

A neural network for noise correlation classification: Supplementary material

Patrick Paitz

August 17, 2017

General Information

This is the documentation for the supplementary material to the paper '*A neural network for noise correlation classification*' by Patrick Paitz, Alexey Gohkberg and Andreas Fichtner in the *Geophysical Journal International*. **Reference needs to be added when publishing**. This document allows you to apply the algorithms described in the paper to a small example dataset and to therefore understand them in greater detail. The newly developed feature extraction algorithm (Wavelet Based Feature Extraction, WBFEE) is further described and an application example is presented.

This supplementary material includes a software toolbox consisting of four different packages:

1. Visual classification of seismic data
2. Feature extraction for time-series data (wavelet based and frequency domain)
3. Creating subsets out of larger datasets based on specific properties
4. Artificial Neural Networks for seismogram classification

The software is written by Patrick Paitz. For any questions and remarks please contact him via email: patrick.paitz@erdw.ethz.ch. The documentation has various interactions with code and the terminal. The following background colors are used for different types:

Terminal interaction: Input and execution of commands

Terminal outputs and interaction with outputs (dialogs)

Changes in existing .py files

Contents

1	Prerequisites	3
1.1	Initialize your project	3
2	Tool for visual classification of seismograms	4
2.1	About	4
2.2	Configuration	4
2.3	Using the Tool	4
3	Tool for feature extraction	6
3.1	About	6
3.1.1	Wavelet Based Feature Extraction 'WBFE'	6
3.1.2	Frequency Domain Feature Extraction 'FDFE'	9
3.2	Configuration	11
3.3	Using the tool	11
4	Tool for subset creation	14
5	Tool for seismogram classification with artificial neural networks	15
5.1	Training and testing the artificial neural network	15
5.2	Extracting specific data out of your initial dataset	17
5.3	Convergence and parameter tests	18

1 Prerequisites

In order to use the included software, some prerequisites must be met. A Python 3 distribution should be installed. Furthermore the following packages are needed (in brackets are the used versions, other versions may also work but have not been tested for stability):

- (os, glob, sys, inspect, pathlib, these are standard python libraries)
- numpy (v 1.12.0)
- scipy (v 0.18.1)
- pandas (v 0.19.2)
- matplotlib (v 2.0.0)
- PyQt4 (pyqt v 4.11.4)
- ObsPy (v 1.0.2) (Krischer et al. (2015))

If you do not want to install python and the packages manually, we suggest the use of miniconda. You can download miniconda from <https://conda.io/miniconda.html>. After that you can create a python environment to interfere with other python installations.

```
$ conda create -n ann python=3.5
$ conda config --add channels conda-forge
$ source activate ann
$ conda install numpy scipy pandas matplotlib pyqt=4.11.4 obspy
# If you want to deactivate your conda environamnet:
$ source deactivate ann
```

If these prerequisites are met, you are ready to initialize the toolbox. Therefore move the software package directory ('Ann') to your wanted path (including all the subfolders). Then go to that path.

```
$ mv Ann /your/path/
```

1.1 Initialize your project

You can have various projects at the same time. To initialize your project run the following:

```
$ cd /your/path/Ann
$ python initialize.py
```

You can choose if you want to create a project '1' or delete an existing project '2'. We start with (1). After entering 1 and confirming with ENTER you have to insert a project name. Lets call your project 'example' and confirm with ENTER. Now some folders are created:

```
...
created /FOLDER/subfolders
...
```

Once the project is initialized it can also be deleted again by running the same command and choosing the init mode '2'.

Note: deleting a project deletes all files created for that project so far - so make sure to **back up the data you want to keep outside of the software directory**. The project removal also needs to be confirmed in the terminal.

2 Tool for visual classification of seismograms

2.1 About

This python tool can be used to visually classify seismograms into two distinct categories - here called (1) "good" and (2) "bad". In the application example mentioned in the paper, this refers to (1) suitable and (2) unsuitable correlations for noise tomography respectively, but other classification examples can also be used.

2.2 Configuration

Before configuring the classification tool, please make sure to do the initialization procedure (section 1.1). Then you can adjust the details in your configuration file. You can open this file with the following command (after going to the right directory):

```
$ cd /your/path/Ann/CLASSTOOL
$ open mydetails.py # use your favorite editor
```

We will now see the details you can adjust:

```
project = 'example'
name     = 'My_Name'
fname    = 'example.npy'
path_waveforms = '/your/path/to/seismograms/*.SAC'
```

here you adjust the *project* variable to be the project you just created: 'example'. You can also specify a *name*. The variable *fname* specifies the name where the result of the (visual) seismogram classification is stored (more in the next step). It is important to specify the path to your seismic data in the variable *path_waveforms* as a string (in quotation marks) together with you file endings. The asterisk means that it will read all data located in the specified directory. If the path is set correctly and the project name is the exact same as specified in the previous step (*initialize.py*), you can run the classification tool.

