Vision in Human and Machine - Tutorial 3
Sparse Coding for Visual Hierarchy Representations

Heiko Wersing

1 Learning sparse image representations

The experiments in this tutorial are based on the COIL100 image database. There exists a copy of the database at [/vol/lehre/...] We begin with learning of a sparse basis using the direct image pixel input. The first step is the generation of a `dataset` from the PGM image ensemble (compare Tutorial 2).

1.1 Generating a COIL100 input dataset

A call of the function `save_dataset('input')` will generate a dataset from the COIL images, where the main computation is done by a call to `compute_dataset`. Have a look at `save_dataset.m` and `compute_dataset.m` and analyze it. You can visualize the resulting dataset by calling `show_activations('input',12,100)`, where 12 is the number of columns in the plot and 100 is the number of images(feature activations) in the dataset to show.

1.2 Running the sparse coding learning on the images

Now you can run the `sparse_coding.m` function to learn a basis representation from a patch ensemble sampled from the COIL images. Investigate the following questions:

1. How does the representation differ depending on patch size (try 5-28) and number of available basis vectors ?
2. What happens when the number of available objects is reduced ?
3. How does the structure of the basis change with sparsity factor lambda ?

2 Building hierarchies from features

To build a model of the ventral pathway with several feature detection and integration stages, we use representation of a number of n layers, where each layer is composed of m 'feature planes'. Each feature plane is represented as a matrix, and corresponds to the response map of the feature detector on its input. In Matlab, a layer is therefore represented as a (dim x dim x feature_planes) matrix. Since we use the features to transform the input from one layer to another layer, where both can have different numbers of feature planes, we represent a set of filters in a 4-dimensional matrix `filters(x,y,feature_input_planes,feature_index)`.

2.1 Preparing simple and complex cell layers S1,C1

We start from the function `compute_dataset` and build up the hierarchy incrementally by adding more and more layers to the computation routine.

2.1.1 Layer S1

The first processing layer 1 should use Gabor filters at different orientations to obtain orientation-selective responses similar to V1 simple cell neurons. Use the function `get_gabor_filters (rot, RF_siz, Div, Phi,sigma_factor)` to obtain a set of Gabor filters, and use the function `feed_filters` to compute the response of layer 1. Start with the 'Linear' transfer function, and investigate the output using `show_activations`. Try out different Gabor parameters. Then change the transfer function to 'Linear_Abs_Wtm_Thresh' and investigate the role of the competition parameter (p1) and the threshold (p2);

2.1.2 Layer C1

The second layer is used to pool and integrate the simple cells to obtain complex cell response properties. Use the get_gauss to obtain a Gaussian and then apply the feed_filters with the transfer function 'Linear_Tanh' to obtain the response. Use a 'downsample' factor of 2 to reduce the resolution by a factor of 2 in both x and y direction.

2.1.3 Layer S2 - Learned by sparse coding

Investigate the C1 output on the input ensemble, and make sure that you observe a significantly different output pattern for different objects. Otherwise you have to re-adapt the parameters.

For the saved C1 dataset, you can run the sparse_coding program to obtain a sparse coding basis, based on the multidimensional input patches from the dataset (Here it depends on your chosen number of Gabor filters). Again you have to experiment with the parameters to influence the learned basis sets. Since the C1 layer has as many planes as Gabor features, now the receptive fields are looking into more than one plane. In the visualization this is shown by arranging the input planes of a feature horizontally. The learned features can be read in with `get_learned_features` as a filter set. Use the learned features to define the S2 layer, analogously to the S1 layer. Investigate the responses for different WTM competition and thresholds.