Annex

This supplementary information presents :

- first, the code to generate the figures from the paper,
- second, some control experiments that were mentionned in the paper,
- finally, some perspectives for future work inspired by the algorithms presented in the paper.

Figures for "An adaptive algorithm for unsupervised learning"¶

In [1]:

```
%load_ext autoreload
%autoreload 2
```

In [2]:

```
import numpy as np
np.set_printoptions(precision=2, suppress=True)
seed = 42
np.random.seed(seed)
```

In [3]:

```
# some overhead for the formatting of figures
import matplotlib.pyplot as plt

fontsize = 12
FORMATS = ['.pdf', '.eps', '.png', '.tiff']
FORMATS = ['.pdf', '.png']
dpi_export = 600

fig_width_pt = 318.670  # Get this from LaTeX using \showthe\columnwidth
fig_width_pt = 450  # Get this from LaTeX using \showthe\columnwidth
#fig_width_pt = 1024 #221     # Get this from LaTeX using \showthe\columnwidth / x264 asks f
ppi = 72.27 # (constant) definition of the ppi = points per inch
inches_per_pt = 1.0/ppi  # Convert pt to inches
```

```
#inches_per_cm = 1./2.54
fig_width = fig_width_pt*inches_per_pt  # width in inches
grid_fig_width = 2*fig_width
phi = (np.sqrt(5) + 1. ) /2
#legend.fontsize = 8
#fig_width = 9
fig_height = fig_width/phi
figsize = (fig_width, fig_height)


def adjust_spines(ax, spines):
    for loc, spine in ax.spines.items():
        if loc in spines:
            spine.set_position(('outward', 10))  # outward by 10 points
            spine.set_smart_bounds(True)
        else:
            spine.set_color('none')  # don't draw spine

    # turn off ticks where there is no spine
    if 'left' in spines:
        ax.yaxis.set_ticks_position('left')
    else:
        # no yaxis ticks
        ax.yaxis.set_ticks([])

    if 'bottom' in spines:
        ax.xaxis.set_ticks_position('bottom')
    else:
        # no xaxis ticks
        ax.xaxis.set_ticks([])

import matplotlib
pylab_defaults = {
    'font.size': 10,
    'xtick.labelsize':'medium',
    'ytick.labelsize':'medium',
    'text.usetex': False,
#    'font.family' : 'sans-serif',
#    'font.sans-serif' : ['Helvetica'],
    }

#matplotlib.rcParams.update({'font.size': 18, 'font.family': 'STIXGeneral', 'mathtext.fontse
matplotlib.rcParams.update(pylab_defaults)
#matplotlib.rcParams.update({'text.usetex': True})

import matplotlib.cm as cm
```

```
from IPython.display import Image

DEBUG = True
DEBUG = False
hl, hs = 10*'-', 10*' '
```

In [4]:

```
tag = 'ICLR'
datapath = '../../SparseHebbianLearning/database'

# different runs
#opts = dict(eta=0.007, eta_homeo=0.02, alpha_homeo=.5, cache_dir='cache_dir_1100', datapath
#opts = dict(eta=0.007, eta_homeo=0.02, alpha_homeo=.08, cache_dir='cache_dir_1900', datapat
#opts = dict(eta=0.007, eta_homeo=0.005, alpha_homeo=5., cache_dir='cache_dir_1700', datapat
#opts = dict(eta=0.0033, eta_homeo=0.05, alpha_homeo=.5, cache_dir='cache_dir_frioul', datap
#opts = dict(eta=0.007, eta_homeo=0.005, alpha_homeo=5., cache_dir='cache_dir', datapath=dat
opts = dict(eta=0.0033, eta_homeo=0.05, alpha_homeo=2.5, cache_dir='cache_dir_42', datapath=
```

In [5]:

```
from shl_scripts.shl_experiments import SHL
shl = SHL(**opts)
data = shl.get_data(matname=tag)
```

In [6]:

```
print('number of patches, size of patches = ', data.shape)
print('average of patches = ', data.mean(), ' +/- ', data.mean(axis=1).std())
SE = np.sqrt(np.mean(data**2, axis=1))
print('average energy of data = ', SE.mean(), '+/-', SE.std())
```

```
number of patches, size of patches =  (65520, 324)
average of patches =  1.6601716040762679e-19  +/-  0.009395020606115053
average energy of data =  0.30840286225885366 +/- 0.08622704306730453
```

!ls -l {shl.cache_dir}/{tag}* #!rm {shl.cache_dir}/{tag}*lock* #!rm {shl.cache_dir}/{tag}* #!ls -l {shl.cache_dir}/{tag}*

figure 1: Role of homeostasis in learning sparse representations¶

TODO : cross-validate with 10 different learnings¶

In [7]:

```
fname = 'figure_map'
N_cv = 10
one_cv = 0 # picking one to display intermediate results
```