2.3 Using the Tool

The classification tool has a graphical user interface (GUI) that is depicted in Figure 1. This can be started by running the file *rc.py* (for run classificationtool)

```
$ python rc.py
```

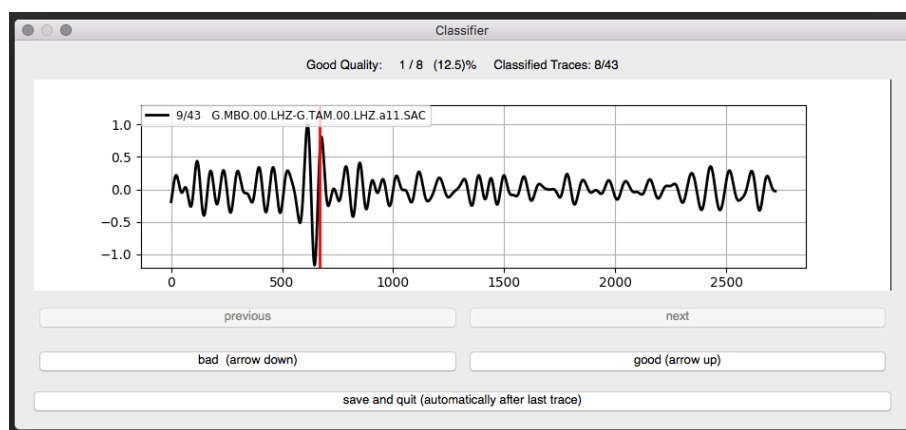


Figure 1: The interface of the classification tool. The red line indicates the estimated surface wave arrival based on a specified surface wave velocity. (Note: The seismograms are plotted normalized.)

At the top of the GUI you can see some statistics about your classification progress. By pressing the 'arrow up' on your keyboard or by clicking on the 'good' button you classify the currently plotted seismogram as 'good'. 'arrow down' classifies the current seismogram as bad. The red line indicates the

expected surface wave arrival based on an estimated surface wave velocity. This velocity can be changed in the *rc.py* function in line 25 (*self.velocity*). If you want to stop your classification at this point you can click save and quit. This will result in not all seismograms to be classified and once you start the program again you have to start over. If you classified the last seismogram, the program will save and close automatically. Some information will be viewed in the console window as well where you started the program:

```
...
Good Quality: 21 / 42 (50.0)% Classified Traces: 42/43
Good Quality: 22 / 43 (51.2)% Classified Traces: 43/43
```

The results then are saved into:

```
/your/path/Ann/CLASSTOOL/classified/project/fname
```

or, in your example:

```
/your/path/Ann/CLASSTOOL/classified/example/example.npy
```

3 Tool for feature extraction

3.1 About

3.1.1 Wavelet Based Feature Extraction 'WBFEE'

This tool is used for the wavelet based feature extraction (WBFEE) as described in the paper (see section 3). The purpose of the feature extraction is to reduce the dimensionality of our noise correlation (or any waveform-like data) in order to reduce the computational costs and improve the accuracy of an automated classification by an artificial neural network.

This tool will also run the frequency domain feature extraction (FDFE) in the background (the algorithm is based on Valentine and Woodhouse (2010)). For every iteration, a wavelet is extracted and nine features are saved in an array.

The WBFEE method combines different information sources described in the paper and is based on wavelets. The following paragraphs within this subsection are adjusted from Paitz (2016).

First we need at least one wavelet function. We decided to use a Morlet wavelet Ψ :

$$\Psi(t, \omega, s = 0.5) = \pi^{-\frac{1}{4}} (e^{i\omega t} - e^{-0.5*(\omega^2)}) e^{-0.5*(t^2)}, \quad t = [0, 1, \dots, l] \quad (1a)$$

with l denoting the length of the wavelet, ω a proxy for the number of significant oscillations, s denoting a scaling factor that windows the wavelet from $-2s\pi$ to $+2s\pi$ and i denoting the imaginary unit. The reason for choosing a Morlet wavelet is that possible surface wave arrivals of the noise correlation data seem to resemble a Morlet wavelet quite well.

In the first feature extraction step we extract the time at which the amplitude maximum of the noise correlation η occurs. With the noise correlation η at lag time t and feature number n , this time $t_{\eta_{max}}$ can be described as:

$$t_{\eta_{max}}(\eta) = t(\max(\eta(t))) \quad (2)$$

With ω denoting the number of significant oscillations, the start time t_s and end time t_e of the feature that can be described as:

$$t_s = t^-(\omega, \eta = 0, t_{\eta_{max}}) \quad (3a)$$

$$t_e = t^+(\omega, \eta = 0, t_{\eta_{max}}) \quad (3b)$$

where t^- and t^+ are the ω -th zero crossing before and after $t_{\eta_{max}}$. The duration of the feature t_d then is defined as:

$$t_d = t_e - t_s \quad (4)$$

Because the actual waveforms of the noise correlation are not necessarily symmetric in time, two additional parameters are introduced: τ_s as a time shift and τ_c as a time stretching. The times t_e and t_s are adjusted with these τ parameters so that:

$$t_s \rightarrow t_s + \tau_s - \tau_c \quad (5a)$$

$$t_e \rightarrow t_e + \tau_s + \tau_c \quad (5b)$$

The duration from Eq. 4 therefore becomes:

$$t_d \rightarrow t_d + 2\tau_c \quad (6)$$

The waveform Ψ fitting onto the noise correlation η in the interval $[t_s, t_e]$ for a normalized noise correlation can be defined as (by normalizing the real part \Re of the wavelet Ψ and using t_d of Eq. 6 in Eq. 1a):

$$\Psi(t, \Psi, t_d, \omega, \eta) = \frac{\Re\{\Psi(t_d, \omega)\}}{\max(\Re\{\Psi(t_d, \omega)\})} \times \max(\eta(t_{\eta_{max}})) \quad (7)$$

