagen.image_entity.ImageEntity*, *pattern_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*)
→ trojai.datagen.image_entity.ImageEntity
Perform the specified merge on the input Entities and return the merged Entity :param img_obj: the image object into which the pattern is to be inserted :param pattern_obj: the pattern object to be inserted :param random_state_obj: used to sample from the possible valid locations, by providing a random state,

we ensure reproducibility of the data

Returns the merged Entity

class trojai.datagen.insert_merges.**InsertRandomLocationNonzeroAlpha**
Bases: *trojai.datagen.merge_interface.ImageMerge*

Inserts a defined pattern into an image in a randomly selected location where the alpha channel is non-zero

do(*img_obj: trojai.datagen.image_entity.ImageEntity*, *pattern_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the described merge operation :param img_obj: The input object into which the pattern is to be inserted :param pattern_obj: The pattern object which is to be inserted into the image :param random_state_obj: used to sample from the possible valid locations, by providing a random state,

we ensure reproducibility of the data

Returns the merged object

class trojai.datagen.insert_merges.**InsertRandomWithMask**
Bases: *trojai.datagen.merge_interface.ImageMerge*

Inserts a defined pattern into an image in a randomly selected location where the specified mask is True

do(*img_obj: trojai.datagen.image_entity.ImageEntity*, *pattern_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the described merge operation :param img_obj: The input object into which the pattern is to be inserted :param pattern_obj: The pattern object which is to be inserted into the image :param random_state_obj: used to sample from the possible valid locations, by providing a random state,

we ensure reproducibility of the data

Returns the merged object

class trojai.datagen.insert_merges.**RandomInsertTextMerge**
Bases: *trojai.datagen.merge_interface.TextMerge*

do(*obj1: trojai.datagen.text_entity.TextEntity*, *obj2: trojai.datagen.text_entity.TextEntity*, *random_state_obj: numpy.random.mtrand.RandomState*)
Perform the actual merge operation :param obj1: the first Entity to be merged :param obj2: the second Entity to be merged :param random_state_obj: a numpy.random.RandomState object to ensure reproducibility :return: the merged Entity

trojai.datagen.insert_merges.**logger = <Logger trojai.datagen.insert_merges (WARNING)>**
Module which defines several insert style merge operations.

trojai.datagen.instagram_xforms module

class trojai.datagen.instagram_xforms.**FilterXForm**(*channel_order: str = 'BGR', pre_normalize: bool = True, post_normalize: bool = True*)

Bases: *trojai.datagen.transform_interface.ImageTransform*

Create filter xform, if no channel order is specified it is assumed to be in BGR order (opencv default), this refers only to the first 3 channels of input data as the alpha channel is handled independently

do(*input_obj: trojai.datagen.image_entity.ImageEntity, random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity

Compresses 3-channel image input image as a specified filetype and stores in memory, passes to into wand and applies filter, stores filtered image as specified filetype again in memory, which is then decompressed back into 3-channel image :param input_obj: entity to be transformed :param random_state_obj: object to hold random state and enable reproducibility :return:new entity with transform applied

abstract filter(*image: wand.image.Image*) → wand.image.Image

subclass defined function to be called by do :param image: wand Image to be filtered :return: filtered wand Image

class trojai.datagen.instagram_xforms.**GothamFilterXForm**(*channel_order: str = 'BGR', pre_normalize: bool = True, post_normalize: bool = True*)

Bases: *trojai.datagen.instagram_xforms.FilterXForm*

Class implementing Instagram's Gotham filter

filter(*image: wand.image.Image*) → wand.image.Image

modified from https://github.com/acoomans/instagram-filters/blob/master/instagram_filters/filters/gotham.py :param image: provided image :return: new filtered image

class trojai.datagen.instagram_xforms.**KelvinFilterXForm**(*channel_order: str = 'BGR', pre_normalize: bool = True, post_normalize: bool = True*)

Bases: *trojai.datagen.instagram_xforms.FilterXForm*

Class implementing Instagram's Kelvin filter

filter(*image: wand.image.Image*) → wand.image.Image

modified from https://github.com/acoomans/instagram-filters/blob/master/instagram_filters/filters/kelvin.py :param image: provided image :return: new filtered image

class trojai.datagen.instagram_xforms.**LomoFilterXForm**(*channel_order: str = 'BGR', pre_normalize: bool = True, post_normalize: bool = True*)

Bases: *trojai.datagen.instagram_xforms.FilterXForm*

Class implementing Instagram's Lomo filter

filter(*image: wand.image.Image*) → wand.image.Image

modified from https://github.com/acoomans/instagram-filters/blob/master/instagram_filters/filters/lomo.py :param image: provided image :return: new filtered image

class trojai.datagen.instagram_xforms.**NashvilleFilterXForm**(*channel_order: str = 'BGR', pre_normalize: bool = True, post_normalize: bool = True*)

Bases: *trojai.datagen.instagram_xforms.FilterXForm*

Class implementing Instagram's Nashville filter

filter(*image: wand.image.Image*) → wand.image.Image
modified from https://github.com/acoomans/instagram-filters/blob/master/instagram_filters/filters/nashville.py :param image: :return: new filtered image

class trojai.datagen.instagram_xforms.**NoOpFilterXForm**(*channel_order: str = 'BGR', pre_normalize: bool = True, post_normalize: bool = True*)

Bases: *trojai.datagen.instagram_xforms.FilterXForm*

No operation Transform for testing purposes

filter(*image: wand.image.Image*) → wand.image.Image
subclass defined function to be called by do :param image: wand Image to be filtered :return: filtered wand Image

class trojai.datagen.instagram_xforms.**ToasterXForm**(*channel_order: str = 'BGR', pre_normalize: bool = True, post_normalize: bool = True*)

Bases: *trojai.datagen.instagram_xforms.FilterXForm*

Class implementing Instagram's Toaster filter

filter(*image: wand.image.Image*) → wand.image.Image
modified from https://github.com/acoomans/instagram-filters/blob/master/instagram_filters/filters/toaster.py :param image: provided image :return: new filtered image

trojai.datagen.label_behavior module

class trojai.datagen.label_behavior.**LabelBehavior**

Bases: abc.ABC

A LabelBehavior is an operation performed on the "true" label to

abstract do(*input_label: int*) → int
Perform the actual desired label manipulation :param input_label: the input label to be manipulated :return: the manipulated label

trojai.datagen.merge_interface module

class trojai.datagen.merge_interface.**ImageMerge**

Bases: *trojai.datagen.merge_interface.Merge*

Subclass of merges for image entities. Prevents the usage of a text merge on an image entity, which has a distinct underlying data structure.

abstract do(*obj1: trojai.datagen.image_entity.ImageEntity, obj2: trojai.datagen.image_entity.ImageEntity, random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the actual merge operation :param obj1: the first Entity to be merged :param obj2: the second

