
Tiresias

Release 0.0.2

Thijs van Ede

Jun 22, 2022

CONTENTS:

1	Installation	3
1.1	From source	3
1.2	Dependencies	3
2	Usage	5
2.1	Overview	5
2.2	Command line tool	5
2.3	Code	6
3	Reference	9
3.1	Preprocessor	9
3.2	Tiresias	12
4	Contributors	15
4.1	Code	15
4.2	Academic Contributors	15
5	License	17
6	Citing	19
6.1	Bibtex	19
	Index	21

Tiresias provides a pytorch implementation of [Tiresias: Predicting Security Events Through Deep Learning](#). This code was implemented as part of the IEEE S&P 2022 [DeepCASE: Semi-Supervised Contextual Analysis of Security Events](#) paper. We ask people to cite both works when using the software for academic research papers, see [Citing](#) for more information.

INSTALLATION

The most straightforward way of installing Tiresias is via pip

```
pip install tiresias
```

1.1 From source

If you wish to stay up to date with the latest development version, you can instead download the [source code](#). In this case, make sure that you have all the required *dependencies* installed.

Once the dependencies have been installed, run:

```
pip install -e <path/to/directory/containing/tiresias/setup.py>
```

1.2 Dependencies

Tiresias requires the following python packages to be installed:

- array-lstm: <https://github.com/Thijsvanede/ArrayLSTM>
- numpy: <https://numpy.org/>
- scikit-learn: <https://scikit-learn.org/>
- pytorch: <https://pytorch.org/>

All dependencies should be automatically downloaded if you install Tiresias via pip. However, should you want to install these libraries manually, you can install the dependencies using the requirements.txt file

```
pip install -r requirements.txt
```

Or you can install these libraries yourself

```
pip install -U array-lstm numpy scikit-learn torch
```


USAGE

This section gives a high-level overview of the modules implemented by Tiresias. Furthermore it provides insights into the use of the command line tool. We also include several working examples to guide users through the code. For detailed documentation of individual methods, we refer to the [Reference](#) guide.

2.1 Overview

This section explains the design of Tiresias on a high level. Tiresias is a network that is implemented as a `torch-train` Module, which is an extension of `torch.nn.Module` including automatic methods to `fit()` and `predict()` data. This means it can be trained and used as any neural network module in the pytorch library.

In addition, we provide automatic methods to train and predict events given previous event sequences using the `torch-train` library. This follows a `scikit-learn` approach with `fit()`, `predict()` and `fit_predict()` methods. We refer to its documentation for a detailed description.

2.2 Command line tool

When Tiresias is installed, it can be used from the command line. The `__main__.py` file in the `tiresias` module implements this command line tool. The command line tool provides a quick and easy interface to predict sequences from `.csv` files. The full command line usage is given in its help page:

```
usage: tiresias.py [-h] [--csv CSV] [--txt TXT] [--length LENGTH] [--timeout TIMEOUT] [--
↳hidden HIDDEN] [-i INPUT] [-k K] [-o] [-t TOP] [--save SAVE] [--load LOAD] [-b BATCH_
↳SIZE] [-d DEVICE] [-e EPOCHS]
                    {train,predict}
```

Tiresias: Predicting Security Events Through Deep Learning

positional arguments:

{train,predict} mode in which to run Tiresias

optional arguments:

-h, --help show this help message and exit

Input parameters:

--csv	CSV	CSV events file to process	
--txt	TXT	TXT events file to process	
--length	LENGTH	sequence LENGTH	(default = 20)
--timeout	TIMEOUT	sequence TIMEOUT (seconds)	(default = inf)

(continues on next page)

(continued from previous page)

```
Tiresias parameters:
--hidden      HIDDEN          hidden dimension          (default = 128)
-i, --input   INPUT           input dimension         (default = 300)
-k, --k       K               number of concurrent memory cells (default = 4)
-o, --online
-t, --top     TOP             accept any of the TOP predictions (default = 1)
--save        SAVE            save Tiresias to specified file
--load        LOAD            load Tiresias from specified file

Training parameters:
-b, --batch-size BATCH_SIZE  batch size          (default = 128)
-d, --device  DEVICE         train using given device (cpu|cuda|auto) (default = auto)
-e, --epochs  EPOCHS         number of epochs to train with (default = 10)
```

2.2.1 Examples

Use first half of <data.csv> to train Tiresias and use second half of <data.csv> to predict and test the prediction.

```
python3 -m tiresias train --csv <data_train.csv> --save tiresias.save # Training
python3 -m tiresias predict --csv <data_test.csv> --load tiresias.save # Predicting
```

2.3 Code

To use Tiresias into your own project, you can use it as a standalone module. Here we show some simple examples on how to use the Tiresias package in your own python code. For a complete documentation we refer to the [Reference](#) guide.

