Padasip - open source library for adaptive signal processing in language Python

Matouš Cejnek*

CTU in Prague, FME, Department of Instrumentation and Control Engineering, Technická 4, 166 07 Prague 6, Czech Rebpublic

Abstract

The Padasip library is designed to simplify adaptive signal processing tasks within python (filtering, prediction, detection, reconstruction, classification). Also in this library is presented some new methods for adaptive signal processing. The library is designed to be used with datasets and also with real-time measuring (sample-after-sample feeding). The library is open source project distributed under MIT license. For code optimisation, this library uses Numpy for array operations.

Key-words: adaptive signal processing; adaptive filtering; detection; neural networks

1. Introduction

In the last few years, Python became benchmark language for various fields of research and computations. This is true especially in field of neural networks and deep-learning. Strong reason for this is fact, that companies like Google has interest in Python, Linux and Open Source technologies in general. This phenomena is probably caused by multiple reasons and according to current trends in industry and cybernetics (Industry 4.0, big data).

As was mentioned before, multiple great Python libraries exists for tasks related to deep learning (for example [1]). It is hard to find competition for Python in this field. However, in some parts of the machine learning field the other languages (for example R [2] and Matlab [3]) still has greater foundation. Especially the Python tools for classical adaptive signal processing are underdeveloped in comparison with other Python libraries. This fact is the key motivation behind creation of Padasip library.

1.1. Development

The development of the library started in the May of 2016 with primary focus on adaptive filtering. Future releases extend the library with more signal pre-processing functions and neural networks module. The last release 1.0.0 adds detection module into Padasip and remove some back compatibility with older version.

The integrity of the library is checked during every season with automated tests (on start and on end of the season). For the purpose of testing is used standard library *unittest*. Currently 18 tests are used to check the integrity of Padasip.

1.2. Installation and Integration

The simplest way of Padasip installation is with pip from terminal as follows

```
sudo pip install padasip
```

Other way is to download or clone from Padasip official github pages [4], manually or with git. Downloaded library can be placed in Python packages, or directly into target project (to avoid dependency). Padasip has only one dependency - Numpy [5]. In order to work with Padasip it is necessary to have Numpy installed.

The library can be imported as any other Python module. All examples in this paper utilises the following import

```
import padasip as pa
```

Documentation for all functions and features is at [6].

2. Current content of the library

2.1. Data preprocessing module

In this module are placed functions related to preprocessing of data.

2.1.1. Input matrix construction

This function creates input matrix from historical values. Example follows:

```
>>> a
array([1, 2, 3, 4, 5, 6])
>>> pa.input_from_history(a,3)
array([[1, 2, 3],
       [2, 3, 4],
       [3, 4, 5],
       [4, 5, 6]])
```

2.1.2. Linear discriminant analysis

Linear discriminant analysis (LDA) [7] is a method used to determine the features that separates some classes of items. The output of LDA may be used as a linear classifier, or for dimensionality reduction for purposes of classification. Code example follows:

```
new_x = pa.preprocess.LDA(x, labels, n)
```

*Kontakt na autora: matous.cejnek@fs.cvut.cz

2.1.3. Principal component analysis

Principal component analysis (PCA) is a statistical method how to convert a set of observations with possibly correlated variables into a data-set of linearly uncorrelated variables (principal components). The number of principal components is less or equal than the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance. Example follows:

```
new_x = pa.preprocess.PCA(x, n)
```

2.1.4. Data standardization

This function standardizes (z-score) the series according to equation

$$\mathbf{x}_s = \frac{\mathbf{x} - a}{b} \tag{1}$$

where x is time series to standardize, a is offset to remove and b scale to remove. Example follows:

```
xs = pa.standardize(x)
```

The inverse operation can be done as follows:

```
x = pa.standardize(xs, offset=a, scale=b)
```

2.2. Adaptive filtering module

In this module are stored classes functions related to adaptive filters. Example of NLMS filter usage follows:

```
f = pa.filters.AdaptiveFilter(model="NLMS", n
    =4, mu=0.1, w="random")
y, e, w = f.run(d, x)
```

All adaptive filters can be also used online - sample by sample feeding. For this purpose are implemented two functions:

```
f.adapt(d, x)
f.predict(x)
```

The *adapt* function adapts weighs of the filter according given target and regression vector. The function *predict* predict (or filter) new value from given regression vector.