Entity to be merged :param random_state_obj: a numpy.random.RandomState object to ensure reproducibility :return: the merged Entity

class trojai.datagen.merge_interface.**Merge**
Bases: abc.ABC

A Merge is defined as an operation on two Entities and returns a single Entity

abstract do(*obj1: trojai.datagen.entity.Entity*, *obj2: trojai.datagen.entity.Entity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.entity.Entity
Perform the actual merge operation :param obj1: the first Entity to be merged :param obj2: the second Entity to be merged :param random_state_obj: a numpy.random.RandomState object to ensure reproducibility :return: the merged Entity

class trojai.datagen.merge_interface.**TextMerge**
Bases: *trojai.datagen.merge_interface.Merge*

Subclass of merges for text entities. Prevents the usage of an image merge on a text entity, which has a distinct underlying data structure.

abstract do(*obj1: trojai.datagen.text_entity.TextEntity*, *obj2: trojai.datagen.text_entity.TextEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.text_entity.TextEntity
Perform the actual merge operation :param obj1: the first Entity to be merged :param obj2: the second Entity to be merged :param random_state_obj: a numpy.random.RandomState object to ensure reproducibility :return: the merged Entity

trojai.datagen.pipeline module

class trojai.datagen.pipeline.**Pipeline**
Bases: object

A pipeline is a composition of Entities, Transforms, and Merges to produce an output Entity

abstract process(*imglist: Iterable[trojai.datagen.entity.Entity]*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.entity.Entity
The method which executes the pipeline, moving data through each of Transform & Merge objects, with data flow being defined by the implementation. :param imglist: A list of Entity objects to be processed by the Pipeline :param random_state_obj: a random state to pass to the transforms and merge operation to ensure

reproducibility of Entities produced by the pipeline

Returns The output of the pipeline

trojai.datagen.static_color_xforms module

class trojai.datagen.static_color_xforms.**GrayscaleToRGBXForm**
Bases: *trojai.datagen.transform_interface.Transform*

Converts an 3-channel grayscale image to RGB

do(*input_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Convert the input object from 3-channel grayscale to RGB :param input_obj: Entity to be colorized :param random_state_obj: ignored :return: The colorized entity

class trojai.datagen.static_color_xforms.**RGBAtoRGB**
Bases: *trojai.datagen.transform_interface.Transform*

Converts input Entity from RGBA to RGB

do(*input_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the RGBA to RGB transformation :param input_obj: the Entity to be transformed :param random_state_obj: ignored :return: the transformed Entity

class trojai.datagen.static_color_xforms.**RGBtoRGBA**
Bases: *trojai.datagen.transform_interface.Transform*

Converts input Entity from RGB to RGBA

do(*input_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the RGBA to RGB transformation :param input_obj: the Entity to be transformed :param random_state_obj: ignored :return: the transformed Entity

trojai.datagen.static_color_xforms.**logger = <Logger trojai.datagen.static_color_xforms (WAI**
Defines several transformations related to static (non-random) color manipulation

trojai.datagen.transform_interface module

class trojai.datagen.transform_interface.**ImageTransform**
Bases: *trojai.datagen.transform_interface.Transform*

A Transform specific to ImageEntity objects

abstract do(*input_obj: trojai.datagen.image_entity.ImageEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.image_entity.ImageEntity
Perform the specified transformation :param input_obj: the input ImageEntity to be transformed :param random_state_obj: a random state used to maintain reproducibility through transformations :return: the transformed ImageEntity

class trojai.datagen.transform_interface.**TextTransform**
Bases: *trojai.datagen.transform_interface.Transform*

A Transform specific to TextEntity objects

abstract do(*input_obj: trojai.datagen.text_entity.TextEntity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.text_entity.TextEntity
Perform the specified transformation :param input_obj: the input TextEntity to be transformed :param random_state_obj: a random state used to maintain reproducibility through transformations :return: the transformed TextEntity

class trojai.datagen.transform_interface.**Transform**
Bases: abc.ABC

A Transform is defined as an operation on an Entity.

abstract do(*input_obj: trojai.datagen.entity.Entity*, *random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.entity.Entity
Perform the specified transformation :param input_obj: the input Entity to be transformed :param random_state_obj: a random state used to maintain reproducibility through transformations :return: the transformed Entity

trojai.datagen.utils module

trojai.datagen.utils.**logger = <Logger trojai.datagen.utils (WARNING)>**
Contains general utilities helpful for data generation

trojai.datagen.utils.**process_xform_list**(*input_obj: trojai.datagen.entity.Entity, xforms: Iterable[trojai.datagen.transform_interface.Transform], random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.entity.Entity
Processes a list of transformations in a serial fashion on a copy of the input X :param input_obj: input object which should be transformed by the list of

transformations

Parameters

- **xforms** – a list of Transform objects
- **random_state_obj** –

Returns The transformed object

trojai.datagen.xform_merge_pipeline module

class trojai.datagen.xform_merge_pipeline.**XFormMerge**(*xform_list: Sequence[Sequence[Sequence[trojai.datagen.transform_i*
merge_list: Sequence[trojai.datagen.merge_interface.Merge],
final_xforms: Sequence[trojai.datagen.transform_interface.Transform] = None)
Bases: *trojai.datagen.pipeline.Pipeline*

Implements a pipeline which is a series of cascading transform and merge operations. The following diagram shows 4 objects as a series of serial transforms + merges. Each pair of transformations is considered a "stage", and stages are processed in serial fashion. In the diagram below, the data that each stage processes is:

Stage1: obj1, obj2 Stage2: Stage1_output, obj3 Stage3: Stage2_output, obj4

This extends in the obvious way to more objects, depending on how deep the pipeline is.

obj1 –> xform obj3 –> xform obj4 –> xform + –> xform –> + –> xform –> + –> xform output /

obj2 –> xform

process(*imglist: Sequence[trojai.datagen.entity.Entity], random_state_obj: numpy.random.mtrand.RandomState*) → trojai.datagen.entity.Entity
Processes the provided objects according to the Xform->Merge->Xform paradigm. :param imglist: a sequence of Entity objects to be processed according to the pipeline :param random_state_obj: a random state to pass to the transforms and merge operation to ensure

reproducibility of Entities produced by the pipeline

Returns the modified & combined Entity object

trojai.datagen.xform_merge_pipeline.**logger = <Logger trojai.datagen.xform_merge_pipeline (**
Defines all functions and classes related to the transform+merge pipeline & data movement paradigm.

trojai.datagen.xform_merge_pipeline.**modify_clean_image_dataset**(*clean_dataset_rootdir: str*, *clean_csv_file: str*, *output_rootdir: str*, *output_subdir: str*, *mod_cfg: trojai.datagen.config.XFormMergePipelineC*, *method: str = 'insert'*, *random_state_obj: numpy.random.mtrand.RandomState = RandomState(MT19937) at 0x7FD5B25C4AF0*) → None