We choose the parameters ω, τ_s and τ_c so that the misfit function M :

$$M(\tau_s, \tau_c) = \sum_{t=t_s}^{t_e} (|\eta(t) - \Psi(\Psi, \tau_s, \tau_c, \eta)|) \quad (8a)$$

is minimal. We currently do this by iterating through all possibilities and keeping the parameter combination with the lowest misfit, but other and more efficient methods are possible. Values for τ_s within $(-40, 40)$, for τ_c within $(-5, 5)$ and for ω within $(0, 10)$ are identified to work well for our data.

Extracted Features

The set of extracted features for one iteration of WBFEE then consists of the following variables: The start-time t_s (Eq. 5a) and the duration t_d (Eq. 6). The ratio T between the expected first arrival time t_{est} obtained from the surface velocity $v_{section}$ from the noise correlation section, the inter-station distance d_{inter} and the start time:

$$T = \frac{t_s}{\frac{d_{inter}}{v_{section}}} \quad (9a)$$

The next extracted feature is the number of samples of the total noise correlation N . We can use that to reconstruct the wavelet and compare it to the actual noise correlation over the whole time period. The next stored feature is the side-lobe number ω . The feature fit F of waveform and the noise correlation as well as the residual energy R are also stored with:

$$F(t_s, t_e) = \frac{\sum_{t=t_s}^{t_e} (|\eta(t) - \Psi(\Psi, \tau_s, \tau_c, \eta)|)}{t_d} \quad (10a)$$

$$R(t) = \frac{\sum_{t=0}^{t=max(lagtime)} (|\eta(t) - \Psi(\Psi, \tau_s, \tau_c, \eta)|)}{N} \quad (10b)$$

Because sometimes the better fit is obtained when the wavelet is upside down, the polarity information p is also stored (in form of -1 for an upside down wavelet and $+1$ otherwise). The last stored feature is the Amplitude Ratio A between the maximum Amplitude of the Wavelet Ψ and maximum amplitude of the full noise correlation η . This becomes important for extracting more than one waveform. After storing the 9 variables into a vector, the wavelet is subtracted of the original noise correlation. This modified noise correlation can then be used to extract the next features and so on. Therefore, the ratio A will decrease as the number of extracted features increases. An overview about all the stored variables for one iteration of WBFEE is provided in Tab. 1.

Table 1: Extracted variables for one WBFE iteration. If more than one iteration is performed, the next iteration is performed on the noise correlation with the first waveform removed. So every iteration returns 9 features.

Variables extracted by the Wavelet Based Feature Extraction Algorithm		
#	Variable	Short description
1	t_s	Start time of the waveform (Eq. 5a)
2	t_d	Duration of the waveform (Eq. 6)
3	r_t	Start time ratio (Eq. 9a)
4	p	Polarity
5	ω	Number of significant oscillations of the wavelet
6	F	Fit of wavelet and noise correlation (Eq. 10a)
7	R	Residuals (Eq. 10b)
8	N	Number of samples within η
9	A	Amplitude ratio

3.1.2 Frequency Domain Feature Extraction 'FDFE'

Besides the WBF, this tool is also used to extract Features in the Frequency Domain. This approach was introduced by Valentine and Woodhouse (2010). An N -point representation P_n of the entire discretized power spectrum $f(\omega)$ is defined for a modest N as:

$$P_n = \frac{1}{\alpha \delta_\omega} \int_{\omega_a + (n-1)\delta_\omega}^{\omega_a + n\delta_\omega} |f(\omega)|^2 d\omega \quad (n = 1, 2, \dots, N) \quad (11a)$$

$$\alpha = \int_{\omega_a}^{\omega_b} |f(\omega)|^2 d\omega \quad (11b)$$

$$\delta_\omega = \frac{\omega_b - \omega_a}{N} \quad (11c)$$

with $\omega_a < \omega_b$ as outer bandpass limits.

The authors used $N = 15$, based on empirical decisions. In addition to this 15-point representation of the power-spectrum, the epicentral angle Δ and the event depth z are appended to the power spectrum representation, resulting in an 17-point representation of an earthquake event. All the parameters were chosen to be in the range $(0, 1)$. For example the depth therefore needs to be normalized over all available events. These representations are then used to train and test an ANN with 60 Nodes in one hidden layer with two output nodes with initialized weights in the range $(-1, 1)$. The output of $(1, 0)$ corresponds to an acceptable output whereas $(0, 1)$ corresponds to unacceptable waveforms. To improve confidence and reduce random errors, a committee of 10 independent networks is used and the output averaged over all ten ANNs. The training was independent. The first layer activation function was a sigmoid function and the output layer activation function a linear activation function (Valentine and Woodhouse, 2010).

The information about the epicentral angle and the event depth are not usable in the noise correlation scenario. But in addition to the normalized amplitude spectra we store α and the distance between the stations as separate inputs when implementing this method. We also normalized the amplitude spectra as well as the N -point representation so that the input to the neural network is within an acceptable range.