2.2.1. The least-mean-squares

The least-mean-squares (LMS) adaptive filter [8] is the most popular adaptive filter.

The LMS adaptive filter could be described as

$$y(k) = w_1 \cdot x_1(k) + ... + w_n \cdot x_n(k), \tag{2}$$

or in a vector form

$$y(k) = \mathbf{x}^T(k)\mathbf{w}(k), \tag{3}$$

where k is discrete time index, $(.)^T$ denotes the transposition, $y(k)$ is filtered signal, $\mathbf{w}$ is vector of filter adaptive parameters and $\mathbf{x}$ is input vector (for a filter of size n) as follows

$$\mathbf{x}(k) = [x_1(k), ..., x_n(k)]. \tag{4}$$

The LMS weights adaptation could be described as follows

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k), \tag{5}$$

where $\Delta\mathbf{w}(k)$ is

$$\Delta\mathbf{w}(k) = \frac{1}{2}\mu\frac{\partial e^2(k)}{\partial \mathbf{w}(k)} \ = \mu \cdot e(k) \cdot textbfx(k), \tag{6}$$

where μ is the learning rate (step size) and $e(k)$ is error defined as

$$e(k) = d(k) - y(k). \tag{7}$$

2.2.2. The normalized least-mean-squares

The normalized least-mean-squares (NLMS) adaptive filter [8] is an extension of the popular LMS adaptive filter. The extension is based on normalization of learning rate. The learning rage μ is replaced by learning rate $\eta(k)$ normalized with every new sample according to input power as follows

$$\eta(k) = \frac{\mu}{\epsilon + ||\mathbf{x}(k)||^2}, \tag{8}$$

where $||\mathbf{x}(k)||^2$ is norm of input vector and ϵ is a small positive constant (regularisation term). This constant is introduced to preserve the stability in cases where the input is close to zero.

2.2.3. The recursive least squares

The update of Recursive Least Squares filter [9] may be described as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k), \tag{9}$$

where $\Delta\mathbf{w}(k)$ is obtained as follows

$$\Delta\mathbf{w}(k) = \mathbf{R}(k)\mathbf{x}(k)e(k), \tag{10}$$

where $e(k)$ is error and it is estimated according to filter output and desired value $d(k)$ as follows

$$e(k) = d(k) - y(k). \tag{11}$$

The $\mathbf{R}(k)$ is inverse of auto-correlation matrix and it is calculated as follows

$$\mathbf{R}(k) = \frac{1}{\mu}(\mathbf{R}(k-1) - \frac{\mathbf{R}(k-1)\mathbf{x}(k)\mathbf{x}(k)^T\mathbf{R}(k-1)}{\mu + \mathbf{x}(k)^T\mathbf{R}(k-1)\mathbf{x}(k)}). \tag{12}$$

The initial value of auto-correlation matrix should be set to

$$\mathbf{R}(0) = \frac{1}{\delta}\mathbf{I}, \tag{13}$$

where $\mathbf{I}$ is identity matrix and δ is small positive constant.

2.2.4. The generalized normalized gradient descent

The generalized normalized gradient descent (GNGD) adaptive filter [10] is an extension of the NLMS adaptive filter.

2.2.5. The affine projection

The Affine Projection (AP) algorithm is implemented according to the paper [11]. Usage of this filter should be benefical especially when input data is highly correlated. This filter is based on LMS. The difference is, that AP uses multiple input vectors in every sample. The number of vectors is called projection order. In this implementation the historic input vectors from input matrix are used as the additional input vectors in every sample.

