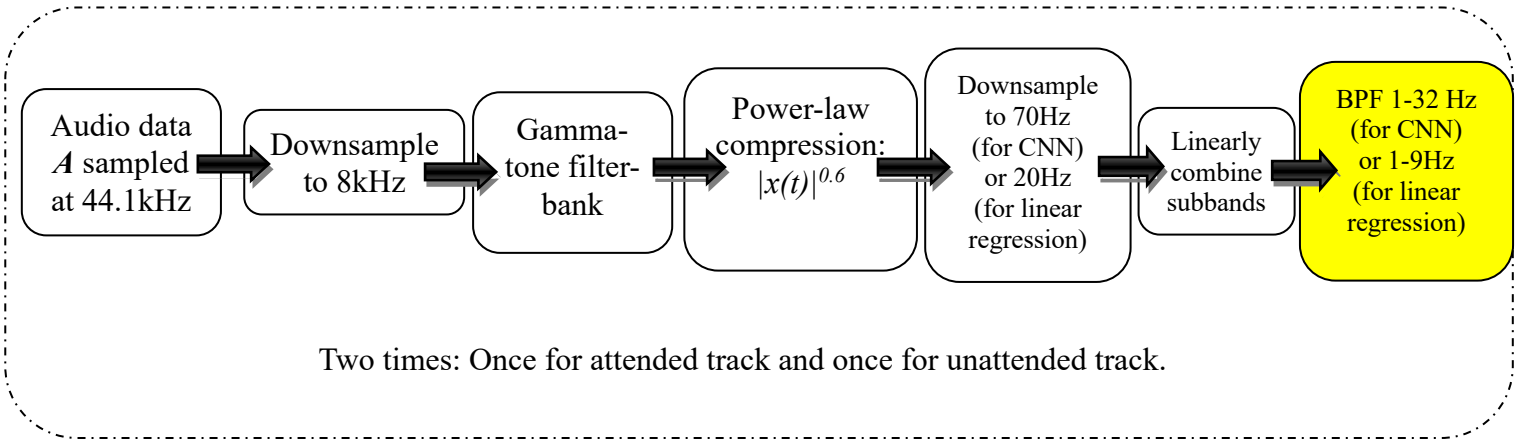
Data Preprocessing for Auditory Attention Decoding

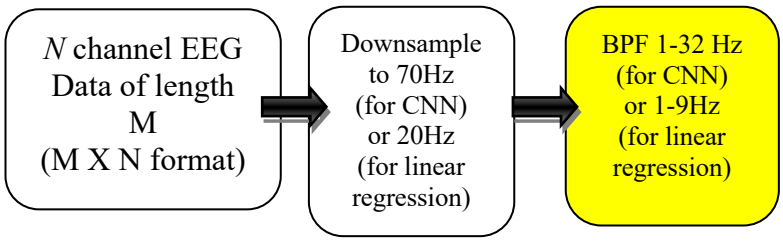
Data preprocessing is an essential step in any signal processing project. Based on domain knowledge, the relevant part of the data can be selected and irrelevant parts (e.g. due to a too high signal-to-noise ratio) can be filtered out. Below, we describe a series of preprocessing steps for the audio and EEG recordings that make them more suited as input for the CNN or linear regression analysis.

Rather than blindly applying these preprocessing steps, can you understand why they are introduced?

For the 64 channel data, we already provide you preprocessed data (at 70Hz), both for the EEG recordings as well as for the audio. For the 24 channel data and for your own recordings, you will have to do the preprocessing yourself, following the description below. Note that in that case, it's best to also do the audio preprocessing again, to make sure the two are consistent.

Note that, after the preprocessing step, you still need one extra step before you can input the data into the linear regression or CNN modules: the continuous stream of data needs to be divided in small chunks ('decision windows'), the length of which is an important parameter to experiment with (say between 1 sec and 1 minute).





Preprocessing Steps

Audio Data

Required files/tools:

1. Audio WAV files (the *dry* files from Table I; **NOT** HRTF filtered)
2. Preprocessing constants provided in *audioprepro_constants.mat*

Step 1: Read audiofile *filename.wav* to the variable **A** (use 'audioread' in Matlab). The audio is sampled at 44.1kHz.

Step 2: Resample the audio data in **A** to 8kHz.

Step 3: Decompose the audio in 15 frequency subbands using a gamma-tone filter bank, which models the frequency analysis in the cochlea of a human listener. The FIR impulse responses of the 15 gamma-tone filters are provided in the *audioprepro_constants.mat file* as the variable **g**. Filter the speech signal with each of these filters, to generate 15 subband signals.