Modifies a clean dataset given a configuration

Parameters

- **clean_dataset_rootdir** – root directory where the clean data lives
- **clean_csv_file** – filename of the CSV file which contains information about the clean data The modification method determines which columns and information are expected in the CSV file.
- **output_rootdir** – root directory where the modified data will be stored
- **output_subdir** – subdirectory where the modified data will be stored. This is expected to be one level below the root-directory, and can prove useful if different types of modifications are stored in different subdirectories under the main root directory. An example tree structure might be: root_data
  - **modification_1** ... data ...
  - **modification_2** ... data ...
- **mod_cfg** – A configuration object for creating a modified dataset
- **method** – Can be "insert" only/ In the insert method, the function takes the clean image, and inserts a specified Entity (likely, a pattern) into the clean image. Additional modes to be added!
- **random_state_obj** – RandomState object to ensure reproduciblity of dataset

Returns None

trojai.datagen.xform_merge_pipeline.**modify_clean_text_dataset**(*clean_dataset_rootdir: str*, *clean_csv_file: str*, *output_rootdir: str*, *output_subdir: str*, *mod_cfg: trojai.datagen.config.XFormMergePipelineC*, *method='insert'*, *random_state_obj: numpy.random.mtrand.RandomState = RandomState(MT19937) at 0x7FD5B25C4C00*) → None

Modifies a clean image dataset given a configuration

Parameters

- **clean_dataset_rootdir** – root directory where the clean data lives
- **clean_csv_file** – filename of the CSV file which contains information about the clean data The modification method determines which columns and information are expected in the CSV file.
- **output_rootdir** – root directory where the modified data will be stored
- **output_subdir** – subdirectory where the modified data will be stored. This is expected to be one level below the root-directory, and can prove useful if different types of modifications are stored in different subdirectories under the main root directory. An example tree structure might be: root_data
  - **modification_1** ... data ...
  - **modification_2** ... data ...
- **mod_cfg** – A configuration object for creating a modified dataset
- **method** – Can only be "insert" In the insert method, the function takes the clean text blurb, and inserts a specified TextEntity (likely, a pattern) into the first text input object.
- **random_state_obj** – RandomState object to ensure reproduciblity of dataset

Returns None

trojai.datagen.xform_merge_pipeline.**subset_clean_df_by_labels**(*df*, *labels_to_include*)

Subsets a dataframe with an expected column 'label', to only keep rows which are in that list of labels to include :param df: the dataframe to subset :param labels_to_include: a list of labels to include, or a string 'all' indicating that everything should be kept :return: the subsetted data frame

Module contents

6.1.2 trojai.modelgen package

Subpackages

trojai.modelgen.architectures package

Submodules

trojai.modelgen.architectures.cifar10_architectures module

class trojai.modelgen.architectures.cifar10_architectures.**AlexNet**(*num_classes=10*)
Bases: torch.nn.Module

Modified AlexNet for CIFAR From: https://github.com/icpm/pytorch-cifar10/blob/master/models/AlexNet.py

forward(*x*)

class trojai.modelgen.architectures.cifar10_architectures.**Bottleneck**(*in_planes*, *growth_rate*)
Bases: torch.nn.Module

Bottleneck module in DenseNet Arch. See: https://arxiv.org/abs/1608.06993

forward(*x*)

class trojai.modelgen.architectures.cifar10_architectures.**DenseNet**(*block*, *num_block*, *growth_rate=12*, *reduction=0.5*, *num_classes=10*)
Bases: torch.nn.Module

From: https://github.com/icpm/pytorch-cifar10/blob/master/models/DenseNet.py

forward(*x*)

trojai.modelgen.architectures.cifar10_architectures.**DenseNet121**()

trojai.modelgen.architectures.cifar10_architectures.**DenseNet161**()

trojai.modelgen.architectures.cifar10_architectures.**DenseNet169**()

trojai.modelgen.architectures.cifar10_architectures.**DenseNet201**()

class trojai.modelgen.architectures.cifar10_architectures.**Transition**(*in_planes*, *out_planes*)
Bases: torch.nn.Module

Transition module in DenseNet Arch. See: https://arxiv.org/abs/1608.06993

forward(*x*)

trojai.modelgen.architectures.cifar10_architectures.**densenet_cifar**()

trojai.modelgen.architectures.mnist_architectures module

class trojai.modelgen.architectures.mnist_architectures.**BadNetExample**
Bases: torch.nn.Module

Mnist network from BadNets paper Input - 1x28x28 C1 - 1x28x28 (5x5 kernel) -> 16x24x24 ReLU S2 - 16x24x24 (2x2 kernel, stride 2) Subsampling -> 16x12x12 C3 - 16x12x12 (5x5 kernel) -> 32x8x8 ReLU S4 - 32x8x8 (2x2 kernel, stride 2) Subsampling -> 32x4x4 F6 - 512 -> 512 tanh F7 - 512 -> 10 Softmax (Output)

forward(*img*)

class trojai.modelgen.architectures.mnist_architectures.**ModdedLeNet5Net**(*channels=1*)
Bases: torch.nn.Module

A modified LeNet architecture that seems to be easier to embed backdoors in than the network from the original badnets paper Input - (1 or 3)x28x28 C1 - 6@28x28 (5x5 kernel) ReLU S2 - 6@14x14 (2x2 kernel, stride 2) Subsampling C3 - 16@10x10 (5x5 kernel) ReLU S4 - 16@5x5 (2x2 kernel, stride 2) Subsampling C5 - 120@1x1 (5x5 kernel) F6 - 84 ReLU F7 - 10 (Output)

forward(*img*)

Module contents

Submodules

trojai.modelgen.architecture_factory module

class trojai.modelgen.architecture_factory.**ArchitectureFactory**
Bases: abc.ABC

Factory object that returns architectures (untrained models) for training.

abstract new_architecture(***kwargs*) → torch.nn.Module
Returns a new architecture (untrained model) :return: an untrained torch.nn.Module

trojai.modelgen.config module

class trojai.modelgen.config.**ConfigInterface**
Bases: abc.ABC

Defines the interface for all configuration objects

class trojai.modelgen.config.**DefaultOptimizerConfig**(*training_cfg: trojai.modelgen.config.TrainingConfig = None*, *reporting_cfg: trojai.modelgen.config.ReportingConfig = None*)
Bases: *trojai.modelgen.config.OptimizerConfigInterface*

Defines the configuration needed to setup the DefaultOptimizer

get_device_type()
Returns the device associated w/ this optimizer configuration. Needed to save/load for UGE. :return (str): the device type represented as a string

static load(*fname*)
Loads a configuration from disk :param fname: the filename where the config is stored :return: the loaded configuration

save(*fname*)
Saves the optimizer configuration to a file :param fname: the filename to save the config to :return: None

class trojai.modelgen.config.**DefaultSoftToHardFn**
Bases: object

The default conversion from soft-decision outputs to hard-decision

class trojai.modelgen.config.**EarlyStoppingConfig**(*num_epochs: int = 5*, *val_loss_eps: float = 0.001*)
Bases: *trojai.modelgen.config.ConfigInterface*

