ScatNet Implementation Document

June 6, 2013

1 Introduction

2 The Scattering Transform

The scattering transform is implemented in ScatNet using the `scat` function. This function can be used to calculate several types of scattering transforms by varying the linear operators supplied to it.

2.1 The `scat` Function

The `scat` function takes as input a signal `x` and a set of linear operators `Wop` and outputs a scattering transform `S` and intermediate modulus coefficients `U`. A call to the `scat` thus looks like

```
[S, U] = scat(x, Wop);
```

Often, the modulus coefficients `U` are not necessary and so can be left out.

Outputs `S` and `U` are cell arrays, each element corresponding to a layer of the scattering transform. The format of these layers is described in the next subsection.

Each element of the `Wop` is a function handle, with signature

```
[A, V] = Wop{m+1}(X);
```

Note that operators are indexed starting at $m = 0$, so an offset of 1 is necessary for compatibility with MATLAB. Here, `X`, `A` and `V` are all in the network layer format described in the next subsection. The linear operator `Wop{m+1}` transforms the signals in the `X` into two new layers: an invariant layer `A` (average) and a covariant layer `V` (variations). In the case of a wavelet transform, `A` corresponds to the averaging of the lowpass filter ϕ while `V` consists of the wavelet coefficients obtained by convolving with ψ_j.

Initializing `U1` using the input signal, the following loop is then executed in `scat`

```
for m = 0:numel(Wop)-1
    if (m < numel(Wop)-1)
        [S{m+1}, V] = Wop{m+1}(U{m+1});
        U{m+2} = modulus_layer(V);
    else
        S{m+1} = Wop{m+1}(U{m+1});
    end
end
```

For each intermediate layer `U{m+1}`, we thus apply the linear operator `Wop{m+1}`, assigning the invariant output to the mth-order scattering layer `S{m+1}`, and computing the modulus of the covariant part to obtain the $(m + 1)$th order intermediate coefficients `+2Um`. For the last layer, we do not need the intermediate coefficients, and so only compute the scattering coefficients.

2.2 Network Layers `S{m+1}` and Linear Operators `Wop{m+1}`

As mentioned earlier, each element of `S` and `U` are in the network layer format. These consist of two fields, `signal` and `meta`. The former is a cell array of signals and the latter is a structure containing information related to each signal.

Specifically, each of the fields in `meta` is an array with the same number of columns as the length of `signal`. For example, the `meta.resolution` field has one row, indicating the extent the signal has been subsampled with respect to the original length. In the case of wavelet transforms used as linear operators, an important field is `meta.j`, which has a variable number of rows, describing the scales of the wavelets used to compute the coefficient. For a first-order coefficient $|x \star \psi_{j_1}| \star \phi(t)$, this will simply be one row containing j_1. For a second-order coefficient $||x \star \psi_{j_1}| \star \psi_{j_2}| \star \phi(t)$, the first row will contain j_1 while the second will contain j_2.

Let us consider the example output `S` from the `scat` function using wavelet transforms as linear operators. In this case, `S{m+1}` will contain the mth-order scattering coefficients, with `S{m+1}.signal{p}` being the pth signal among these. Its scales $(j_1, j_2, \ldots, j_m)$ are specified in `S{m+1}.meta.j(:,p)`.

As mentioned earlier, the `Wop{m+1}` function handles take as an input a network layer and outputs two layers, one invariant and one covariant. Any function with these inputs and outputs can be used as an element of `Wop`. Two basic functions can be used for this purpose: `wavelet_layer_1d` and `wavelet_layer_2d`, which define wavelet transforms on network layers. For example, supposing we have defined a filter bank `filters` (see next section), we can create an accompanying 1D wavelet transform by defining

```
Wop{m+1} = @(U)(wavelet_layer_1d(U, ...
    filters));
```

This has to be done for each layer of the scattering transform.

3 Creating Operators

Linear operators can be constructed manually as indicated in the previous section by defining filters and function handles. However, factory functions presented in this section simplify this process.

3.1 `wavelet_factory_1d`

The `wavelet_factory_1d` function takes a signal size along with a number of parameters and returns a cell array of linear operators corresponding to wavelet transforms. Its signature is given by

```
[Wop, filters] = wavelet_factory_1d(N, ...
    filt_opt, scat_opt, M);
```

where `N` is the size of the signal, `filt_opt` are the filter parameters, `scat_opt` are other parameters for the wavelet transforms, and `M` is the maximal order of the scattering transform (i.e. how many layers/-linear operators to create). The function outputs the wavelet transforms `Wop` and the filters used to define them `filters`. To calculate the scattering, only the former are necessary.