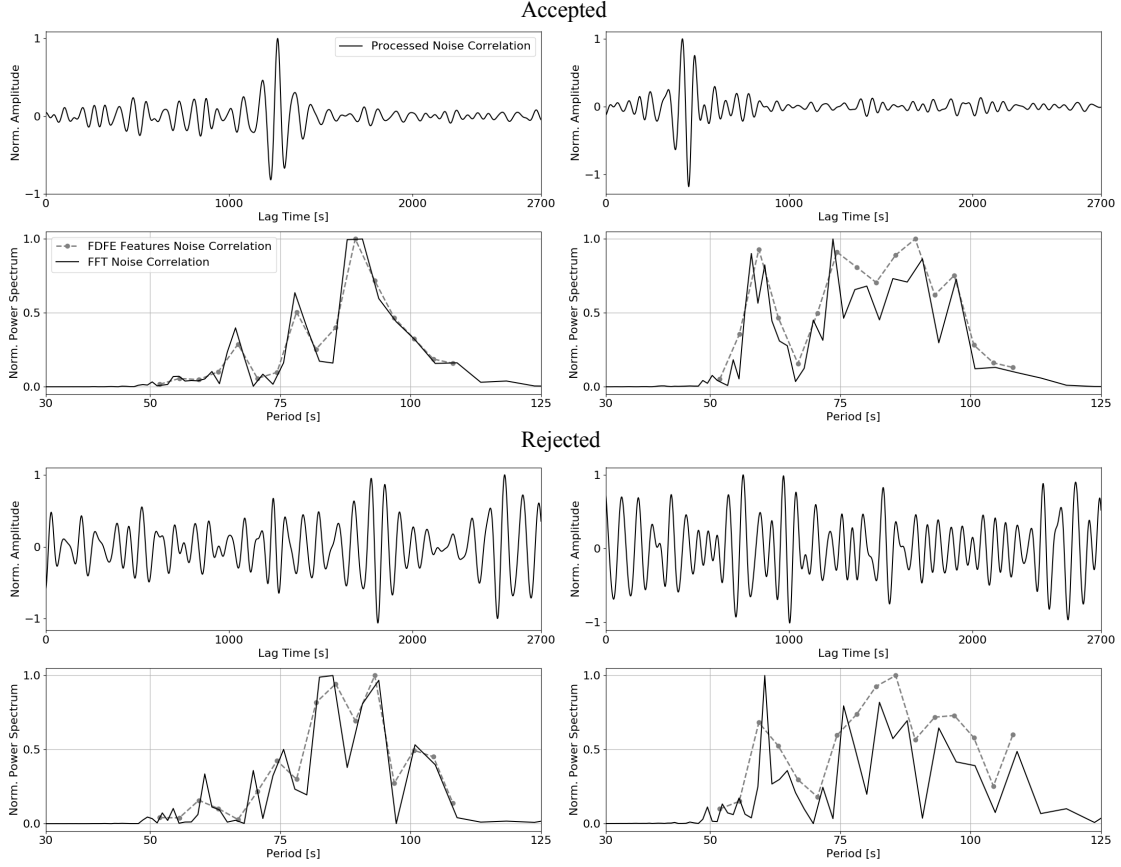


Figure 2: Example for the FDFE for two accepted (top) and two rejected noise correlations (bottom). The upper picture shows the processed noise correlation and the lower picture shows the power spectrum (black line) as well as the extracted FDFE features (grey dots, connected with dashed grey line).

The problem with the FDFE is that the spectra of the accepted and rejected noise correlations are too similar for an accurate automated classification - for this application example. Two examples are displayed in Fig. 2. The spectra at the bottom (rejected) do not strongly deviate from the spectra of the top (accepted). This is the reason why we developed the WBFE algorithm to improve the classification accuracy.

3.2 Configuration

Similar to the tool for classification you also have a configuration file which can be opened via:

```
$ cd /your/path/Ann/FUNCTIONS/PREP
$ open mydetails.py
```

where you can specify your project, paths and filenames again:

```
project = 'example'
name    = 'My_Name'
fname   = 'example.npy'
path_waveforms = '/your/path/to/seismograms/*.SAC'
```

The reason why there is another configuration file is that it makes it possible to work in various projects that are at different states at the same time.

3.3 Using the tool

We can run the feature extractor by running the file *fe.py* (for feature extractor):

```
$ python fe.py
```

This results in a small dialog where you can set various parameters:

```
Running the feature extraction. Please be sure that your traces have the attributes
    trace.data and trace.stats.dist.
How many iterations do you want to perform?
>> 2
Lower Period limit for Bandpass Filter [s]
>> 50
Upper Period limit for Bandpass Filter [s]
>> 100
Approx. surface wave velocity [m/s]
>> 3800
Extracting FT features...
Extracting WBF features...
This may take a while
done
```

The extracted features for the WBF and the FDF are saved in:

```
/your/path/Ann/FILES/features/example
/your/path/Ann/FILES/features/example/ft
```

respectively.