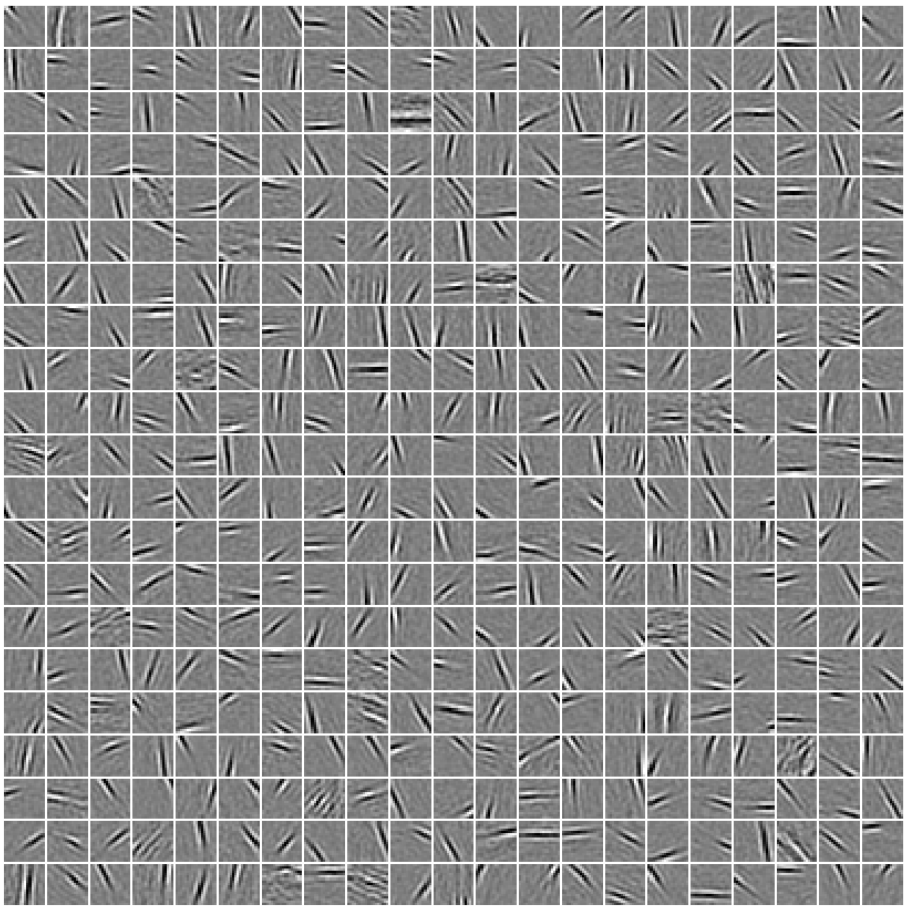
learning¶

The actual learning is done in a second object (here `dico`) from which we can access another set of properties and functions (see the shl_learn.py script):

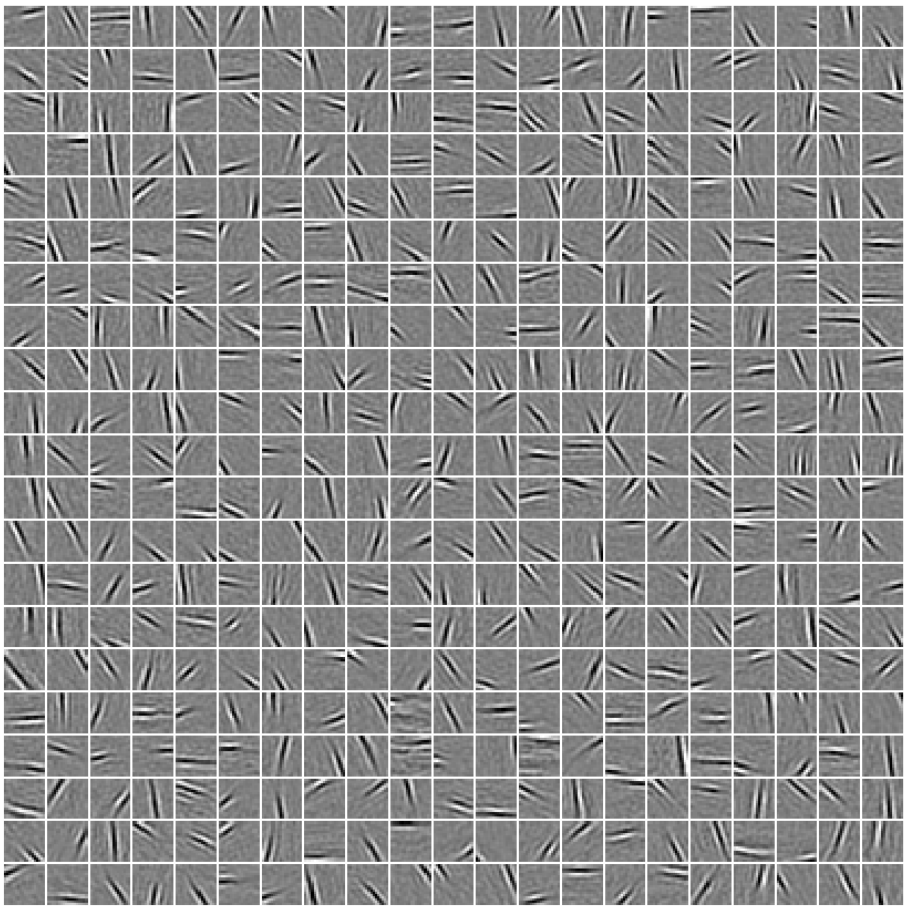
In [8]:

```
homeo_methods = ['None', 'OLS', 'HEH']

list_figures = ['show_dico', 'time_plot_error', 'time_plot_logL', 'time_plot_MC', 'show_Pcum
list_figures = []
dico = {}
for i_cv in range(N_cv):
    dico[i_cv] = {}
    for homeo_method in homeo_methods:
        shl = SHL(homeo_method=homeo_method, seed=seed+i_cv, **opts)
        dico[i_cv][homeo_method] = shl.learn_dico(data=data, list_figures=list_figures, matr

list_figures = ['show_dico']
for i_cv in [one_cv]:
    for homeo_method in homeo_methods:
        print(hl + hs + homeo_method[:3] + hs + hl)
        shl = SHL(homeo_method=homeo_method, seed=seed+i_cv, **opts)
        shl.learn_dico(data=data, list_figures=list_figures, matname=tag + '_' + homeo_metho

        print('size of dictionary = (number of filters, size of imagelets) = ', dico[i_cv][h
        print('average of filters = ',  dico[i_cv][homeo_method].dictionary.mean(axis=1).mea
              '+/-',  dico[i_cv][homeo_method].dictionary.mean(axis=1).std())
        SE = np.sqrt(np.sum(dico[i_cv][homeo_method].dictionary**2, axis=1))
        print('average energy of filters = ', SE.mean(), '+/-', SE.std())
        plt.show()
```