2.3.1 Import

To import components from Tiresias simply use the following format

```
from tiresias          import <Object>
from tiresias.<module> import <Object>
```

For example, the following code imports the Tiresias neural network as found in the [Reference](#).

```
# Imports
from tiresias import Tiresias
```

2.3.2 Working example

In this example, we load data from either a .csv or .txt file and use that data to train and predict with Tiresias.

```
# import Tiresias and Preprocessor
from tiresias import Tiresias
from tiresias.preprocessor import Preprocessor

#####
#                               Load data                               #
#####

# Create preprocessor for loading data
preprocessor = Preprocessor(
    length = 20,          # Extract sequences of 20 items
    timeout = float('inf'), # Do not include a maximum allowed time between events
)

# Load data from csv file
X, y, label, mapping = preprocessor.csv("<path/to/file.csv>")
# Load data from txt file
X, y, label, mapping = preprocessor.txt("<path/to/file.txt>")

#####
#                               Tiresias                               #
#####

# Create Tiresias object
tiresias = Tiresias(
    input_size = 300, # Number of different events to expect
    hidden_size = 128, # Hidden dimension, we suggest 128
    output_size = 300, # Number of different events to expect
    k = 4, # Number of parallel LSTMs for ArrayLSTM
)

# Optionally cast data and Tiresias to cuda, if available
tiresias = tiresias.to("cuda")
X = X.to("cuda")
y = y.to("cuda")

# Train tiresias
tiresias.fit(
    X = X,
    y = y,
    epochs = 10,
    batch_size = 128,
)

# Predict using tiresias
y_pred, confidence = tiresias.predict_online(
    X = X,
    y = y,
    k = 3,
)
```

2.3.3 Modifying Tiresias

Tiresias itself works with an LSTM as implemented by ArrayLSTM from the *array-lstm* package. Suppose that we want to use a regular LSTM instead, we can simply create a new class that extends Tiresias and overwrite the `__init__` method to replace the ArrayLSTM with a regular LSTM.

```
# Imports
import torch.nn as nn
from tiresias import Tiresias

# Create a new class of Tiresias to overwrite the original
class TiresiasLSTM(Tiresias):

    # We overwrite the __init__ method
    def __init__(self, input_size, hidden_size, output_size, k):
        # Initialise super
        super().__init__(input_size, hidden_size, output_size, k)

        # Replace the lstm layer with a regular LSTM
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
```

REFERENCE

This is the reference documentation for the classes and methods objects provided by the Tiresias module.

3.1 Preprocessor

The Preprocessor class provides methods to automatically extract event sequences from various common data formats. To start sequencing, first create the Preprocessor object.

class `preprocessor.Preprocessor`(*length, timeout, NO_EVENT=- 1337*)

Preprocessor for loading data from standard data formats.

`Preprocessor.__init__`(*length, timeout, NO_EVENT=- 1337*)

Preprocessor for loading data from standard data formats.

Parameters

- **length** (*int*) – Number of events in context.
- **timeout** (*float*) – Maximum time between context event and the actual event in seconds.
- **NO_EVENT** (*int, default=-1337*) – ID of NO_EVENT event, i.e., event returned for context when no event was present. This happens in case of timeout or if an event simply does not have enough preceding context events.

3.1.1 Formats

We currently support the following formats:

- .csv files containing a header row that specifies the columns 'timestamp', 'event' and 'machine'.
- .txt files containing a line for each machine and a sequence of events (integers) separated by spaces.

Transforming .csv files into sequences is the quickest method and is done by the following method call:

`Preprocessor.csv`(*path, nrows=None, labels=None, verbose=False*)

Preprocess data from csv file.