Important note: It is important that the '.SAC' seismic traces have the attribute 'trace.stats.dist' because this information is required in the feature extraction process. Furthermore an approximate surface wave velocity needs to be specified. This can be obtained by plotting the section with obspy and estimating a velocity from the plot. For a plotting tutorial with obspy, see https://docs.obspy.org/tutorial/code_snippets/waveform_plotting_tutorial.html.

A plot like Figure 3 of the paper can be created as well to visualize the extracted features. The first step is to copy the function *Ann/FILES/plots/fplt.py* to the project plotting directory (in your case *Ann/FILES/plots/example/fplt.py*):

```
$ cp /your/path/Ann/FILES/plots/fplt.py \
/your/path/Ann/FILES/plots/example
```

This ensures that you can have a separate plotting function for every project. You then must prepare the data you want to plot. It is currently only possible to plot one seismogram with two iterations of the WBF in this process. You prepare the data with:

```
$ cd /your/path/Ann/FUNCTIONS/PREP
$ python pfplt.py
```

Where you are asked which seismogram you want to plot (and also for which project - this would be 'example' in our example). After this has been executed correctly you can run the actual plotting function:

```
$ cd /your/path/Ann/FILES/plots/example
$ python fplt.py
```

The result (in your example project, two iterations of the WBFE and seismogram number 30) is depicted in Figure 3

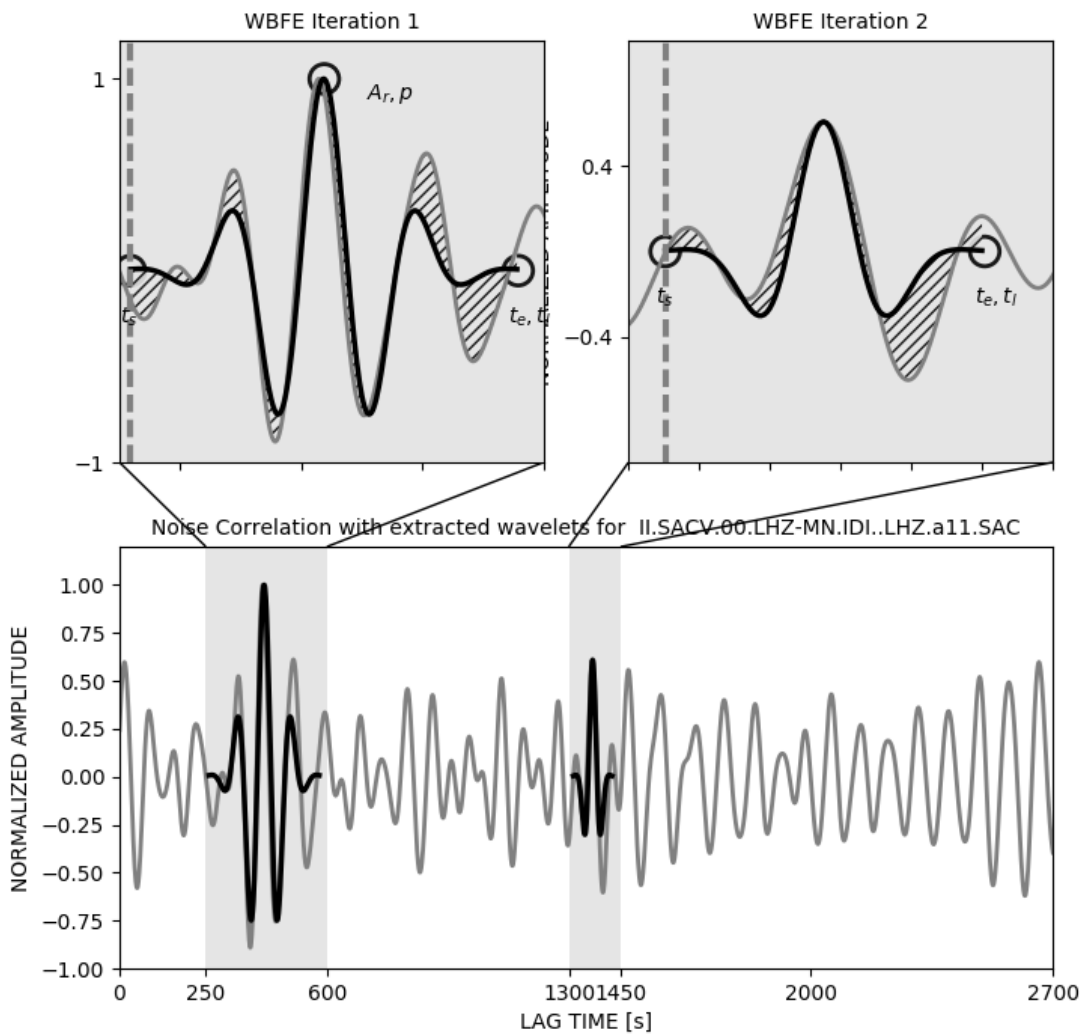


Figure 3: Example output plot of the *fplt.py* function (2 WBFE iterations, example dataset, seismogram 30).