The input for AP filter is created as follows

$$\mathbf{X}_{AP}(k) = (\mathbf{x}(k), ..., \mathbf{x}(k-L)), \tag{14}$$

where $\mathbf{X}_{AP}$ is filter input, L is projection order, k is discrete time index and $\mathbf{x}_k$ is input vector. The output of filter is calculated as follows:

$$\mathbf{y}_{AP}(k) = \mathbf{X}^T_{AP}(k)\mathbf{w}(k), \tag{15}$$

where $\mathbf{x}(k)$ is the vector of filter adaptive parameters. The vector of targets is constructed as follows

$$\mathbf{d}_{AP}(k) = (d(k), ..., d(k-L))^T, \tag{16}$$

where $d(k)$ is target in time k. The error of the filter is estimated as follows

$$\mathbf{e}_{AP}(k) = \mathbf{d}_{AP}(k) - \mathbf{y}_{AP}(k). \tag{17}$$

And the adaptation of adaptive parameters is calculated according to equation

$$\begin{aligned}\mathbf{w}_{AP}(k+1) = \mathbf{w}_{AP}(k+1) + \\ \mu\mathbf{X}_{AP}(k)(\mathbf{X}^T_{AP}(k)\mathbf{X}_{AP}(k) + \epsilon\mathbf{I})^{-1}\mathbf{e}_{AP}(k).\end{aligned} \tag{18}$$

During the filtering we are interested just in output of filter $y(k)$ and the error $e(k)$. These two values are the first elements in vectors: $\mathbf{y}_{AP}(k)$ for output and $\mathbf{e}_{AP}(k)$ for error.

2.3. Neural networks module

In this module is currently implemented only one neural network. This model is curently not a priority, because there are plenty of good Python modules featuring neural networks.

2.3.1. Multi-layer perceptron

The Multi-layer perceptron (MLP) is probably the most popular neural network in machine learning field. In this field is commonly used only with few layers (unlike in field of Deep learning). The Padasip MPL implementation is done according practical recommendation [12]. The rule for the learning rate selection (if not defined by user) of neuron j in layer i is as follows

$$\mu_{ij} = m^{-0.5}, \tag{19}$$

where m is number of nodes on input of current node.

2.4. Detection module

This module features two methods - Learning Entropy (LE) and Error and Learning Based Novelty Detection (ELBND). Both implemented methods can be used together with any adaptive filter from Padasip. Description of methods follows.

2.4.1. Error and Learning Based Novelty Detection

The ELBND [13] can describe every sample with vector of values estimated from the adaptive increments and the model error as follows

$$\text{ELBND}(k) = \Delta\mathbf{w}(k)e(k). \tag{20}$$

The output is a vector of values describing novelty in given sample. Padasip features two methods how to turn this vector into more convenient single value - maximum of absolute values and sum of absolute values.

It is important to highlight, that this method does not need any additional parameters, so there is no issues related to method tuning.

2.4.2. Learning Entropy

The LE [14] is window based function. Value for every sample is defined as follows

$$\text{LE}(k) = \frac{1}{n \cdot n_\alpha} \sum f(\Delta w_i(k), \alpha), \tag{21}$$

where the n is number of the adaptive weights, the n_α is number of used detection sensitivities

$$\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_{n_\alpha}]. \tag{22}$$

The function $f(\Delta w_{i,j})$ is defined as follows

$$\begin{aligned}&\Delta w_i(k), \alpha) = \\ &\quad \{\text{if } \left(|\Delta w_i(k)|\,; \alpha\overline{|\Delta w_{Mi}(k)|}\right) \text{ then } 1, \text{else } 0\},\end{aligned} \tag{23}$$

where $\overline{|\Delta w_{Mi}(k)|}$ is the mean value of the window used for the LE evaluation.

The optimal number of detection sensitivities and their values depends on task and data. The sensitivities should be chosen in range where the function $LE(k)$ returns a value lower than 1 for at least one sample in the data, and for at maximally one sample returns value of 0.

2.5. Miscellaneous functions module

In this module are implemented functions that does not belong to any other category. Currently this module features only functions for error evaluation. These error functions are often used for evaluation of an error rather than just the error itself or its mean value. All functions are done in the way, that user can pass just the vector of errors directly, or two vectors - true conditions and predicted conditions.