Note: Format: The assumed format of a .csv file is that the first line of the file contains the headers, which should include `timestamp`, `machine`, `event` (and *optionally* `label`). The remaining lines of the .csv file will be interpreted as data.

Parameters

- **path** (*string*) – Path to input file from which to read data.
- **nrows** (*int*, *default=None*) – If given, limit the number of rows to read to *nrows*.
- **labels** (*int or array-like of shape=(n_samples,)*, *optional*) – If a *int* is given, label all sequences with given *int*. If an array-like is given, use the given labels for the data in file. Note: will overwrite any ‘label’ data in input file.
- **verbose** (*boolean*, *default=False*) – If *True*, prints progress in transforming input to sequences.

Returns

- **events** (*torch.Tensor of shape=(n_samples,)*) – Events in data.
- **context** (*torch.Tensor of shape=(n_samples, context_length)*) – Context events for each event in events.
- **labels** (*torch.Tensor of shape=(n_samples,)*) – Labels will be *None* if no labels parameter is given, and if data does not contain any ‘labels’ column.

Transforming `.txt` files into sequences is slower, but still possible using the following method call:

`Preprocessor.text(path, nrows=None, labels=None, verbose=False)`

Preprocess data from text file.

Note: Format: The assumed format of a text file is that each line in the text file contains a space-separated sequence of event IDs for a machine. I.e. for *n* machines, there will be *n* lines in the file.

Parameters

- **path** (*string*) – Path to input file from which to read data.
- **nrows** (*int*, *default=None*) – If given, limit the number of rows to read to *nrows*.
- **labels** (*int or array-like of shape=(n_samples,)*, *optional*) – If a *int* is given, label all sequences with given *int*. If an array-like is given, use the given labels for the data in file. Note: will overwrite any ‘label’ data in input file.
- **verbose** (*boolean*, *default=False*) – If *True*, prints progress in transforming input to sequences.

Returns

- **events** (*torch.Tensor of shape=(n_samples,)*) – Events in data.
- **context** (*torch.Tensor of shape=(n_samples, context_length)*) – Context events for each event in events.
- **labels** (*torch.Tensor of shape=(n_samples,)*) – Labels will be *None* if no labels parameter is given, and if data does not contain any ‘labels’ column.

Future supported formats

Note: These formats already have an API entrance, but are currently **NOT** supported.

- `.json` files containing values for ‘timestamp’, ‘event’ and ‘machine’.
- `.ndjson` where each line contains a json file with keys ‘timestamp’, ‘event’ and ‘machine’.

Preprocessor.`json`(*path*, *labels=None*, *verbose=False*)

Preprocess data from json file.

Note: json preprocessing will become available in a future version.

Parameters

- **path** (*string*) – Path to input file from which to read data.
- **labels** (*int or array-like of shape=(n_samples,)*, *optional*) – If a int is given, label all sequences with given int. If an array-like is given, use the given labels for the data in file. Note: will overwrite any ‘label’ data in input file.
- **verbose** (*boolean*, *default=False*) – If True, prints progress in transforming input to sequences.

Returns

- **events** (*torch.Tensor of shape=(n_samples,)*) – Events in data.
- **context** (*torch.Tensor of shape=(n_samples, context_length)*) – Context events for each event in events.
- **labels** (*torch.Tensor of shape=(n_samples,)*) – Labels will be None if no labels parameter is given, and if data does not contain any ‘labels’ column.

Preprocessor.`ndjson`(*path*, *labels=None*, *verbose=False*)

Preprocess data from ndjson file.

Note: ndjson preprocessing will become available in a future version.

Parameters

- **path** (*string*) – Path to input file from which to read data.
- **labels** (*int or array-like of shape=(n_samples,)*, *optional*) – If a int is given, label all sequences with given int. If an array-like is given, use the given labels for the data in file. Note: will overwrite any ‘label’ data in input file.
- **verbose** (*boolean*, *default=False*) – If True, prints progress in transforming input to sequences.

Returns

- **events** (*torch.Tensor of shape=(n_samples,)*) – Events in data.
- **context** (*torch.Tensor of shape=(n_samples, context_length)*) – Context events for each event in events.

- **labels** (*torch.Tensor of shape=(n_samples,)*) – Labels will be None if no labels parameter is given, and if data does not contain any ‘labels’ column.