Note: The windows for the top subplots can be adjusted within the *FILES/plots/example/fplt.py* file in lines 156 to 161. If the first extracted feature is at a later lag time than the second one, the variable *swap_subplots* can be changed (valid values are *True* and *False*).

```

x0 = 1300 #Left bound of first WBF iteration subplot
x1 = 1450 #Right bound of first WBF iteration subplot

x2 = 250 #Left bound of second WBF iteration subplot
x3 = 600 #Right bound of second WBF iteration subplot

swap_subplots = True

```

A plot for the visualization of the FDFE (like the ones in Fig. 2) can be created in a similar way.

```

$ cp /your/path/Ann/FILES/plots/fd_fplt.py \
/your/path/Ann/FILES/plots/example/ft

```

Then we prepare the plot:

```

$ cd /your/path/Ann/FILES/plots/example
$ python fd_fplt.py

```

and follow the same instructions as above. You are again asked which seismogram you want to plot (and also for which project - this would be 'example' in our example). After this has been executed correctly you can run the actual plotting function:

```

$ cd /your/path/Ann/FILES/plots/example/ft
$ python fd_fplt.py

```

The output is depicted in Fig. 4

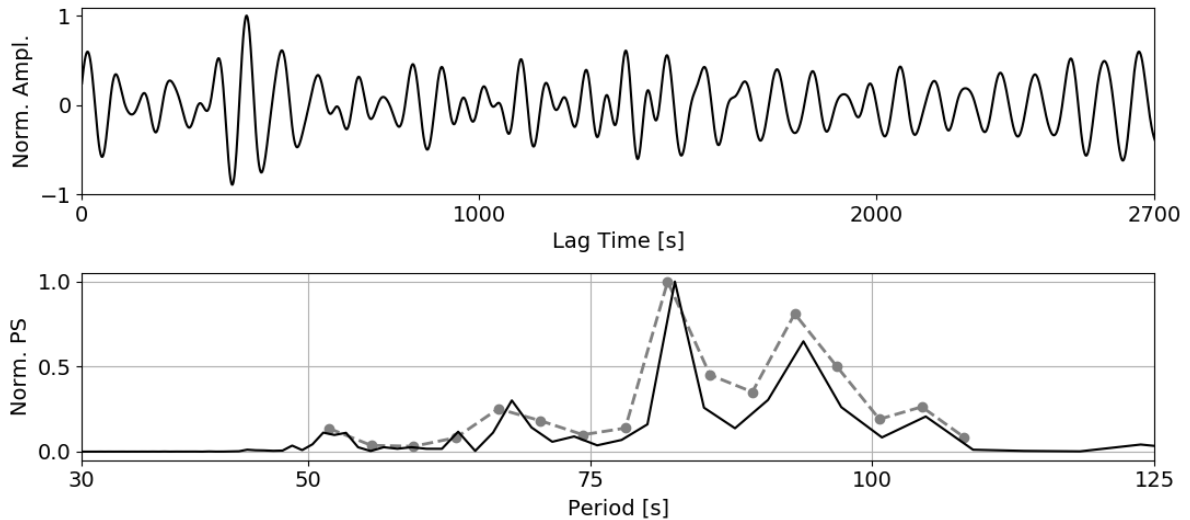


Figure 4: Example output plot of the *fd_fplt.py* function (2 FDFE iterations, example dataset, seismogram 30).

4 Tool for subset creation

To check the performance of the artificial neural network you have to create a training and a testing subset from your dataset. For this specific task you have a python file called *sc.py* (for subset creator). The configuration (*mydetails.py*) is the same as for the feature extractor and can be opened via:

```
$ cd /your/path/Ann/FUNCTIONS/PREP
$ open mydetails.py
```

To create a subset you can run:

```
$ python sc.py
```

The number of samples in the training subset can be chosen here. The specified amount of samples will be taken randomly for each category ('good' and 'bad') from the already classified data saved to your classification file (*fname* in *mydetails.py*). An example is listed here:

```
How many "good" samples in the training subset?
>> 8
How many "bad" samples in the training subset?
>> 12
Choose a rank (single digit integer) as identifier of your subset.
>> 0
Creating SUBSETS ! .... PLEASE WAIT....
Rank:    0      Training size:      8  (g) 12  (b)
Training:    #good    8
Testing :    #good   10
Training:    #bad    12
Testing :    #bad    13
```

The subsets are then saved into you project directory in (where *r* is your specified rank, a single digit integer):

```
/your/path/Ann/FILES/subsets/example/r/
```

specifically in your case:

```
/your/path/Ann/FILES/subsets/example/0/Test :
*good.npy  *bad.npy

/your/path/Ann/FILES/subsets/example/0/Training :
*good.npy  *.npy
```

With the created subsets you now can train and test your artificial neural network. This is described in the next step.