2.5.1. Mean absolute error

mean absolute error (MAE) is also known as MAD - mean absolute deviation. This metric is obtained as follows

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}(e_i). \tag{24}$$

Example code follows:

```
mse = pa.misc.MAE(x1, x2)
```

2.5.2. Mean squared error

Mean squared error (MSE) is also known as MSD. Relation of this error metric to data follows

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (e_i)^2. \tag{25}$$

2.5.3. Root-mean-square error

Root-mean-square error (RMSE) is also known as RMSD. The formula follows

$$\text{RMSE} = \sqrt{\text{MSE}}. \tag{26}$$

2.5.4. Logarithmic squared error

Logarithmic squared error returns a vector of squared error values in dB as follows

$$\mathbf{logSE} = 10 \log_{10}(\mathbf{e}^2). \tag{27}$$

3. Conclusion and discussion

3.1. Availability

More information about usage and implementations can be found at Padasip documentation webpage [6]. The software is released under MIT license. No commercial software is required to run Padasip. More examples and tutorials can be found at [15].

3.2. Performance

Padasip does not define any new data structure, but instead uses only standard Python and NumPy [5] data structures. In most of the functions it was possible to done almost all exhaustive operations with Numpy array operations, which uses underlying functions written in C/C++/Fortran.

3.3. Future Development

The main goal for next versions is extension of detection module with new functions (not necessarily adaptive functions, but not well implemented for Python in general). Next plan is extension of neural networks module with Radial basis network and Self organising maps.

Acknowledgement

This work was supported by student grant SGS17/070/OHK2/1T/12.

Nomenclature

ELBND	Error and learning based novelty detection (−)
LDA	Linear discriminant analysis (−)
LE	Learning Entropy (−)
LMS	Least mean squares (−)
MLP	Multilayer perceptron (−)
MAE	Mean absolute error (−)
MSE	Mean square error (−)
NLMS	Normalised least mean squares (−)
PDA	Principal component analysis (−)
SNR	Signal to noise ratio (−)
α	vector of LE sensitivities (−)
μ	learning rate (−)
ϵ	regularisation term (−)

References

[1] James Bergstra et al. "Theano: Deep learning on gpus with python". In: *NIPS 2011, BigLearning Workshop, Granada, Spain.* Vol. 3. Citeseer. 2011.

[2] R Core Team. "R language definition". In: *Vienna, Austria: R foundation for statistical computing* (2000).

[3] User's Guide Matlab. "The mathworks". In: *Inc., Natick, MA* (1992).

[4] *Padasip 1.0.0 documentation.* Github Inc. Available at: https://github.com/ (visited on 03/22/2017).

[5] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation". In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[6] Matouš Cejnek. *Padasip 1.0.0 documentation.* ASPICC. Available at: http://matousc89.github.io/padasip/ (visited on 03/22/2017).

[7] Ronald A Fisher. "The use of multiple measurements in taxonomic problems". In: *Annals of eugenics* 7.2 (1936), pp. 179–188.

[8] Ali H Sayed. *Fundamentals of adaptive filtering.* John Wiley & Sons, 2003.

[9] Ali H Sayed and Thomas Kailath. "Recursive least-squares adaptive filters". In: *The Digital Signal Processing Handbook* (1998), pp. 21–1.

[10] Danilo P Mandic. "A generalized normalized gradient descent algorithm". In: *IEEE Signal Processing Letters* 11.2 (2004), pp. 115–118.

[11] Alberto Gonzalez et al. "Affine projection algorithms: Evolution to smart and fast algorithms and applications". In: *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European.* IEEE. 2012, pp. 1965–1969.

[12] Yann A LeCun et al. "Efficient backprop". In: *Neural networks: Tricks of the trade.* Springer, 2012, pp. 9–48.

[13] Matouš Cejnek, Peter Mark Benes, and Ivo Bukovsky. "Another Adaptive Approach to Novelty Detection in Time Series". In: *Academy & Industry Research Collaboration Center (AIRCC)* (2014), pp. 341–351.

[14] Ivo Bukovsky. "Learning entropy: Multiscale measure for incremental learning". In: *Entropy* 15.10 (2013), pp. 4159–4187.

[15] Matouš Cejnek. *Python Adaptive Signal Processing Handbook.* ASPICC. Available at: https://github.com/matousc89/Python-Adaptive-Signal-Processing-Handbook (visited on 03/22/2017).