Defines configuration related to early stopping.

validate()

class trojai.modelgen.config.**ModelGeneratorConfig**(*arch_factory: trojai.modelgen.architecture_factory.ArchitectureFactory*, *data: trojai.modelgen.data_manager.DataManager*, *model_save_dir: str*, *stats_save_dir: str*, *num_models: int*, *arch_factory_kwargs: dict = None*, *arch_factory_kwargs_generator: Callable = None*, *optimizer: Union[trojai.modelgen.optimizer_interface.OptimizerInterfa trojai.modelgen.config.DefaultOptimizerConfig*, *Sequence[Union[trojai.modelgen.optimizer_interface.Optim trojai.modelgen.config.DefaultOptimizerConfig]]] = None*, *parallel=False*, *amp=False*, *experiment_cfg: dict = None*, *run_ids: Union[Any*, *Sequence[Any]] = None*, *filenames: Union[str*, *Sequence[str]] = None*, *save_with_hash: bool = False*)
Bases: *trojai.modelgen.config.ConfigInterface*

Object used to configure the model generator

static load(*fname: str*)
Loads a saved modelgen_cfg object from data that was saved using the .save() function. :param fname: the filename where the modelgen_cfg object is saved :return: a ModelGeneratorConfig object

save(*fname: str*)
Saves the ModelGeneratorConfig object in two different parts. Every object within the config, except for the optimizer is saved in the .klass.save file, and the optimizer is saved separately. :param fname - the filename to save the configuration to :return: None

validate() → None
Validate the input arguments to construct the object :return: None

class trojai.modelgen.config.**OptimizerConfigInterface**
Bases: *trojai.modelgen.config.ConfigInterface*

abstract get_device_type()

abstract static load(*fname*)

save(*fname*)

class trojai.modelgen.config.**ReportingConfig**(*num_batches_per_logmsg: int = 100, disable_progress_bar: bool = False, num_epochs_per_metric: int = 1, num_batches_per_metrics: int = 50, tensorboard_output_dir: str = None, experiment_name: str = 'experiment'*)

Bases: *trojai.modelgen.config.ConfigInterface*

Defines all options to setup how data is reported back to the user while models are being trained

validate()

class trojai.modelgen.config.**RunnerConfig**(*arch_factory: trojai.modelgen.architecture_factory.ArchitectureFactory, data: trojai.modelgen.data_manager.DataManager, arch_factory_kwargs: dict = None, arch_factory_kwargs_generator: Callable = None, optimizer: Union[trojai.modelgen.optimizer_interface.OptimizerInterface, trojai.modelgen.config.DefaultOptimizerConfig, Sequence[Union[trojai.modelgen.optimizer_interface.OptimizerInterface, trojai.modelgen.config.DefaultOptimizerConfig]]] = None, parallel: bool = False, amp: bool = False, model_save_dir: str = '/tmp/models', stats_save_dir: str = '/tmp/model_stats', model_save_format: str = 'pt', run_id: Any = None, filename: str = None, save_with_hash: bool = False*)

Bases: *trojai.modelgen.config.ConfigInterface*

Container for all parameters needed to use the Runner to train a model.

static setup_optimizer_generator(*optimizer*, *data*)

Converts an optimizer specification to a generator, to be compatible with sequential training. :param optimizer: the optimizer to configure into a generator :param num_datasets: the number of datasets for which optimizers need to be created :return: A generator that returns optimizers for every dataset to be trained

validate() → None

Validate the RunnerConfig object :return: None

static validate_optimizer(*optimizer*, *data*)

Validates an optimzer configuration :param optimizer: the optimizer/optimizer configuration to be validated :param data: the data to be optimized :return:

class trojai.modelgen.config.**TorchTextOptimizerConfig**(*training_cfg: trojai.modelgen.config.TrainingConfig = None, reporting_cfg: trojai.modelgen.config.ReportingConfig = None, copy_pretrained_embeddings: bool = False*)

Bases: *trojai.modelgen.config.OptimizerConfigInterface*

Defines the configuration needed to setup the TorchTextOptimizer

get_device_type()

Returns the device associated w/ this optimizer configuration. Needed to save/load for UGE. :return (str):

the device type represented as a string

static load(*fname*)
Loads a configuration from disk :param fname: the filename where the config is stored :return: the loaded configuration

save(*fname*)
Saves the optimizer configuration to a file :param fname: the filename to save the config to :return: None

validate()

class trojai.modelgen.config.**TrainingConfig**(*device: Union[str, torch.device] = 'cpu', epochs: int = 10, batch_size: int = 32, lr: float = 0.0001, optim: Union[str, trojai.modelgen.optimizer_interface.OptimizerInterface] = 'adam', optim_kwargs: dict = None, objective: Union[str, Callable] = 'cross_entropy_loss', objective_kwargs: dict = None, save_best_model: bool = False, train_val_split: float = 0.05, val_data_transform: Callable[[Any], Any] = None, val_label_transform: Callable[[int], int] = None, val_dataloader_kwargs: dict = None, early_stopping: trojai.modelgen.config.EarlyStoppingConfig = None, soft_to_hard_fn: Callable = None, soft_to_hard_fn_kwargs: dict = None, lr_scheduler: Any = None, lr_scheduler_init_kwargs: dict = None, lr_scheduler_call_arg: Any = None, clip_grad: bool = False, clip_type: str = 'norm', clip_val: float = 1.0, clip_kwargs: dict = None, adv_training_eps: float = None, adv_training_iterations: int = None, adv_training_ratio: float = None*)

Bases: *trojai.modelgen.config.ConfigInterface*

Defines all required items to setup training with an optimizer

get_cfg_as_dict()
Returns a dictionary representation of the configuration :return: (dict) a dictionary

validate() → None
Validate the object configuration :return: None

class trojai.modelgen.config.**UGEConfig**(*queues: Union[trojai.modelgen.config.UGEQueueConfig, Sequence[trojai.modelgen.config.UGEQueueConfig]], queue_distribution: Sequence[float] = None, multi_model_same_gpu: bool = False*)

Bases: object

Defines a configuration for the UGE

validate()
Validate the UGEConfig object

class trojai.modelgen.config.**UGEQueueConfig**(*queue_name: str, gpu_enabled: bool, sync_mode: bool = False*)

Bases: object

Defines the configuration for a Queue w.r.t. UGE in TrojAI

validate() → None
Validate the UGEQueueConfig object

trojai.modelgen.config.**identity_function**(*x*)

trojai.modelgen.config.**logger = <Logger trojai.modelgen.config (WARNING)>**
Defines all configurations pertinent to model generation.

