DeepLog

Release 0.0.2

Thijs van Ede

Apr 12, 2022

CONTENTS:

DeepLog provides a pytorch implementation of Deeplog: Anomaly detection and diagnosis from system logs 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.

CHAPTER

ONE

INSTALLATION

The most straigtforward way of installing DeepLog is via pip

```
pip install deeplog
```

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/deeplog/setup.py>
```

1.2 Dependencies

DeepLog requires the following python packages to be installed:

- argformat: https://github.com/Thijsvanede/argformat
- numpy: https://numpy.org/
- scikit-learn: https://scikit-learn.org/
- pytorch: https://pytorch.org/

All dependencies should be automatically downloaded if you install DeepLog 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 argformat numpy scikit-learn torch
```

CHAPTER

TWO

USAGE

This section gives a high-level overview of the modules implemented by DeepLog. 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 DeepLog on a high level. DeepLog 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 DeepLog is installed, it can be used from the command line. The `__main__.py` file in the `deeplog` 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: deeplog.py [-h] [--csv CSV] [--txt TXT] [--length LENGTH] [--timeout TIMEOUT] [--
↪hidden HIDDEN]
                  [-i INPUT] [-l LAYERS] [-k TOP] [--save SAVE] [--load LOAD] [-b BATCH_
↪SIZE]
                  [-d DEVICE] [-e EPOCHS]
                  {train,predict}

Deeplog: Anomaly detection and diagnosis from system logs through deep learning

positional arguments:
  {train,predict}             mode in which to run DeepLog

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
```

(continues on next page)

(continued from previous page)

```
  --length    LENGTH           sequence LENGTH                          (default =   20)
  --timeout   TIMEOUT          sequence TIMEOUT (seconds)               (default =  inf)

DeepLog parameters:
  --hidden    HIDDEN           hidden dimension                         (default =   64)
  -i, --input INPUT            input  dimension                         (default =  300)
  -l, --layers LAYERS          number of lstm layers to use             (default =    2)
  -k, --top   TOP              accept any of the TOP predictions        (default =    1)
  --save      SAVE             save DeepLog to   specified file
  --load      LOAD             load DeepLog 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 DeepLog and use second half of <data.csv> to predict and test the prediction.

```
python3 -m deeplog train   --csv <data.csv> --save deeplog.save # Training
python3 -m deeplog predict --csv <data.csv> --load deeplog.save # Predicting
```

2.3 Code

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

2.3.1 Import

To import components from DeepLog simply use the following format

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

For example, the following code imports the DeepLog neural network as found in the *Reference*.

```
# Imports
from deeplog import DeepLog
```

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 DeepLog.

```
# import DeepLog and Preprocessor
from deeplog              import DeepLog
from deeplog.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>")

################################################################################
#                                   DeepLog                                    #
################################################################################

# Create DeepLog object
deeplog = DeepLog(
    input_size  = 300, # Number of different events to expect
    hidden_size = 64 , # Hidden dimension, we suggest 64
    output_size = 300, # Number of different events to expect
)

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

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

# Predict using deeplog
y_pred, confidence = deeplog.predict(
    X = X,
    y = y,
    k = 3,
)
```

CHAPTER

THREE

REFERENCE

This is the reference documentation for the classes and methods objects provided by the DeepLog 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 DeepLog

The DeepLog class uses the torch-train library for training and prediction. This class implements the neural network as described in the paper Deeplog: Anomaly detection and diagnosis from system logs through deep learning.

class deeplog.**DeepLog**(**args: Any*, ***kwargs: Any*)

3.2.1 Initialization

DeepLog.**__init__**(*input_size*, *hidden_size*, *output_size*, *num_layers=2*)

DeepLog model used for training and predicting logs.

Parameters

- **input_size** (*int*) – Dimension of input layer.
- **hidden_size** (*int*) – Dimension of hidden layer.
- **output_size** (*int*) – Dimension of output layer.
- **num_layers** (*int, default=2*) – Number of hidden layers, i.e. stacked LSTM modules.

3.2.2 Forward

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

DeepLog.**forward**(*X*)

Forward sample through DeepLog.

Parameters **X** (*tensor*) – Input to forward through DeepLog network.

Returns **result**

Return type tensor

3.2.3 Fit

DeepLog inherits its fit method from the torch-train module. See the documentation for a complete reference.

DeepLog.**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, DeepLog outputs the *k* most likely outputs, therefore it overwrites the `predict()` method of the `Module` class from torch-train.

`DeepLog.`**predict**(*X*, *y=None*, *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)
- **y** (*Ignored*) – Ignored
- **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

CHAPTER

FOUR

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.

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

CHAPTER

FIVE

LICENSE

CHAPTER

SIX

CITING

To cite DeepLog 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]

Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS) (pp. 1285-1298). [PDF DeepLog]

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 DeepLog

```
@inproceedings{du2017deeplog,
  title={Deeplog: Anomaly detection and diagnosis from system logs through deep learning}
↪,
  author={Du, Min and Li, Feifei and Zheng, Guineng and Srikumar, Vivek},
  booktitle={Proceedings of the 2017 ACM SIGSAC Conference on Computer and␣
↪Communications Security},
  pages={1285--1298},
  year={2017}
}
```

INDEX

Symbols

C

D

F

J

N

P

T