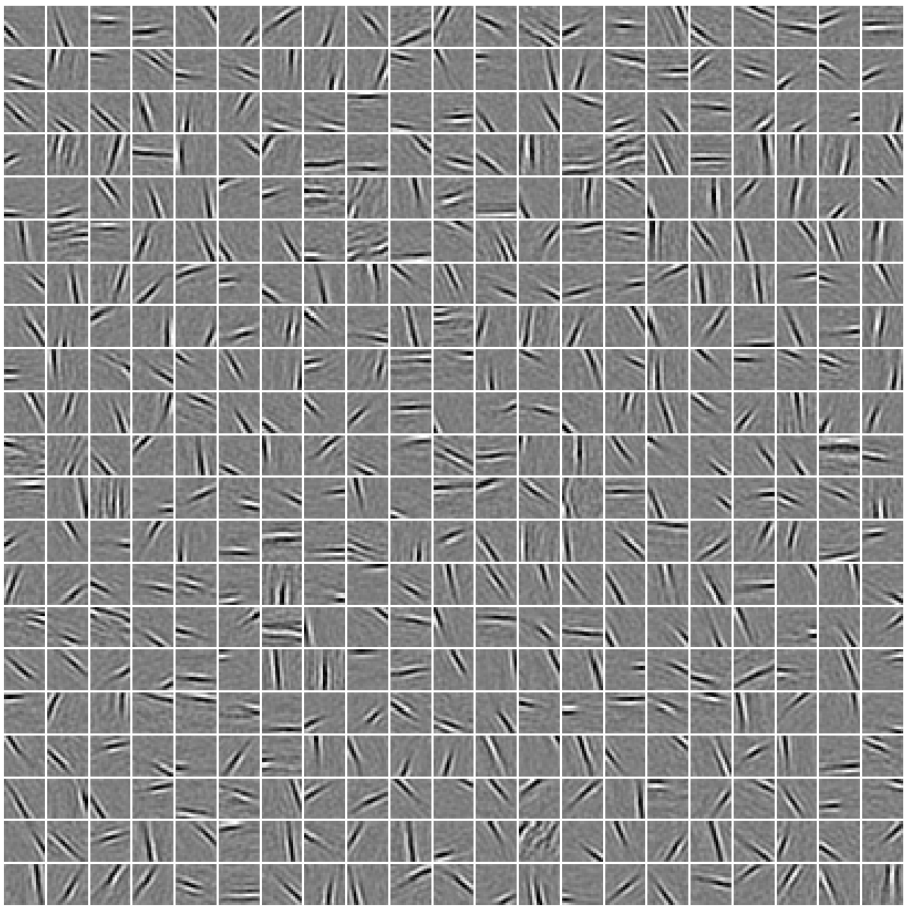
```
----------          Non          ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 324)
average of filters =  -8.681523664810371e-06 +/- 0.0010856202387987371
average energy of filters =  1.0 +/- 4.698989566404069e-17
```



```
----------           OLS           ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 324)
average of filters =  1.4204508020925339e-05 +/- 0.0011077610780090214
average energy of filters =  1.0 +/- 3.991428581974025e-17
```



```
----------           HEH           ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 324)
average of filters =  -7.478164618956859e-06 +/- 0.0010812190907883345
average energy of filters =  1.0 +/- 3.738315377818512e-17
```



panel A: plotting some dictionaries¶

In [9]:

```
pname = '/tmp/panel_A' #pname = fname + '_A'
```

In [10]:

```
from shl_scripts import show_dico
if DEBUG: show_dico(shl, dico[one_cvi_cv][homeo_method], dim_graph=(2,5))
```

In [11]:

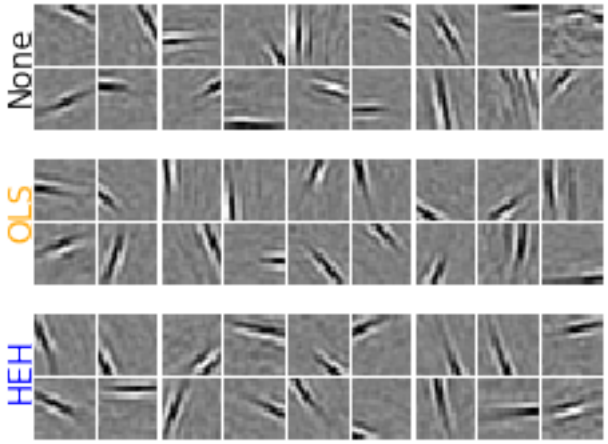
```
dim_graph = (2, 9)
colors = ['black', 'orange', 'blue']
homeo_methods
```

Out[11]:

```
['None', 'OLS', 'HEH']
```

In [12]:

```
subplotpars = dict( left=0.042, right=1., bottom=0., top=1., wspace=0.05, hspace=0.05,)
fig, axs = plt.subplots(3, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subplotp

for ax, color, homeo_method in zip(axs.ravel(), colors, homeo_methods):
    ax.axis(c=color, lw=2, axisbg='w')
    ax.set_facecolor('w')
    fig, ax = show_dico(shl, dico[one_cv][homeo_method], dim_graph=dim_graph, seed=194, fig=
    # ax.set_ylabel(homeo_method)
    ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color=color, rotation=90)#, backg

for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
```



In [13]:

```
if DEBUG: Image(pname +'.png')
```

In [14]:

```
if DEBUG: help(fig.subplots_adjust)
```

In [15]:

```
if DEBUG: help(plt.subplots)
```

In [16]:

```
if DEBUG: help(matplotlib.gridspec.GridSpec)
```

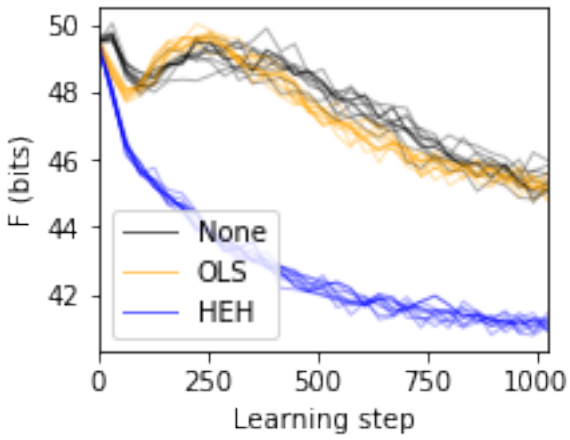
panel B: quantitative comparison¶

In [17]:

```
pname = '/tmp/panel_B' #fname + '_B'
```

In [18]:

```
from shl_scripts import time_plot
variable = 'F'
alpha = .3
subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95)#, wspace=0.05, hspace=0.05,)
fig, ax = plt.subplots(1, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subplotpa
for i_cv in range(N_cv):
    for color, homeo_method in zip(colors, homeo_methods):
        ax.axis(c='b', lw=2, axisbg='w')
        ax.set_facecolor('w')
        if i_cv==0:
            fig, ax = time_plot(shl, dico[i_cv][homeo_method], variable=variable, unit='bits
        else:
            fig, ax = time_plot(shl, dico[i_cv][homeo_method], variable=variable, unit='bits
        # ax.set_ylabel(homeo_method)
        #ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color='k', rotation=90)#, ba
ax.legend(loc='best')
for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
if DEBUG: Image(pname +'.png')
```



Montage of the subplots¶

In [19]:

```
import tikzmagic
```

In [20]:

```
%load_ext tikzmagic
```

In [21]:

```
#DEBUG = True
if DEBUG: help(tikzmagic)
```

%tikz \draw (0,0) rectangle (1,1);%%tikz --save {fname}.pdf \draw[white, fill=white] (0.\linewidth,0) rectangle (1.\linewidth, .382\linewidth) ;