3.2 Tiresias

The Tiresias class uses the `torch-train` library for training and prediction. This class implements the neural network as described in the paper [Tiresias: Predicting Security Events Through Deep Learning](#).

class `tiresias.Tiresias(*args: Any, **kwargs: Any)`

Implementation of Tiresias

From [Tiresias: Predicting security events through deep learning](#) by Shen et al.

Note: This is a `batch_first=True` implementation, hence the `forward()` method expect inputs of `shape=(batch, seq_len, input_size)`.

input_size

Size of input dimension

Type

int

hidden_size

Size of hidden dimension

Type

int

output_size

Size of output dimension

Type

int

k

Number of parallel memory structures, i.e. cell states to use

Type

int

3.2.1 Initialization

`Tiresias.__init__(input_size, hidden_size, output_size, k)`

Implementation of Tiresias

Parameters

- **input_size** (*int*) – Size of input dimension
- **hidden_size** (*int*) – Size of hidden dimension
- **output_size** (*int*) – Size of output dimension
- **k** (*int*) – Number of parallel memory structures, i.e. cell states to use

3.2.2 Forward

As Tiresias is a Neural Network, it implements the `forward()` method which passes input through the entire network.

Tiresias.forward(X)

Forward data through the network

Parameters

X (*torch.Tensor of shape=(n_samples, seq_len)*) – Input of sequences, these will be one-hot encoded to an array of shape=(n_samples, seq_len, input_size)

Returns

result – Returns a probability distribution of the possible outputs

Return type

torch.Tensor of shape=(n_samples, size_out)

3.2.3 Fit

Tiresias inherits its fit method from the `torch-train` module. See the [documentation](#) for a complete reference.

Tiresias.fit(X, y, epochs=10, batch_size=32, learning_rate=0.01, criterion=torch.nn.NLLLoss, optimizer=torch.optim.SGD, variable=False, verbose=True, **kwargs)

Train the module with given parameters

Parameters

- **X** (*torch.Tensor*) – Tensor to train with
- **y** (*torch.Tensor*) – Target tensor
- **epochs** (*int, default=10*) – Number of epochs to train with
- **batch_size** (*int, default=32*) – Default batch size to use for training
- **learning_rate** (*float, default=0.01*) – Learning rate to use for optimizer
- **criterion** (*nn.Loss, default=nn.NLLLoss*) – Loss function to use
- **optimizer** (*optim.Optimizer, default=optim.SGD*) – Optimizer to use for training
- **variable** (*boolean, default=False*) – If True, accept inputs of variable length
- **verbose** (*boolean, default=True*) – If True, prints training progress

Returns

result – Returns self

Return type

self

3.2.4 Predict

The regular network gives a probability distribution over all possible output values. However, Tiresias outputs the k most likely outputs, therefore it overwrites the `predict()` method of the `Module` class from `torch-train`.

`Tiresias.predict(X, k=1, variable=False, verbose=True)`

Predict the k most likely output values

Parameters

- **X** (*torch.Tensor of shape=(n_samples, seq_len)*) – Input of sequences, these will be one-hot encoded to an array of shape=(n_samples, seq_len, input_size)
- **k** (*int, default=1*) – Number of output items to generate
- **variable** (*boolean, default=False*) – If True, predict inputs of different sequence lengths
- **verbose** (*boolean, default=True*) – If True, print output

Returns

- **result** (*torch.Tensor of shape=(n_samples, k)*) – k most likely outputs
- **confidence** (*torch.Tensor of shape=(n_samples, k)*) – Confidence levels for each output

In addition to regular prediction, Tiresias introduces *online prediction*. In this implementation, the network predicts outputs for given inputs and compares them to what actually occurred. If the prediction does not match the actual output event, we update the neural network before predicting the next events. This is done using the method `predict_online()`.