5 Tool for seismogram classification with artificial neural networks

5.1 Training and testing the artificial neural network

This tool again has its own configuration file for the previously mentioned reasons. It can be viewed and changed via

```
$ cd /your/path/Ann/FUNCTIONS/ANN
$ open mydetails.py
```

Some more configurations can be done in *an.py* (for artificial neural network):

```
$ open an.py
```

In lines 21-83 some parameters can be changed. This will be explained further.

```
# ----- USER INPUTS -----

subset = 0          # Training Rank (str)
n = [1]             # Number of repetitions
filelist = ['example.npy'] # Classification files
name_id = 'example_id' # Output file ID
qlist = [1.3]       # d0 Values to test
nrange = [2001]     # Max. training iterations
maxerrs = [0.0001]  # Max. training error to stop
repeat = 2          # No. of trained networks

hidlay1 = [20]      # No. of nodes in the first
                   # hidden layer
hidlay2 = [60]      # No. of nodes in the second
                   # hidden layer

convergence = False # Save errors every iteration?
                  # if yes: provide Path and filename

mode = 'all'        # Allowed Modes for the feature
                   # extraction. Allowed are:
                   # 'WBFE', 'FDFE' and 'all'

# -----
```

- *subset* Which training and testing dataset should be used. The subsets are specified and created with the tool *FUNCTIONS/PREP/sc.py*
- *n* : Number of networks that should be used for the testing runs.
- *filelist* : List of classification files (from *CLASSTOOL/rc.py*). If more than one file is used, the ANN will first do the training and testing with one file and then with the other file. This can be used to chain jobs within one file.
- *name_id* : Filename extension for the output file. This may help you localise and identify your results later.
- *qlist* : List of classification threshold values. If it is set to 1.3 it will go from 0.6 to 1.3 in steps of 0.1.
- *nrange* : List for maximum training iterations (epochs).
- *maxerrs* : Threshold values for the mean training error. If the training misfit is below this threshold, the training will stop.
- *repeat* : How often do you want to repeat the full training / testing process. This saves a trained network for every repetition.

- *hidlay1* : Number of nodes in the first hidden layer of the Ann.
- *hidlay2* : Number of nodes in the second hidden layer of the Ann.
- *convergence* : Will be explained later
- *mode* : The method of the feature extraction you want to train and test your artificial neural network with. This can be either wavelet based feature extraction ('WBFEE'), frequency domain feature extraction ('FDFE') or both ('all'). In the latter case the network will be first trained and tested with the WBFEE and then another Ann will be trained and tested with the FDFE. The results then can be compared. The default setting here is the WBFEE.

The parameters of the ANN itself can be changed in :

```
$ open /control/parameters.py
```

```
learning_rate_hid = 0.5
learning_rate_out = 0.2
tau_hid = 2000.
tau_out = 500.
eps = 0.01
```

These default parameters are adapted Valentine and Woodhouse (2010). After defining your network and training parameters, you can run the training and testing with:

```
$ python an.py
```

We should get an output like this:

```
-----
Running the script for the following configuration:

Output keys      1
subset size      [1]
Classifile       ['example.npy']
Qualitycri       [1.3]
Train Test       [['/example/', '/example/']]
Create subs      False
iterations       [201]
maxerrors        [1e-06]
addoptions       None
-----

=====NEW TRAINING AND TESTING SEQUENCE=====
The ANN control parameters are:
lrate_hid= 0.5
lrate_out= 0.2
tau_hid = 2000.0
tau_out = 500.0
eps = 0.01
+-----start-TRAINING-----+
-> inner_iteration: 0 | max. Error: 6.735e-01 | mean Error: 2.159e-01 | P(good):
    0.47 |
-> inner_iteration: 200 | max. Error: 9.517e-01 | mean Error: 1.405e-01 | P(good):
    0.47 |
Saving trained Network to File:
/Ann/FILES/networks/1/example//example/_extra0rank1exampleexample
+-----end-TRAINING-----+

+-----start-TESTING-----+
...
The testing results can be viewed in the file:
/Ann/RESULTS/exampleexample_id_1FINAL1_i_.csv
+-----end-TESTING-----+
...
```


The results are saved to */RESULTS/filename.csv* in a human readable csv table. The results are structured in table form (example depicted in Fig 5):

nnconfig	fname	qcr	maxerr	type	train	test	iter_t	iter	e_tr	sample	ngood	nbad	err(%)g	err(%)b	err(%)t
[20, 60]	example	1.300e+00	1.000e-04	WBFE	/example/	/example/	2001	2000	3.825e-03	20	9	13	44.44	0.00	18.18
[20, 60]	example	1.300e+00	1.000e-04	FDFE	/example/	/example/	2001	2000	6.347e-04	20	9	13	55.56	38.46	45.45

Figure 5: Example csv output (RESULTS).

The meaning of the single columns is explained in Tab. 2.

Table 2: CSV output columns of the ANN classification.

Result file output columns	
nnconfig	Configuration of the ANN (Hidden Layer Nodes)
fname	Project name
qcr	d0 Value
maxerr	Maximum training error
type	Type of the Feature Extraction (WBFE or FDFE)
train & test	Training and testing projects
iter_t	Maximum Training iterations +1 (max. epochs+1)
iter	Performed training iterations (epochs)
e_tr	Training error
sample	Number of training samples (total)
ngood	Number of 'good' testing samples
nbad	Number of 'bad' testing samples
err(%)g	Percent of misclassified 'good' samples (good samples classified as bad)
err(%)b	Percent of misclassified 'bad' samples (bad samples classified as good)
err(%)t	Percentage of total misclassified samples

In the column type you will also see something like **SUM.WBFEn**. This means that it is the classification based on n separately trained networks with combined output for the WBFE (or FDFE) type. It can happen that the number of training and testing samples slightly deviates from your specified subset sizes. This can be due to errors happening within the feature extraction. If the peak amplitudes are too close to the edges of the noise correlation (very early or late lag times), the WBFE currently fails (at the current development phase).