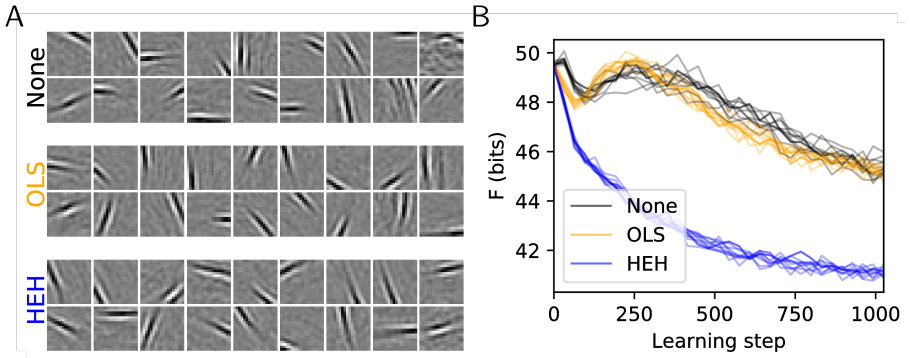
In [22]:

```
%%tikz -f pdf --save {fname}.pdf
\draw[white, fill=white] (0.\linewidth,0) rectangle (1.\linewidth, .382\linewidth) ;
\draw [anchor=north west] (.0\linewidth, .382\linewidth) node {\includegraphics[width=.5\lin
\draw [anchor=north west] (.5\linewidth, .382\linewidth) node {\includegraphics[width=.5\lin
\begin{scope}[font=\bf\sffamily\large]
\draw [anchor=west,fill=white] (.0\linewidth, .382\linewidth) node [above right=-3mm] {$\mat
\draw [anchor=west,fill=white] (.53\linewidth, .382\linewidth) node [above right=-3mm] {$\ma
\end{scope}
```

In [23]:

```
!convert  -density {dpi_export} {fname}.pdf {fname}.jpg
!convert  -density {dpi_export} {fname}.pdf {fname}.png
#!convert  -density {dpi_export} -resize 5400  -units pixelsperinch -flatten  -compress lzw
Image(fname +'.png')
```

Out[23]:



!echo "width=" ; convert {fname}.tiff -format "%[fx:w]" info: !echo ", \nheight=" ; convert {fname}.tiff -format "%[fx:h]" info: !echo ", \nunit=" ; convert {fname}.tiff -format "%U" info:!identify {fname}.tiff

figure 2: Histogram Equalization Homeostasis¶

In [24]:

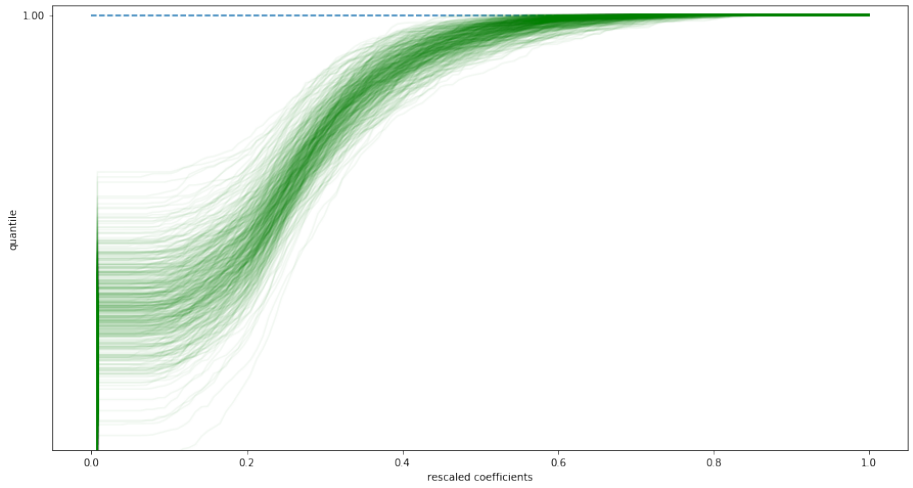
```
fname = 'figure_HEH'
```

First collecting data:

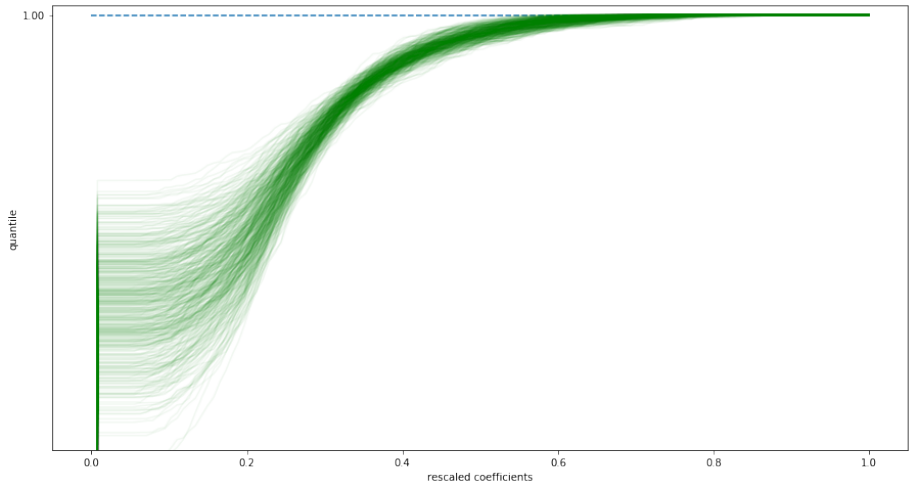
In [25]:

```
list_figures = ['show_Pcum']

dico = {}
for homeo_method in homeo_methods:
    print(hl + hs + homeo_method + hs + hl)
    shl = SHL(homeo_method=homeo_method, **opts)
    #dico[homeo_method] = shl.learn_dico(data=data, list_figures=list_figures, matname=tag +
    dico[homeo_method] = shl.learn_dico(data=data, list_figures=list_figures, matname=tag +
    plt.show()
```