Default filter options are obtained by calling `default_filter_options`, which takes as input a type of filter and an averaging scale. The type of filter is one of `'audio'`, `'dyadic'` or `'image'`, corresponding to audio signals, other (less oscillatory) 1D signals, and images, respectively. The averaging scale determines the size of the largest wavelet and consequently that of the lowpass filter ϕ. For more filter options, see next section.

The `scat_opt` supports the following options:

- `scat_opt.antialiasing`: The extent to which the final scattering output is over-sampled. Specifically, the sampling rate is `2^scat_opt.antialiasing` times the critical sampling rate. By default equal to `1`.

To specify default values, `scat_opt` can be left empty.

Using the above information, we can create a second-order scattering transform for an audio signal of length 65536 with an averaging scale of 4096

```
filt_opt = ...
    default_filter_options('audio', 4096);
scat_opt = struct();

Wop = wavelet_factory_1d(65536, ...
    filt_opt, scat_opt, 2);
```

The scattering transform of a signal `x` is then obtained by calling `scat(x, Wop)`.

3.2 `wavelet_factory_2d`

LAURENT

4 Defining Filters

As described in the previous section, filter parameters are specified in the `filt_opt` structure passed to `wavelet_factory_1d` or `wavelet_factory_2d`. Depending on the type of filter used, this structure can contain different parameters, but the following can always be specified:

- `filt_opt.filter_type`: The wavelet type, such as 'morlet_1d', 'morlet_2d', 'spline_1d', for example (default `'morlet_1d'` for 1D signals, `'morlet_2d'` for 2D signals).
- `filt_opt.precision`: The numeric precision of the filters. Either `'double'` or `'single'` (default `'double'`).

Since the `filter_type` field determines the type of filter bank to create, it determines what other options can be specified. The rest of this section covers the different types of filters and their parameters.

For 1D signals, it is often useful to specify different filter parameters for different orders. This is done by using arrays instead of scalar values for parameters in `filt_opt`. In the case of numeric parameters, this means that they are regular arrays, whereas for string parameters, such as `filter_type`, they are cell arrays.

For example, setting `filt_opt.filter_type ... = {'gabor_1d','spline_1d'}`, means that the first-order filter bank should be a Gabor filter bank whereas the second should be a spline wavelet filter bank. If more filter banks are needed than parameters are available in `filt_opt`, the last element in each array is repeated as necessary.

4.1 1D Morlet/Gabor Filters

The Gabor wavelets consist of Gaussian envelopes modulated by complex exponentials to cover the entire frequency spectrum. Morlet wavelets are derived from these by subtracting the envelope multiplied by a constant such that the integral of the filter equals zero. As they are closely related, the Gabor and Morlet wavelet have identical sets of options.

These are obtained by setting `filt_opt.filter_type` to `'morlet_1d'` or `'gabor_1d'`, respectively. The Morlet filters are the default filters for 1D signals.

The Morlet/Gabor filter bank has the following options:

- `filt_opt.Q`: The number of wavelets per octave. By default 1.
- `filt_opt.J`: The number of wavelet scales.
- `filt_opt.B`: The reciprocal octave bandwidth of the wavelets. By default `filt_opt.Q`.
- `filt_opt.sigma_psi`: The standard deviation of the mother wavelet in space. By default calculated from `filt_opt.B`.
- `filt_opt.sigma_phi`: The standard deviation of the scaling function in space. By default calculated from `filt_opt.B`.

The maximal wavelet bandwidth (in space) is determined by $2^{J/Q}$ times the bandwidth of the mother wavelet, which is proportional to `sigma_psi`. If `sigma_psi` is smaller than a certain threshold, a number of constant-bandwidth filters are added, linearly spaced, to cover the low frequencies.

Again, we can specify different filter banks by setting `filt_opt.Q` and `filt_opt.J`, etc. to arrays instead of scalars. This is often useful if the nature of the signal is different at different orders, which is usually the case in audio. For this reason, the `default_filter_options` for `'audio'` specify `filt_opt.Q = [8 1]`, whereas for `'dyadic'`, `filt_opt.Q` is set to `1`. A higher frequency resolution is needed in audio for the first-order filter banks due to the highly oscillatory structure of the signals.