trojai.modelgen.config.**modelgen_cfg_to_runner_cfg**(*modelgen_cfg: trojai.modelgen.config.ModelGeneratorConfig*, *run_id=None*, *filename=None*) → trojai.modelgen.config.RunnerConfig
Convenience function which creates a RunnerConfig object, from a ModelGeneratorConfig object. :param modelgen_cfg: the ModelGeneratorConfig to convert :param run_id: run_id to be associated with the RunnerConfig :param filename: filename to be associated with the RunnerConfig :return: the created RunnerConfig object

trojai.modelgen.constants module

Defines valid devices on which models can be trained

trojai.modelgen.constants.**VALID_DEVICES = ['cpu', 'cuda']**
Defines valid loss functions which can be specified when configuring an optimizer implementing the OptimizerInterface

trojai.modelgen.constants.**VALID_LOSS_FUNCTIONS = ['cross_entropy_loss', 'BCEWithLogitsLoss**
Defines valid optimization algorithms which can be specified when configuring an optimizer implementing the OptimizerInterface

trojai.modelgen.constants.**VALID_OPTIMIZERS = ['adam', 'sgd', 'adamw']**
Defines the valid types of data that the modelgen pipeline can handle

trojai.modelgen.data_configuration module

class trojai.modelgen.data_configuration.**DataConfiguration**
Bases: object

class trojai.modelgen.data_configuration.**ImageDataConfiguration**
Bases: *trojai.modelgen.data_configuration.DataConfiguration*

class trojai.modelgen.data_configuration.**TextDataConfiguration**(*max_vocab_size: int = 25000*, *embedding_dim: int = 100*, *embedding_type: str = 'glove'*, *num_tokens_embedding_train: str = '6B'*, *text_field_kwargs: dict = None*, *label_field_kwargs: dict = None*)
Bases: *trojai.modelgen.data_configuration.DataConfiguration*

set_embedding_vectors_cfg()

validate()

trojai.modelgen.data_configuration.**logger = <Logger trojai.modelgen.data_configuration (WAI**

Configurations for various types of data

trojai.modelgen.data_descriptions module

File describes data description classes, which contain specific information that may be used in order to instantiate an architecture

class trojai.modelgen.data_descriptions.**CSVImageDatasetDesc**(*num_samples*, *shuffled*, *num_classes*)

Bases: *trojai.modelgen.data_descriptions.DataDescription*

Information potentially relevant to instantiating models to process image data

class trojai.modelgen.data_descriptions.**CSVTextDatasetDesc**(*vocab_size*, *unk_idx*, *pad_idx*)

Bases: *trojai.modelgen.data_descriptions.DataDescription*

Information potentially relevant to instantiating models to process text data

class trojai.modelgen.data_descriptions.**DataDescription**

Bases: object

Generic Data Description class from which all specific data type data descriptors

trojai.modelgen.data_manager module

class trojai.modelgen.data_manager.**DataManager**(*experiment_path: str, train_file: Union[str, Sequence[str]], clean_test_file: str, triggered_test_file: str = None, data_type: str = 'image', train_data_transform: Callable[[Any], Any] = <function DataManager.<lambda>>, train_label_transform: Callable[[int], int] = <function DataManager.<lambda>>, test_data_transform: Callable[[Any], Any] = <function DataManager.<lambda>>, test_label_transform: Callable[[int], int] = <function DataManager.<lambda>>, file_loader: Union[Callable[[str], Any], str] = 'default_image_loader', shuffle_train=True, shuffle_clean_test=False, shuffle_triggered_test=False, data_configuration: trojai.modelgen.data_configuration.DataConfiguration = None, custom_datasets: dict = None, train_dataloader_kwargs: dict = None, test_dataloader_kwargs: dict = None*)

Bases: object

Manages data from an experiment from trojai.datagen.

load_data()
Load experiment data as given from initialization. :return: Objects containing training and test, and triggered data if it was provided.

TODO: [ ] - extend the text data-type to have more input arguments, for example the tokenizer and FIELD options [ ] - need to support sequential training for text datasets

validate() → None
Validate the construction of the TrojaiDataManager object :return: None

TODO:

[ ] - think about whether the contents of the files passed into the DataManager should be validated, in addition to simply checking for existence, which is what is done now

trojai.modelgen.datasets module

class trojai.modelgen.datasets.**CSVDataset**(*path_to_data: str*, *csv_filename: str*, *true_label=False*, *path_to_csv=None*, *shuffle=False*, *random_state: Union[int, numpy.random.mtrand.RandomState] = None*, *data_loader: Union[str, Callable] = 'default_image_loader'*, *data_transform=<function identity_transform>*, *label_transform=<function identity_transform>*)
Bases: *trojai.modelgen.datasets.DatasetInterface*

Defines a dataset that is represented by a CSV file with columns "file", "train_label", and optionally "true_label". The file column should contain the path to the file that contains the actual data, and "train_label" refers to the label with which the data should be trained. "true_label" refers to the actual label of the data point, and can differ from train_label if the dataset is poisoned. A CSVDataset can support any underlying data that can be loaded on the fly and fed into the model (for example: image data)

get_data_description()

set_data_description()

class trojai.modelgen.datasets.**CSVTextDataset**(*path_to_data: str*, *csv_filename: str*, *true_label: bool = False*, *text_field: torchtext.data.Field = None*, *text_field_kwargs: dict = None*, *label_field: torchtext.data.LabelField = None*, *label_field_kwargs: dict = None*, *shuffle: bool = False*, *random_state=None*, ***kwargs*)
Bases: torchtext.data.Dataset, *trojai.modelgen.datasets.DatasetInterface*

Defines a text dataset that is represented by a CSV file with columns "file", "train_label", and optionally "true_label". The file column should contain the path to the file that contains the actual data, and "train_label" refers to the label with which the data should be trained. "true_label" refers to the actual label of the data point, and can differ from train_label if the dataset is poisoned. A CSVTextDataset can support text data, and differs from the CSVDataset because it loads all the text data into memory and builds a vocabulary from it.

build_vocab(*embedding_vectors_cfg*, *max_vocab_size*, *use_vocab=True*)

get_data_description()

set_data_description()

static sort_key(*ex*)

class trojai.modelgen.datasets.**DatasetInterface**(*path_to_data: str*, **args*, ***kwargs*)
Bases: torch.utils.data.Dataset

abstract get_data_description()

abstract set_data_description()

trojai.modelgen.datasets.**csv_dataset_from_df**(*path_to_data*, *data_df*, *true_label=False*, *shuffle=False*, *random_state: Union[int, numpy.random.mtrand.RandomState] = None*, *data_loader: Union[str, Callable] = 'default_image_loader'*, *data_transform=<function identity_transform>*, *label_transform=<function identity_transform>*)

Initializes a CSVDataset object from a DataFrame rather than a filepath. :param path_to_data: root folder where all the data is located :param data_df: the dataframe in which the data lives :param true_label: (bool) if True, then use the column "true_label" as the label associated with each datapoint. If False (default), use the column "train_label" as the label associated with each datapoint :param shuffle: if True, the dataset is shuffled before loading into the model :param random_state: if specified, seeds the random sampler when shuffling the data :param data_loader: either a string value (currently only supports *default_image_loader*), or a callable

function which takes a string input of the file path and returns the data

Parameters

- **data_transform** – a callable function which is applied to every data point before it is fed into the model. By default, this is an identity operation
- **label_transform** – a callable function which is applied to every label before it is fed into the model. By default, this is an identity operation.