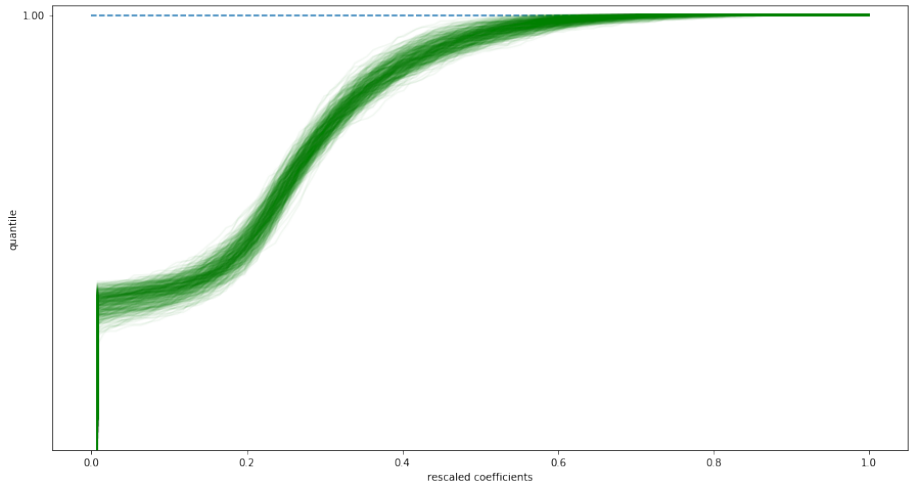
```
----------          None          ----------
```




```
----------          OLS           ----------
```



---------- HEH ----------



In [26]:

```
dico[homeo_method].P_cum.shape
```

Out[26]:

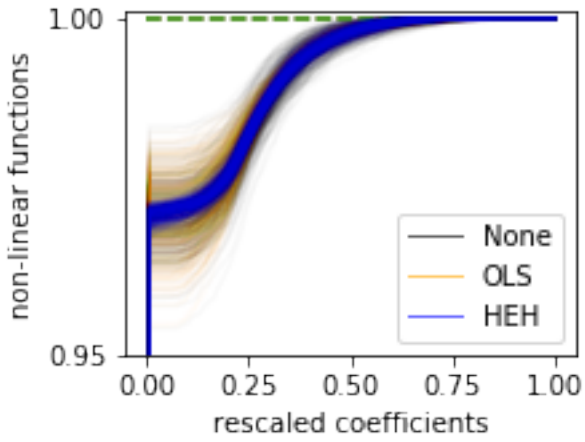
```
(441, 128)
```

panel A: different P_cum¶

In [27]:

```
pname = '/tmp/panel_A' #pname = fname + '_A'
```

```
from shl_scripts import plot_P_cum
variable = 'F'
subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95)#, wspace=0.05, hspace=0.05,)
fig, ax = plt.subplots(1, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subplotpa
for color, homeo_method in zip(colors, homeo_methods):
    ax.axis(c='b', lw=2, axisbg='w')
    ax.set_facecolor('w')
    fig, ax = plot_P_cum(dico[homeo_method].P_cum, ymin=0.95, ymax=1.001,
                         title=None, suptitle=None, ylabel='non-linear functions',
                         verbose=False, n_yticks=21, alpha=.02, c=color, fig=fig, ax=ax)
    ax.plot([0], [0], lw=1, color=color, label=homeo_method, alpha=.6)
    # ax.set_ylabel(homeo_method)
    #ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color='k', rotation=90)#, backgr
ax.legend(loc='lower right')
for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
if DEBUG: Image(pname +'.png')
```



In [28]:

```
if DEBUG: help(fig.legend)
```

panel B: comparing the effects of parameters¶

In [29]:

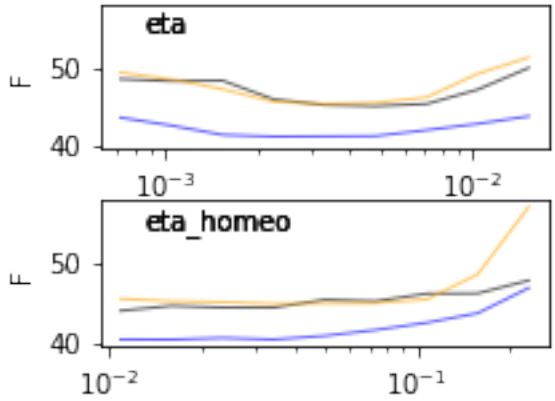
```
pname = '/tmp/panel_B' #fname + '_B'
```

```
from shl_scripts.shl_experiments import SHL_set
```

```
homeo_methods = ['None', 'EMP', 'HAP', 'HEH', 'OLS']
```

```
homeo_methods = ['None', 'OLS', 'HEH']

variables = ['eta', 'alpha_homeo', 'eta_homeo', 'l0_sparseness', 'n_dictionary']
variables = ['eta', 'alpha_homeo', 'eta_homeo', 'l0_sparseness']
variables = ['alpha_homeo', 'eta_homeo']
variables = ['eta', 'alpha_homeo', 'eta_homeo']
variables = ['eta', 'eta_homeo']




list_figures = []

for homeo_method in homeo_methods:
    opts_ = opts.copy()
    opts_.update(homeo_method=homeo_method, datapath=datapath)
    experiments = SHL_set(opts_, tag=tag + '_' + homeo_method)
    experiments.run(variables=variables, n_jobs=1, verbose=0)


import matplotlib.pyplot as plt
subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95, wspace=0.5, hspace=0.35,)

if len(variables)==4:
    fig, axs = plt.subplots(2, 2, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subp
    for i_ax, variable in enumerate(variables):
        for color, homeo_method in zip(colors, homeo_methods):
            opts_ = opts.copy()
            opts_.update(homeo_method=homeo_method, datapath=datapath)
            experiments = SHL_set(opts_, tag=tag + '_' + homeo_method)
            ax = axs[i_ax%2][i_ax//2]
            fig, ax = experiments.scan(variable=variable, list_figures=[], display='final',
            ax.set_xlabel('') #variable
            ax.text(.1, .8,  variable, transform=axs[i_ax].transAxes)
            #axs[i_ax].get_xaxis().set_major_formatter(matplotlib.ticker.ScalarFormatter())

else:
    fig, axs = plt.subplots(len(variables), 1, figsize=(fig_width/2, fig_width/(1+phi)), gri

    for i_ax, variable in enumerate(variables):
        for color, homeo_method in zip(colors, homeo_methods):
            opts_ = opts.copy()
            opts_.update(homeo_method=homeo_method, datapath=datapath)
            experiments = SHL_set(opts_, tag=tag + '_' + homeo_method)
            fig, axs[i_ax] = experiments.scan(variable=variable, list_figures=[], display='f
            axs[i_ax].set_xlabel('') #variable
```

```
            axs[i_ax].text(.1, .8,  variable, transform=axs[i_ax].transAxes)
            #axs[i_ax].get_xaxis().set_major_formatter(matplotlib.ticker.ScalarFormatter())

#fig.legend(loc='lower right')
for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
if DEBUG: Image(pname +'.png')
```