If you are happy with the training and testing results you can continue with the trained network and start extracting data out of your initial dataset by using your trained networks.

5.2 Extracting specific data out of your initial dataset

After training the Ann and saving the neuron weights, you can use it to select data with your wanted property (the data that you classify as 'good'). You need to specify some properties and paths to do so. You therefore open the file

```
$ cd /your/path/Ann/FUNCTIONS/ANN/
$ open gd.py
```

In lines 13 to 19 you have the parameters:

```
path_features = 'FILES/features/example/*.npy'
```

```

path_wave      = '/your/path/to/seismograms/*.SAC'
path_wave0     = '/your/path/to/seismograms/'
path_out       = '/RESULTS/waveforms/example/'
networks       = '/FILES/networks/0/example/example/*extr*.npz'
threshold      = 1.0

```

To obtain the seismograms with the wanted property (output > threshold) you can run:

```
$ python gd.py
```

The seismograms from the *path_wave* will be saved to *path_out*. The classification is based on the trained neural networks specified in *networks*. The output may then look like that:

```
n 'good' files copied to:      /RESULTS/waveforms/example/
```

(where n is the number of copied files) NOTE: If no files are copied, you may try to decrease the threshold variable in the *gd.py* file.

5.3 Convergence and parameter tests

To help you getting an understanding on the behaviour of the Ann on different training parameters, you can look at convergence plots. A convergence plot shows the training misfit in dependence of the number of iterations. Ideally the misfit will converge towards its global minimum - that is dependent on which misfit function you choose, how the training parameters are set up and especially dependent on the data and its visual classification. To plot the training misfit for every iteration, you want to save that data. You can do this by going back to the file:

```
$ cd /your/path/Ann/FUNCTIONS/ANN/
$ open an.py
```

and changing the *convergence* parameter to a *.csv* file with your wanted location. In your example:

```
convergence = '/RESULTS/convergence/example/convergence_test.csv'
```

Important note: If you do not change the name of the output file id in *an.py* your networks will be overwritten. So please go again through the full process of setting new names as identifiers if you want to prevent that. This will help you keeping track of your files.

If you now run:

```
$ python an.py
```

you should confirm that you want to save the errors for every iteration. This will create another testing and training sequence.

After running training and testing again, the iterations with their errors are saved to the file that you just specified. If you now open:

```
$ cd /your/path/Ann/RESULTS/convergence/
$ open convergence_plot.py
```

and change the *file* variable (line 10) to the one specified in *an.py*

```
file = '/your/path/RESULTS/convergence/example/convergence_test.csv'
```

you can run

```
$ python convergence_plot.py
```

and you will receive a plot that looks similar to the one in Figure 6.

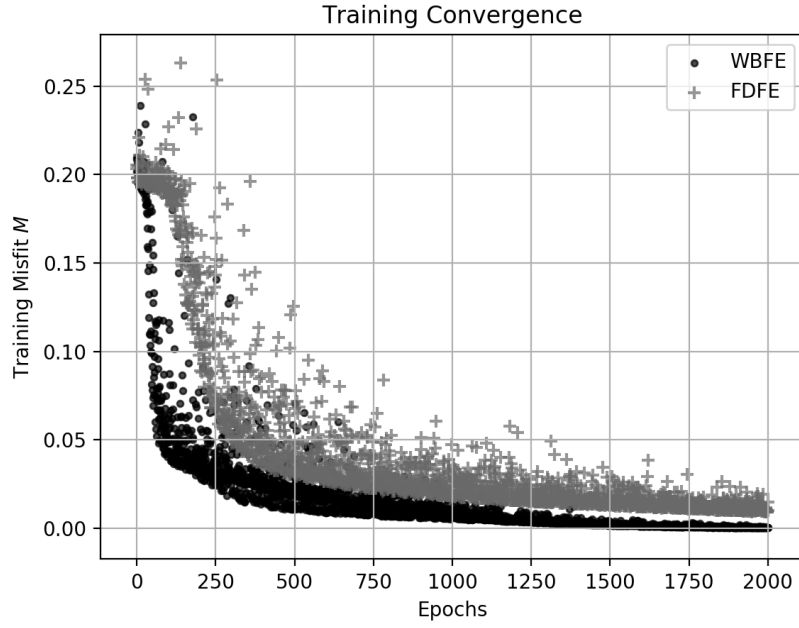


Figure 6: Example convergence plot for an Ann that is trained with WBFE and FDFE data.

Because the training of the Ann starts in a randomized state, the plots will never look exactly the same, but in general the trend should be similar.

Remarks

For suggestions, remarks, issues and questions please do contact the author via:
patrick.paitz@erdw.ethz.ch

References

- Valentine, A. P. and Woodhouse, J.H., Approaches to automated data selection for global seismic tomography, *Geophys. J. Int.* **182**(2), 1001-1012
- Krischer, L., Megies, T., Barsch, R., Beyreuther, M., Lecocq, T., Caudron, C., Wassermann, J., ObsPy: A bridge for seismology into the scientific Python ecosystem, *Computational Science & Discovery* **8**(1), 014003
- Paitz, P., Artificial neural networks for automated classification of ambient noise correlation data, *Master Thesis*, ETH Zürich