trojai.modelgen.datasets.**csv_textdataset_from_df**(*data_df*, *true_label: bool = False*, *text_field: torchtext.data.Field = None*, *label_field: torchtext.data.LabelField = None*, *shuffle: bool = False*, *random_state=None*, ***kwargs*)

Initializes a CSVDataset object from a DataFrame rather than a filepath. :param data_df: the dataframe in which the data lives :param true_label: if True, then use the column "true_label" as the label associated with each :param text_field: defines how the text data will be converted to

a Tensor. If none, a default will be provided and tokenized with spacy

Parameters

- **label_field** – defines how to process the label associated with the text
- **max_vocab_size** – the maximum vocabulary size that will be built
- **shuffle** – if True, the dataset is shuffled before loading into the model
- **random_state** – if specified, seeds the random sampler when shuffling the data
- **kwargs** – any additional keyword arguments, currently unused

trojai.modelgen.datasets.**default_image_file_loader**(*img_loc*)

trojai.modelgen.datasets.**identity_transform**(*x*)

trojai.modelgen.datasets.**logger = <Logger trojai.modelgen.datasets (WARNING)>**
Define some basic default functions for dataset defaults. These allow Dataset objects to be pickled; vs lambda functions.

trojai.modelgen.default_optimizer module

class trojai.modelgen.default_optimizer.**DefaultOptimizer**(*optimizer_cfg: trojai.modelgen.config.DefaultOptimizerConfig = None*)
Bases: *trojai.modelgen.optimizer_interface.OptimizerInterface*

Defines the default optimizer which trains the 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

Returns a string representing the device used to train the model

static load(*fname: str*) → trojai.modelgen.optimizer_interface.OptimizerInterface
Reconstructs a DefaultOptimizer, by loading the configuration used to construct the original DefaultOptimizer, and then creating a new DefaultOptimizer object from the saved configuration :param fname: The filename of the saved optimzier :return: a DefaultOptimizer object

save(*fname: str*) → None
Saves the configuration object used to construct the DefaultOptimizer. NOTE: because the DefaultOptimizer object itself is not persisted, but rather the

DefaultOptimizerConfig object, the state of the object is not persisted!

Parameters **fname** – the filename to save the DefaultOptimizer's configuration.

Returns None

test(*net: torch.nn.Module, clean_data: trojai.modelgen.datasets.CSVDataset, triggered_data: trojai.modelgen.datasets.CSVDataset, clean_test_triggered_labels_data: trojai.modelgen.datasets.CSVDataset, torch_dataloader_kwargs: dict = None*) → dict
Test the trained network :param net: the trained module to run the test data through :param clean_data: the clean Dataset :param triggered_data: the triggered Dataset, if None, not computed :param clean_test_triggered_labels_data: triggered part of the training dataset but with correct labels; see

DataManger.load_data for more information.

Parameters **torch_dataloader_kwargs** – any keyword arguments to pass directly to PyTorch's DataLoader

Returns a dictionary of the statistics on the clean and triggered data (if applicable)

train(*net: torch.nn.Module, dataset: trojai.modelgen.datasets.CSVDataset, torch_dataloader_kwargs: dict = None, use_amp: bool = False) -> (torch.nn.Module, typing.Sequence[trojai.modelgen.training_statistics.EpochStatistics], <class 'int'>*)
Train the network. :param net: the network to train :param dataset: the dataset to train the network on :param torch_dataloader_kwargs: any additional kwargs to pass to PyTorch's native DataLoader :param use_amp: if True, uses automated mixed precision for FP16 training. :return: the trained network, and a list of EpochStatistics objects which contain the statistics for training,

and the # of epochs on which the net was trained

train_epoch(*model: torch.nn.Module*, *train_loader: torch.utils.data.DataLoader*, *val_clean_loader: torch.utils.data.DataLoader*, *val_triggered_loader: torch.utils.data.DataLoader*, *epoch_num: int*, *use_amp: bool = False*)

Runs one epoch of training on the specified model

Parameters

- **model** – the model to train for one epoch
- **train_loader** – a DataLoader object pointing to the training dataset
- **val_clean_loader** – a DataLoader object pointing to the validation dataset that is clean
- **val_triggered_loader** – a DataLoader object pointing to the validation dataset that is triggered
- **epoch_num** – the epoch number that is being trained
- **use_amp** – if True use automated mixed precision for FP16 training.

Returns a list of statistics for batches where statistics were computed

trojai.modelgen.default_optimizer.**split_val_clean_trig**(*val_dataset*)

Splits the validation dataset into clean and triggered.

Parameters **val_dataset** – the validation dataset to split

Returns A tuple of the clean & triggered validation dataset

trojai.modelgen.default_optimizer.**train_val_dataset_split**(*dataset: torch.utils.data.Dataset*, *split_amt: float*, *val_data_transform: Callable*, *val_label_transform: Callable)* -> *(torch.utils.data.Dataset*, *torch.utils.data.Dataset*)

Splits a PyTorch dataset (of type: torch.utils.data.Dataset) into train/test TODO:

[ ] - specify random seed to torch splitter

Parameters

- **dataset** – the dataset to be split
- **split_amt** – fraction specifying the validation dataset size relative to the whole. 1-split_amt will be the size of the training dataset
- **val_data_transform** – (function: any -> any) how to transform the validation data to fit into the desired model and objective function
- **val_label_transform** – (function: any -> any) how to transform the validation labels

Returns a tuple of the train and validation datasets

trojai.modelgen.model_generator module

class trojai.modelgen.model_generator.**ModelGenerator**(*configs: Union[trojai.modelgen.config.ModelGeneratorConfig, Sequence[trojai.modelgen.config.ModelGeneratorConfig]], *args, **kwargs*)

Bases: *trojai.modelgen.model_generator_interface.ModelGeneratorInterface*

Generates models based on requested data and saves each to a file.

run(**args*, ***kwargs*) → None
Train and save models as specified. :return: None

validate() → None
Validate the provided input when constructing the ModelGenerator interface

trojai.modelgen.model_generator_interface module

class trojai.modelgen.model_generator_interface.**ModelGeneratorInterface**(*configs: Union[trojai.modelgen.conf Sequence[trojai.modelgen.co*

Bases: abc.ABC

Generates models based on requested data and saves each to a file.

abstract run() → None
Train and save models as specified. :return: None

trojai.modelgen.model_generator_interface.**validate_model_generator_interface_input**(*configs: Union[troj Sequence[tro* → None

Validates a ModelGeneratorConfig :param configs: (ModelGeneratorConfig or sequence) configurations to be used for model generation :return None

trojai.modelgen.optimizer_interface module

class trojai.modelgen.optimizer_interface.**OptimizerInterface**

Bases: abc.ABC

Object that performs training and testing of TrojAI models.

abstract get_cfg_as_dict() → dict
Return a dictionary with key/value pairs that describe the parameters used to train the model.