To calculate the appropriate `J` necessary for a given window size `T` and filter parameters `filt_opt`, there is the conversion function `T_to_J`, which is called using:

```
filt_opt.J = T_to_J(T, filt_opt);
```

This will ensure that the maximum wavelet scale, and thus the averaging scale of the lowpass filter, will be approximately `T`.

4.2 1D Spline Filters

The spline wavelet filter bank is an unitary filter bank as defined in [?]. As a result, the scattering transform defined using these filters has perfect energy conservation [?].

In addition to the parameters listed above, the spline filter bank has the following options:

- `filt_opt.J`: The number of wavelet scales.
- `filt_opt.spline_order`: The order of the splines. Only linear (spline order 1) and cubic (spline order 3) are supported.

The maximal bandwidth is specified here by 2^J.

4.3 2D Morlet/Gabor Filters

LAURENT

5 Manipulating, Formatting Scattering Coefficients

To facilitate the manipulation of the scattering coefficients, several functions are provided that perform different operations on the network layer format described above. In addition, we can convert the coefficients to an easier-to-manage array representation, concatenating scattering coefficients and separating out meta-data.

5.1 Post-processing

This section describes the different post-processing function that are available to apply to scattering coefficients.

5.1.1 `renorm_scat`

As described in [?], second-order scattering coefficients can be renormalized by dividing $Sx(t, j_1, j_2)$ by $Sx(t, j_1)$, which decorrelates the second order from the first. A similar procedure is available for higher orders.

In ScatNet, this transformation is available through the `renorm_scat` function. Its signature is as follows

```
S_renorm = renorm_scat(S, epsilon);
```

where `S` is the scattering transform to be renormalized, and `epsilon` is a regularization constant that prevents the fraction from diverging when the first order becomes zero.

5.1.2 `log_scat`

For several signals, it is sometimes necessary to compute the logarithm of scattering coefficients. This is achieved in ScatNet using the `log_scat` function. Its signature is

```
S_log = log_scat(S, epsilon);
```

where `S` is the scattering transform whose logarithm we want to compute, and `epsilon` is a regularization constant that prevents the logarithm from going to negative infinity when the scattering coefficients approach zero.

5.1.3 `average_scat`

5.2 Formatting

In order to use the scattering coefficients for classification and other tasks, it is often useful to convert the representation into a purely numeric format, separating out the metadata. This is the role of the `format_scat` function, whose signature is

```
[S_table, meta] = format_scat(S, fmt);
```

where `S` is the scattering transform, `fmt` is the type of output we want, `S_table` is the formatted output, and `meta` contain the metadata from `S` associated with the signals in `S_table`.

There are three different formats available for `fmt`. The first is `'raw'`, which leaves the representation intact, setting `S_table = S`. The second is `'order_table'`, which creates a two-dimensional table for each scattering order, and sets `S_table` to the cell array of these tables, with `meta` being a cell array of the associated `S{m+1}.meta`. Finally, `'table'` (the default), concatenates the tables of all orders and does the same for the meta fields. In the case where `meta` fields have different number of rows for different orders, the smaller fields are extended with `-1`. Note that this last format is only possible if scattering coefficients of different orders are of the same resolution, which is most often the case.

6 Display

6.1 1D

6.1.1 scattergram

The `scattergram` function displays the scattering coefficients as a function of time and the last filter index with the remaining filter indices fixed. For first-order coefficients, this means that it is shown as a function of time and j_1. For second-order coefficients, it is shown as a function of time and j_2 for a fixed j_1, and so on.

For one display, a pair consisting of a scattering layer along with a scale prefix is needed. We can specify multiple displays by specifying multiple of these pairs, which will then be displayed one on top of the other.

Its signature is the following

```
img = scattergram(S{m1+1}, jprefix_1, ...
    S{m2+1}, jprefix_2, ...);
```

where `S` is the scattering transform, `m1` is the order of the first display, `jprefix_1` is the first scale prefix, and so on. Note that is `m1 = 1` the layer is of first order so no scale prefix is necessary and we set `jprefix_1` to `[]`.

Instead of a scattering output `S`, we can also specify intermediate modulus coefficients `U`.

6.2 2D

7 Classification

7.1 Batch Computation

7.2 Affine Space Classifier

7.3 Support Vector Classifier