Montage of the subplots¶

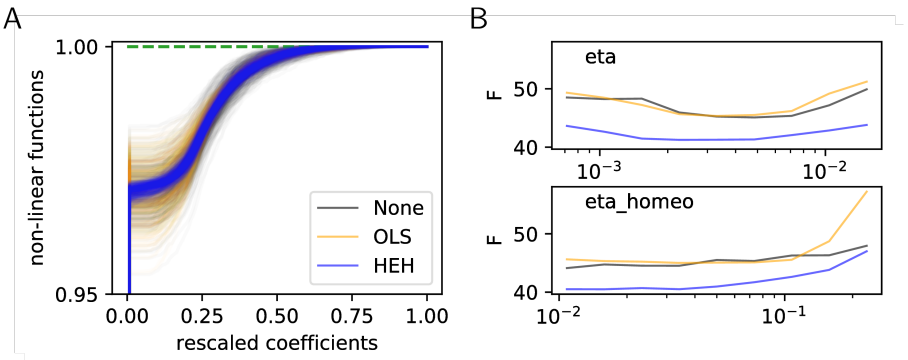
In [30]:

```
%%tikz -f pdf --save {fname}.pdf
\draw[white, fill=white] (0.\linewidth,0) rectangle (1.\linewidth, .382\linewidth) ;
\draw [anchor=north west] (.0\linewidth, .382\linewidth) node {\includegraphics[width=.5\lin
\draw [anchor=north west] (.5\linewidth, .382\linewidth) node {\includegraphics[width=.5\lin
\begin{scope}[font=\bf\sffamily\large]
\draw [anchor=west,fill=white] (.0\linewidth, .382\linewidth) node [above right=-3mm] {$\mat
\draw [anchor=west,fill=white] (.53\linewidth, .382\linewidth) node [above right=-3mm] {$\ma
\end{scope}
```

In [31]:

```
!convert  -density {dpi_export} {fname}.pdf {fname}.jpg
!convert  -density {dpi_export} {fname}.pdf {fname}.png
#!convert  -density {dpi_export} -resize 5400  -units pixelsperinch -flatten  -compress lzw
Image(fname +'.png')
```

Out[31]:



!echo "width=" ; convert {fname}.tiff -format "%[fx:w]" info: !echo ", \nheight=" ; convert {fname}.tiff -format "%[fx:h]" info: !echo ", \nunit=" ; convert {fname}.tiff -format "%U" info:!identify {fname}.tiff

figure 3:¶

learning¶

In [32]:

```
fname = 'figure_HAP'
```

In [33]:

```
colors = ['orange', 'red', 'green', 'blue']
homeo_methods = ['OLS', 'EMP', 'HAP', 'HEH']
list_figures = []
dico = {}
for i_cv in range(N_cv):
    dico[i_cv] = {}
    for homeo_method in homeo_methods:
        shl = SHL(homeo_method=homeo_method, seed=seed+i_cv, **opts)
        dico[i_cv][homeo_method] = shl.learn_dico(data=data, list_figures=list_figures, matr

list_figures = ['show_dico'] if DEBUG else []
for i_cv in [one_cv]:
    for homeo_method in homeo_methods:
        print(hl + hs + homeo_method + hs + hl)
        shl = SHL(homeo_method=homeo_method, seed=seed+i_cv, **opts)
        shl.learn_dico(data=data, list_figures=list_figures, matname=tag + '_' + homeo_metho
        plt.show()
        print('size of dictionary = (number of filters, size of imagelets) = ', dico[i_cv][h
        print('average of filters = ',  dico[i_cv][homeo_method].dictionary.mean(axis=1).mea
              '+/-',  dico[i_cv][homeo_method].dictionary.mean(axis=1).std())
        SE = np.sqrt(np.sum(dico[i_cv][homeo_method].dictionary**2, axis=1))
```

```
        print('average energy of filters = ', SE.mean(), '+/-', SE.std())
```

```
----------          OLS          ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 324)
average of filters =  1.4204508020925339e-05 +/- 0.0011077610780090214
average energy of filters =  1.0 +/- 3.991428581974025e-17
----------          EMP          ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 324)
average of filters =  -3.523143673793376e-05 +/- 0.0011205197173717286
average energy of filters =  1.0 +/- 4.1962486042515756e-17
----------          HAP          ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 324)
average of filters =  1.4205846034666029e-05 +/- 0.0010952255730414218
average energy of filters =  1.0 +/- 3.215820483078219e-17
----------          HEH          ----------
size of dictionary = (number of filters, size of imagelets) =  (441, 324)
average of filters =  -7.478164618956859e-06 +/- 0.0010812190907883345
average energy of filters =  1.0 +/- 3.738315377818512e-17
```

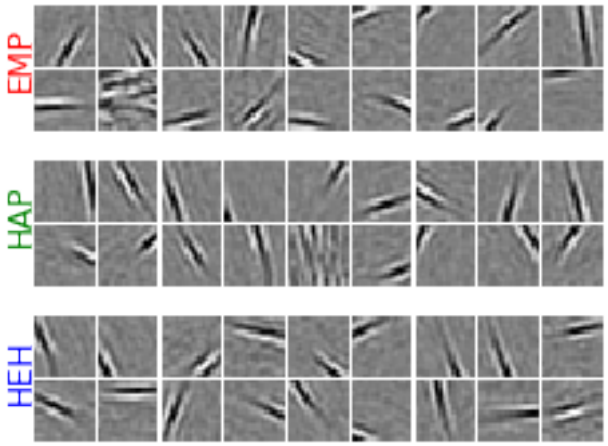
panel A: plotting some dictionaries¶

In [34]:

```
pname = '/tmp/panel_A' #pname = fname + '_A'
```

In [35]:

```
subplotpars = dict( left=0.042, right=1., bottom=0., top=1., wspace=0.05, hspace=0.05,)
fig, axs = plt.subplots(3, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subplotp

for ax, color, homeo_method in zip(axs.ravel(), colors[1:], homeo_methods[1:]):
    ax.axis(c=color, lw=2, axisbg='w')
    ax.set_facecolor('w')
    from shl_scripts import show_dico
    fig, ax = show_dico(shl, dico[one_cv][homeo_method], dim_graph=dim_graph, seed=194, fig=
    # ax.set_ylabel(homeo_method)
    ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color=color, rotation=90)#, backg

for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
```