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

abstract 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

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

abstract test(*model: torch.nn.Module*, *clean_test_data: torch.utils.data.Dataset*, *triggered_test_data: torch.utils.data.Dataset*, *clean_test_triggered_labels_data: torch.utils.data.Dataset*, *torch_dataloader_kwargs*) → dict

Perform whatever tests desired on the model with clean data and triggered data, return a dictionary of results. :param model: (torch.nn.Module) Trained Pytorch model :param clean_test_data: (CSVDataset) Object containing clean test data :param triggered_test_data: (CSVDataset or None) Object containing triggered test data, None if triggered data

was not provided for testing

Parameters

- **clean_test_triggered_labels_data** – triggered part of the training dataset but with correct labels; see DataManger.load_data for more information.
- **torch_dataloader_kwargs** – additional arguments to pass to PyTorch's DataLoader class

Returns

(dict) Dictionary of test accuracy results. Required key, value pairs are:

clean_accuracy: (float in [0, 1]) classification accuracy on clean data clean_n_total: (int) number of examples in clean test set

The following keys are optional, but should be used if triggered test data was provided
triggered_accuracy: (float in [0, 1]) classification accuracy on triggered data triggered_n_total: (int) number of examples in triggered test set

NOTE: This list may be augmented in the future to allow for additional test data collection.

abstract train(*model: torch.nn.Module, data: torch.utils.data.Dataset, progress_bar_disable: bool, torch_dataloader_kwargs: dict = None) -> (torch.nn.Module, typing.Sequence[trojai.modelgen.training_statistics.EpochStatistics], <class 'int'>*)

Train the given model using parameters in self.training_params :param model: (torch.nn.Module) The untrained Pytorch model :param data: (CSVDataset) Object containing training data, output 0 from TrojaiDataManager.load_data() :param progress_bar_disable: (bool) Don't display the progress bar if True :param torch_dataloader_kwargs: additional arguments to pass to PyTorch's DataLoader class :return: (torch.nn.Module, EpochStatistics) trained model, a sequence of EpochStatistics objects (one for

each epoch), and the # of epochs with which the model was trained (useful for early stopping).

trojai.modelgen.runner module

class trojai.modelgen.runner.**Runner**(*runner_cfg: trojai.modelgen.config.RunnerConfig*, *persist_metadata: dict = None*)

Bases: object

Fundamental unit of model generation, which trains a model as specified in a RunnerConfig object.

run() → None

Trains a model and saves it and the associated model statistics

trojai.modelgen.runner.**add_numerical_extension**(*path*, *filename*)

trojai.modelgen.runner.**try_force_json**(*x*)

Tries to make a value JSON serializable

trojai.modelgen.runner.**try_serialize**(*d*, *u*)

trojai.modelgen.torchtext_optimizer module

class trojai.modelgen.torchtext_optimizer.**TorchTextOptimizer**(*optimizer_cfg: trojai.modelgen.config.TorchTextOptimizerCon = None*)

Bases: *trojai.modelgen.optimizer_interface.OptimizerInterface*

An optimizer for training and testing LSTM models. Currently in a prototype state.

convert_dataset_to_dataiterator(*dataset: trojai.modelgen.datasets.CSVTextDataset, batch_size: int = None*) → torchtext.data.iterator.Iterator

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

Returns a string representing the device used to train the model

static load(*fname: str*) → trojai.modelgen.optimizer_interface.OptimizerInterface
Reconstructs an TorchTextOptimizer, by loading the configuration used to construct the original TorchTextOptimizer, and then creating a new TorchTextOptimizer object from the saved configuration :param fname: The filename of the saved TorchTextOptimizer :return: an TorchTextOptimizer object

save(*fname: str*) → None
Saves the configuration object used to construct the TorchTextOptimizer. NOTE: because the TorchTextOptimizer object itself is not persisted, but rather the

TorchTextOptimizerConfig object, the state of the object does not persist!

Parameters **fname** – the filename to save the TorchTextOptimizer's configuration.

test(*model: torch.nn.Module, clean_data: trojai.modelgen.datasets.CSVTextDataset, triggered_data: trojai.modelgen.datasets.CSVTextDataset, clean_test_triggered_labels_data: trojai.modelgen.datasets.CSVTextDataset, progress_bar_disable: bool = False, torch_dataloader_kwargs: dict = None*) → dict
Test the trained network :param model: the trained module to run the test data through :param clean_data: the clean Dataset :param triggered_data: the triggered Dataset, if None, not computed :param clean_test_triggered_labels_data: triggered part of the training dataset but with correct labels; see

DataManger.load_data for more information.

Parameters

- **progress_bar_disable** – if True, disables the progress bar
- **torch_dataloader_kwargs** – additional arguments to pass to PyTorch's DataLoader class

Returns a dictionary of the statistics on the clean and triggered data (if applicable)

train(*net: torch.nn.Module, dataset: trojai.modelgen.datasets.CSVTextDataset, progress_bar_disable: bool = False, torch_dataloader_kwargs: dict = None) -> (torch.nn.Module, typing.Sequence[trojai.modelgen.training_statistics.EpochStatistics], <class 'int'>*)
Train the network. :param net: the model to train :param dataset: the dataset to train the network on :param progress_bar_disable: if True, disables the progress bar :param torch_dataloader_kwargs: additional arguments to pass to PyTorch's DataLoader class :return: the trained network, list of EpochStatistics objects, and the # of epochs on which teh net was trained

train_epoch(*model: torch.nn.Module*, *train_loader: torchtext.data.iterator.Iterator*, *val_loader: torchtext.data.iterator.Iterator*, *epoch_num: int*, *progress_bar_disable: bool = False*)

Runs one epoch of training on the specified model

Parameters

- **model** – the model to train for one epoch
- **train_loader** – a DataLoader object pointing to the training dataset
- **val_loader** – a DataLoader object pointing to the validation dataset
- **epoch_num** – the epoch number that is being trained
- **progress_bar_disable** – if True, disables the progress bar

Returns a list of statistics for batches where statistics were computed

static train_val_dataset_split(*dataset: torchtext.data.Dataset*, *split_amt: float*, *val_data_transform: Callable*, *val_label_transform: Callable)* -> *(torchtext.data.Dataset*, *torchtext.data.Dataset*)

Splits a torchtext dataset (of type: torchtext.data.Dataset) into train/test. NOTE: although this has the same functionality as default_optimizer.train_val_dataset_split, it works with a

torchtext.data.Dataset object rather than torch.utils.data.Dataset.