`Tiresias.predict_online(X, y, k=1, epochs=10, batch_size=32, learning_rate=0.0001, criterion=torch.nn.NLLLoss, optimizer=torch.optim.SGD, variable=False, verbose=True, **kwargs)`

Predict samples in X and update the network only if the prediction does not match y

Parameters

- **X** (*torch.Tensor*) – Tensor to predict/train with
- **y** (*torch.Tensor*) – Target tensor
- **k** (*int, default=1*) – Number of output items to generate
- **epochs** (*int, default=10*) – Number of epochs to train with
- **batch_size** (*int, default=32*) – Default batch size to use for training
- **learning_rate** (*float, default=0.01*) – Learning rate to use for optimizer
- **criterion** (*nn.Loss, default=nn.NLLLoss*) – Loss function to use
- **optimizer** (*optim.Optimizer, default=optim.SGD*) – Optimizer to use for training
- **variable** (*boolean, default=False*) – If True, accept inputs of variable length
- **verbose** (*boolean, default=True*) – If True, prints training progress

Returns

- **result** (*torch.Tensor of shape=(n_samples, k)*) – k most likely outputs
- **confidence** (*torch.Tensor of shape=(n_samples, k)*) – Confidence levels for each output

CONTRIBUTORS

This page lists all the contributors to this project. If you want to be involved in maintaining code or adding new features, please email [t\(dot\)s\(dot\)vanede\(at\)utwente\(dot\)nl](mailto:t(dot)s(dot)vanede(at)utwente(dot)nl).

4.1 Code

- Thijs van Ede

4.2 Academic Contributors

- Thijs van Ede
- Hojjat Aghakhani
- Noah Spahn
- Riccardo Bortolameotti
- Marco Cova
- Andrea Continella
- Maarten van Steen
- Andreas Peter
- Christopher Kruegel
- Giovanni Vigna

LICENSE**MIT License**

Copyright (c) 2021 Thijs van Ede

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CITING

To cite Tiresias please use the following publications:

van Ede, T., Aghakhani, H., Spahn, N., Bortolameotti, R., Cova, M., Continella, A., van Steen, M., Peter, A., Kruegel, C. & Vigna, G. (2022, May). DeepCASE: Semi-Supervised Contextual Analysis of Security Events. In 2022 Proceedings of the IEEE Symposium on Security and Privacy (S&P). IEEE. [PDF DeepCASE]

Shen, Y., Mariconti, E., Vervier, P. A., & Stringhini, G. (2018). Tiresias: Predicting security events through deep learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS) (pp. 592-605). [PDF Tiresias]

6.1 Bibtex

6.1.1 DeepCASE

```
@inproceedings{vanede2020deepcase,
  title={{DeepCASE: Semi-Supervised Contextual Analysis of Security Events}},
  author={van Ede, Thijs and Aghakhani, Hojjat and Spahn, Noah and Bortolameotti,
↪Riccardo and Cova, Marco and Continella, Andrea and van Steen, Maarten and Peter,
↪Andreas and Kruegel, Christopher and Vigna, Giovanni},
  booktitle={Proceedings of the IEEE Symposium on Security and Privacy (S&P)},
  year={2022},
  organization={IEEE}
}
```

6.1.2 Tiresias

```
@inproceedings{shen2018tiresias,
  title={Tiresias: Predicting security events through deep learning},
  author={Shen, Yun and Mariconti, Enrico and Vervier, Pierre Antoine and Stringhini,
↪Gianluca},
  booktitle={Proceedings of the 2018 ACM SIGSAC Conference on Computer and
↪Communications Security},
  pages={592--605},
  year={2018}
}
```


Symbols

`__init__()` (*preprocessor.Preprocessor method*), 9
`__init__()` (*tiresias.Tiresias method*), 12

C

`csv()` (*preprocessor.Preprocessor method*), 9

F

`fit()` (*tiresias.Tiresias method*), 13
`forward()` (*tiresias.Tiresias method*), 13

H

`hidden_size` (*tiresias.Tiresias attribute*), 12

I

`input_size` (*tiresias.Tiresias attribute*), 12

J

`json()` (*preprocessor.Preprocessor method*), 11

K

`k` (*tiresias.Tiresias attribute*), 12

N

`ndjson()` (*preprocessor.Preprocessor method*), 11

O

`output_size` (*tiresias.Tiresias attribute*), 12

P

`predict()` (*tiresias.Tiresias method*), 14
`predict_online()` (*tiresias.Tiresias method*), 14
`Preprocessor` (*class in preprocessor*), 9

T

`text()` (*preprocessor.Preprocessor method*), 10
`Tiresias` (*class in tiresias*), 12