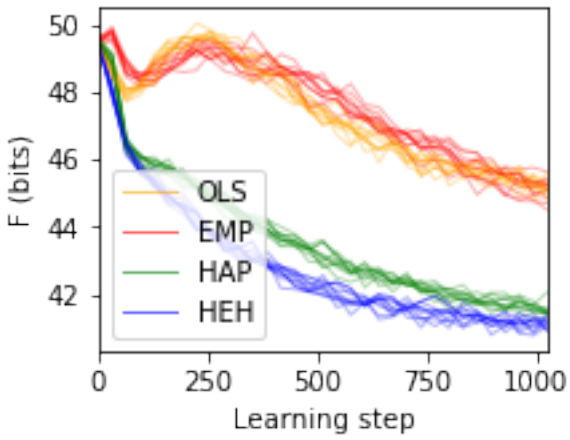
panel B: quantitative comparison¶

In [36]:

```
pname = '/tmp/panel_B' #fname + '_B'
```

In [37]:

```
from shl_scripts import time_plot
variable = 'F'
alpha = .3
subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95)#, wspace=0.05, hspace=0.05,)
fig, ax = plt.subplots(1, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subplotpa
for i_cv in range(N_cv):
    for color, homeo_method in zip(colors, homeo_methods):
        ax.axis(c='b', lw=2, axisbg='w')
        ax.set_facecolor('w')
        if i_cv==0:
            fig, ax = time_plot(shl, dico[i_cv][homeo_method], variable=variable, unit='bits
        else:
            fig, ax = time_plot(shl, dico[i_cv][homeo_method], variable=variable, unit='bits
ax.legend(loc='best')
for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)
if DEBUG: Image(pname +'.png')
```



In [38]:

```
if DEBUG: Image(pname +'.png')
```

Montage of the subplots¶

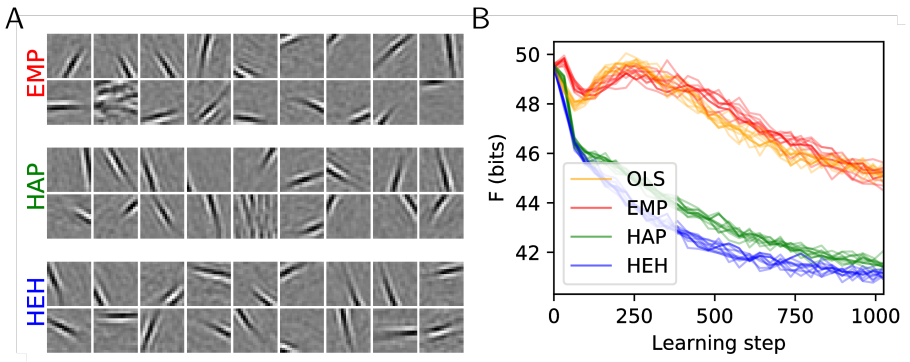
In [39]:

```
%%tikz -f pdf --save {fname}.pdf
\draw[white, fill=white] (0.\linewidth,0) rectangle (1.\linewidth, .382\linewidth) ;
\draw [anchor=north west] (.0\linewidth, .382\linewidth) node {\includegraphics[width=.5\lin
\draw [anchor=north west] (.5\linewidth, .382\linewidth) node {\includegraphics[width=.5\lin
\begin{scope}[font=\bf\sffamily\large]
\draw [anchor=west,fill=white] (.0\linewidth, .382\linewidth) node [above right=-3mm] {$\mat
\draw [anchor=west,fill=white] (.53\linewidth, .382\linewidth) node [above right=-3mm] {$\ma
\end{scope}
```

In [40]:

```
!convert  -density {dpi_export} {fname}.pdf {fname}.jpg
!convert  -density {dpi_export} {fname}.pdf {fname}.png
#!convert  -density {dpi_export} -resize 5400  -units pixelsperinch -flatten  -compress lzw
Image(fname +'.png')
```

Out[40]:



!echo "width=" ; convert {fname}.tiff -format "%[fx:w]" info: !echo ", \nheight=" ; convert {fname}.tiff -format "%[fx:h]" info: !echo ", \nunit=" ; convert {fname}.tiff -format "%U" info:!identify {fname}.tiff

figure 4: Convolutional Neural Network¶

In [41]:

```
fname = 'figure_CNN'
```

In [42]:

```
from CHAMP.DataLoader import LoadData
from CHAMP.DataTools import LocalContrastNormalization, FilterInputData, GenerateMask
from CHAMP.Monitor import DisplayDico, DisplayConvergenceCHAMP, DisplayWhere

import os
datapath = os.path.join("/tmp", "database")
path = os.path.join(datapath, "Face_DataBase")
TrSet, TeSet = LoadData('Face', path, decorrelate=False, resize=(65, 65))

# MP Parameters
nb_dico = 20
width = 9
dico_size = (width, width)
l0 = 20
seed = 42
# Learning Parameters
eta = .05
nb_epoch = 500

TrSet, TeSet = LoadData('Face', path, decorrelate=False, resize=(65, 65))
N_TrSet, _, _, _ = LocalContrastNormalization(TrSet)
Filtered_L_TrSet = FilterInputData(
    N_TrSet, sigma=0.25, style='Custom', start_R=15)
```

```
mask = GenerateMask(full_size=(nb_dico, 1, width, width), sigma=0.8, style='Gaussian')

from CHAMP.CHAMP_Layer import CHAMP_Layer

from CHAMP.DataTools import SaveNetwork, LoadNetwork
homeo_methods = ['None', 'HAP']

for homeo_method, eta_homeo  in zip(homeo_methods, [0., 0.0025]):
    ffname = 'cache_dir_CNN/CHAMP_low_' + homeo_method + '.pkl'
    try:
        L1_mask = LoadNetwork(loading_path=ffname)
    except:
        L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico,
                              dico_size=dico_size, mask=mask, verbose=1)
        dico_mask = L1_mask.TrainLayer(
            Filtered_L_TrSet, eta=eta, eta_homeo=eta_homeo, nb_epoch=nb_epoch, seed=seed)
        SaveNetwork(Network=L1_mask, saving_path=ffname)
```

panel A: plotting some dictionaries¶

In [43]:

```
pname = '/tmp/panel_A' #pname = fname + '_A'
```

subplotpars = dict(left=0.042, right=1., bottom=0., top=1., wspace=0.05, hspace=0.05,) fig, axs = plt.subplots(2, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subplotpars) for ax, color, homeo_method in zip(axs.ravel(), ['black', 'green'], homeo_methods): ax.axis(c=color, lw=2, axisbg='w') ax.set_facecolor('w') ffname = 'cache_dir/CHAMP_low_' + homeo_method + '.pkl' L1_mask = LoadNetwork(loading_path=ffname) fig, ax = DisplayDico(L1_mask.dictionary, fig=fig, ax=ax) # ax.set_ylabel(homeo_method) ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color=color, rotation=90)#, backgroundcolor='white' for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export)