TODO: [ ] - specify random seed to torch splitter

Parameters

- **dataset** – the dataset to be split
- **split_amt** – fraction specificing the validation dataset size relative to the whole. 1-split_amt will be the size of the training dataset
- **val_data_transform** – (function: any -> any) how to transform the validation data to fit into the desired model and objective function
- **val_label_transform** – (function: any -> any) how to transform the validation labels

Returns a tuple of the train and validation datasets

trojai.modelgen.training_statistics module

class trojai.modelgen.training_statistics.**BatchStatistics**(*batch_num: int*, *batch_train_accuracy: float*, *batch_train_loss: float*)

Bases: object

Represents the statistics collected from training a batch NOTE: this is currently unused!

get_batch_num()

get_batch_train_acc()

get_batch_train_loss()

set_batch_train_acc(*acc*)

set_batch_train_loss(*loss*)

class trojai.modelgen.training_statistics.**EpochStatistics**(*epoch_num*, *training_stats=None*, *validation_stats=None*, *batch_training_stats=None*)

Bases: object

Contains the statistics computed for an Epoch

add_batch(*batches: Union[trojai.modelgen.training_statistics.BatchStatistics, Sequence[trojai.modelgen.training_statistics.BatchStatistics]]*)

get_batch_stats()

get_epoch_num()

get_epoch_training_stats()

get_epoch_validation_stats()

validate()

class trojai.modelgen.training_statistics.**EpochTrainStatistics**(*train_acc: float*, *train_loss: float*)

Bases: object

Defines the training statistics for one epoch of training

get_train_acc()

get_train_loss()

validate()

class trojai.modelgen.training_statistics.**EpochValidationStatistics**(*val_clean_acc*, *val_clean_loss*, *val_triggered_acc*, *val_triggered_loss*)

Bases: object

Defines the validation statistics for one epoch of training

get_val_acc()

get_val_clean_acc()

get_val_clean_loss()

get_val_loss()

get_val_triggered_acc()

get_val_triggered_loss()

validate()

class trojai.modelgen.training_statistics.**TrainingRunStatistics**

Bases: object

Contains the statistics computed for an entire training run, a sequence of epochs TODO:

[ ] - have another function which returns detailed statistics per epoch in an easily serialized manner

add_best_epoch_val(*best_epoch*)

add_epoch(*epoch_stats: Union[trojai.modelgen.training_statistics.EpochStatistics, Sequence[trojai.modelgen.training_statistics.EpochStatistics]]*)

add_num_epochs_trained(*num_epochs*)

autopopulate_final_summary_stats()

Uses the information from the final epoch's final batch to auto-populate the following statistics:
final_train_acc final_train_loss final_val_acc final_val_loss

get_epochs_stats()

get_summary()
Returns a dictionary of the summary statistics from the training run

save_detailed_stats_to_disk(*fname: str*) → None
Saves all batch statistics for every epoch as a CSV file

Parameters **fname** – filename to save the detailed information to

Returns None

save_summary_to_json(*json_fname: str*) → None
Saves the training summary to a JSON file

set_final_clean_data_n_total(*n*)

set_final_clean_data_test_acc(*acc*)

set_final_clean_data_triggered_label_n(*n*)

set_final_clean_data_triggered_label_test_acc(*acc*)

set_final_train_acc(*acc*)

set_final_train_loss(*loss*)

set_final_triggered_data_n_total(*n*)

set_final_triggered_data_test_acc(*acc*)

set_final_val_clean_acc(*acc*)

set_final_val_clean_loss(*loss*)

set_final_val_combined_acc(*acc*)

set_final_val_combined_loss(*loss*)

set_final_val_triggered_acc(*acc*)

set_final_val_triggered_loss(*loss*)

trojai.modelgen.training_statistics.**logger = <Logger trojai.modelgen.training_statistics (**
Contains classes necessary for collecting statistics on the model during training

trojai.modelgen.uge_model_generator module

trojai.modelgen.uge_model_generator.**ALL_EXEC_PERMISSIONS = 365**
This file contains all the functionality needed to train models for a Univa Grid Engine (UGE) HPC cluster.

class trojai.modelgen.uge_model_generator.**UGEModelGenerator**(*configs: Union[trojai.modelgen.config.ModelGenerato Sequence[trojai.modelgen.config.ModelGenerat uge_config: trojai.modelgen.config.UGEConfig, working_directory: str = '/home/docs/uge_model_generator', validate_uge_dirs: bool = True*)

Bases: *trojai.modelgen.model_generator_interface.ModelGeneratorInterface*

Class which generates models utilizing a Univa Grid Engine

expand_modelgen_configs_to_process() → Sequence[trojai.modelgen.config.ModelGeneratorConfig]
Converts a sequence of ModelGeneratorConfig objects into another sequence of ModelGeneratorConfig objects such that each element in the sequence only creates one model. For example:

Input: cfgs = [cfg1->num_models=1, cfg2->num_models=2]. len(cfgs)=2 Output: cfgs = [cfg1->num_models=1, cfg2->num_models=1, cfg2->num_models=1]. len(cfgs)=3

NOTE: This will lead to multiple configs pointing to the same data on disk. I'm not sure if this is a problem for PyTorch or not, but this is something to investigate if unexpected results arise.

Returns expanded config configuration

get_queue_numjobs_assignment() → Sequence
Determine the number of jobs to give to each queue based on UGEConfig :return: a list of tuples, with each tuple containing the queue in index-0, and the number of jobs

assigned to that queue in index-1

run(*mock=False*) → None
Run's the actual UGE job. :param mock: if True, then it generates all the necessary scripts but doesn't execute the UGE command :return: None

validate() → None
Validate the input configuration

trojai.modelgen.utils module

trojai.modelgen.utils.**clamp**(*X*, *l*, *u*, *cuda=True*)
Clamps a tensor to lower bound l and upper bound u. :param X: the tensor to clamp. :param l: lower bound for the clamp. :param u: upper bound for the clamp. :param cuda: whether the tensor should be on the gpu.

trojai.modelgen.utils.**get_uniform_delta**(*shape*, *eps*, *requires_grad=True*)
Generates a troch uniform random matrix of shape within +-eps. :param shape: the tensor shape to create. :param eps: the epsilon bounds 0+-eps for the uniform random tensor. :param requires_grad: whether the tensor requires a gradient.

trojai.modelgen.utils.**make_trojai_model_dict**(*model*)

Create a TrojAI approved dictionary specification of a PyTorch model for saving to a file. E.g. for a trained model

'model': save_dict = make_trojai_model_dict(model) torch.save(save_dict, filename)

Parameters **model** – (torch.nn.Module) The desired model to be saved.

Returns (dict) dictionary containing TrojAI approved information about the model, which can also be used for later loading the model.

trojai.modelgen.utils.**resave_trojai_model_as_dict**(*file*, *new_loc=None*)

Load a fully serialized Pytorch model (i.e. whole model was saved instead of a specification) and save it as a TrojAI style dictionary specification.

Parameters

- **file** – (str) Location of the file to re-save
- **new_loc** – (str) Where to save the file if replacing the original is not desired

Module contents

6.2 Module contents

PYTHON MODULE INDEX

t

INDEX

E

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W

X