Step 4: Perform power-law compression on each of the 15 subband signals, i.e., $|x(t)|^{0.6}$where $x(t)$ is the subband signal at the output of one of the filters in the gammatone filterbank.

Step 5: Downsample the resulting signals to

- 20Hz for linear regression (only using frequencies <10Hz)
- 70Hz for CNN (only using frequencies <35Hz)

Step 6: Linearly combine the bandpass filtered envelopes to a single envelope. Common practice is to use uniform weights (i.e. summing the subbands).

Step 7: Filter the audio using a bandpass filter (BPF):

- For linear regression: filter between 1-9Hz
- For CNN: filter between 1-32Hz

IMPORTANT: Use the exact same filtering operation(s) as in the bandpass filtering on the EEG data (see yellow blocks in the block diagram), such that the filter delays etc. are exactly the same.

Note: The linear regression and CNN-based decoding use different frequency ranges, which is reflected in the different sampling rates and BPFs. These are taken from the literature. To make your life easier and to avoid confusion in terms of differences in filter/kernel lengths, you could opt to use the same sampling rate / BPF for both cases, but take into account that this may (slightly) affect performance. At a later stage, you could then investigate the optimal frequency range for both.

EEG Data

64-channel Biosemi EEG data

This data is already in the correct pre-processed format and no pre-processing is necessary.

Data structure: The data is divided in multiple .mat files for 16 subjects (each in a specific folder). Each mat file in a subject-folder is named *trial XXX.mat*. Instructions:

a) Load *trial XXX.mat*
b) The EEG data and other meta-information are loaded in a structure called **trial.**
c) Important fields in **trial:**
    a. EEG data: **trial.RawData.EegData (Time samples X Number of channels)**
    b. Sampling rate: **trial.FileHeader.SampleRate (Hz)**
    c. The two audio stimuli played during the experiment: **trial.stimuli**
        i. **trial.stimuli{1}** – filename of audio in left-ear
        ii. **trial.stimuli{2}** – filename of audio in right-ear

d. Attention information of a particular file: **trial.attended_track**.
    i. trial.attended_track =1 → trial.stimuli{1} is the attended track
    ii. trial.attended_track =2 → trial.stimuli{2} is the attended track

24-channel Smarting data

Required files/tools:

1. 24-channel mBrainTrain-Smarting recordings.
2. EEGLAB toolbox (https://sccn.ucsd.edu/eeglab/index.php).
3. Biosig extension (http://biosig.sourceforge.net/).
   NOTE: Instructions to add the above two toolboxes to the MATLAB path are provided in the respective websites.
4. **eegread.m** : function to read EEG data from the experiment.

Step 1: Read the EEG data: The EEG data recorded by the mBrainTrain-Smarting device is stored in a format called GDF. In MATLAB, this format can be read using the EEGLAB toolbox by the help of the Biosig extension. EEGLAB is an interactive Matlab toolbox for processing continuous and event-related EEG.

Use the *pop_biosig* function from the Biosig extension of the EEGLAB toolbox to read the data into a variable **EEG_in**. This function returns an EEGLAB dataset structure in **EEG_in**. An EEGLAB dataset structure has various information in its fields with respect to the EEG data, for example

- **EEG_in.data** : The EEG data itself. NOTE: Here, the format is **(Number of channels X Time samples).**
- **EEG_in.srate**: The sampling rate in Hz of the EEG data.
- **EEG_in.events**: A field containing all the recording events and their time instances; like start of the recording, end of recording, etc.

There are other fields containing channel names, select signal-processing analyses applied to the EEG etc. However, we will mostly be working with the above 3 fields.

1) Extracting the EEG: The EEG recorded is marked with *events* which indicate start and end of the experiment. A function **eegread** is provided which helps extract the relevant EEG data from the recording. This function by default extracts the EEG between the start and end of the experiment (event IDs 33027 and 33024, respectively).
2) For the attention switching experiment, you have to extract the events corresponding to the attention switches. To this end, read the code of **eegread** and understand how *events* in an experiment are encoded in the EEG dataset structure. Write your own code to extract the switching events defined by the following two IDs:
    a) Switch to left: 33028
    b) Switch to right: 33029

Step 2: Downsample the EEG signals to

- 20Hz for linear regression (only using frequencies <10Hz)
- 70Hz for CNN (only using frequencies <35Hz)

Step 3: Filter the EEG signals using a bandpass filter (BPF):

- For linear regression: filter between 1-9Hz
- For CNN: filter between 1-32Hz

IMPORTANT: Use the exact same filtering operation(s) as in the bandpass filtering on the audio data (see yellow blocks in the block diagram), such that the filter delays etc. are exactly the same.