In [44]:

```
subplotpars = dict( left=0.042, right=1., bottom=0., top=1., wspace=0.05, hspace=0.05,)

for color, homeo_method in zip(['black', 'green'], homeo_methods):
    #fig, axs = plt.subplots(1, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=sub
    ffname = 'cache_dir_CNN/CHAMP_low_' + homeo_method + '.pkl'
    L1_mask = LoadNetwork(loading_path=ffname)
    fig, ax = DisplayDico(L1_mask.dictionary)
    # ax.set_ylabel(homeo_method)
    #for ax in list(axs):
    #    ax.axis(c=color, lw=2, axisbg='w')
```

```
    #    ax.set_facecolor('w')
    ax[0].text(-4, 3, homeo_method, fontsize=8, color=color, rotation=90)#, backgroundcolor=
    plt.tight_layout( pad=0., w_pad=0., h_pad=.0)


    for ext in FORMATS: fig.savefig(pname + '_' + homeo_method + ext, dpi=dpi_export)
```

```
<Figure size 576x28.8 with 0 Axes>
```


```
<Figure size 576x28.8 with 0 Axes>
```

panel B: quantitative comparison¶

In [45]:

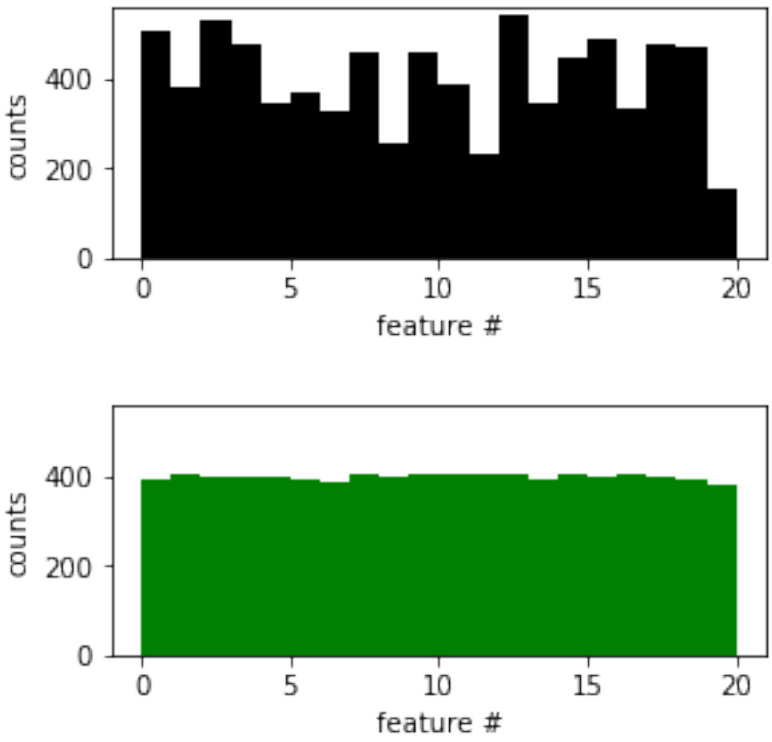
```
pname = '/tmp/panel_B' #fname + '_B'
```

from shl_scripts import time_plot variable = 'F' alpha = .3 subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95)#, wspace=0.05, hspace=0.05,) fig, axs = plt.subplots(2, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=subplotpars) for ax, color, homeo_method in zip(axs, ['black', 'green'], homeo_methods): print(ax, axs) ffname = 'cache_dir_CNN/CHAMP_low_' + homeo_method + '.pkl' L1_mask = LoadNetwork(loading_path=ffname) fig, ax = DisplayConvergenceCHAMP(L1_mask, to_display=['histo'], fig=fig, ax=ax) ax.axis(c=color, lw=2, axisbg='w') ax.set_facecolor('w') # ax.set_ylabel(homeo_method) #ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color=color, rotation=90)#, backgroundcolor='white' for ext in FORMATS: fig.savefig(pname + ext, dpi=dpi_export) if DEBUG: Image(pname +'.png')

In [46]:

```
from shl_scripts import time_plot
variable = 'F'
alpha = .3
subplotpars = dict(left=0.2, right=.95, bottom=0.2, top=.95)#, wspace=0.05, hspace=0.05,)

for color, homeo_method in zip(['black', 'green'], homeo_methods):
    #fig, axs = plt.subplots(1, 1, figsize=(fig_width/2, fig_width/(1+phi)), gridspec_kw=sub
    ffname = 'cache_dir_CNN/CHAMP_low_' + homeo_method + '.pkl'
    L1_mask = LoadNetwork(loading_path=ffname)
    fig, ax = DisplayConvergenceCHAMP(L1_mask, to_display=['histo'], color=color)
    ax.axis(c=color, lw=2, axisbg='w')
```

```
ax.set_facecolor('w')
ax.set_ylabel('counts')
ax.set_xlabel('feature #')
ax.set_ylim(0, 560)
#ax.text(-8, 7*dim_graph[0], homeo_method, fontsize=12, color=color, rotation=90)#, back
#ax[0].text(-8, 3, homeo_method, fontsize=12, color=color, rotation=90)#, backgroundcolo

for ext in FORMATS: fig.savefig(pname + '_' + homeo_method + ext, dpi=dpi_export)
if DEBUG: Image(pname +'.png')
```



Montage of the subplots¶

In [47]:

```
%ls -ltr /tmp/panel_*
```

```
-rw-r--r--  1 laurentperrinet  wheel   72275 Sep 28 13:13 /tmp/panel_A.pdf
-rw-r--r--  1 laurentperrinet  wheel   84631 Sep 28 13:13 /tmp/panel_A.png
-rw-r--r--  1 laurentperrinet  wheel   22106 Sep 28 13:13 /tmp/panel_B.pdf
-rw-r--r--  1 laurentperrinet  wheel  532305 Sep 28 13:13 /tmp/panel_B.png
-rw-r--r--  1 laurentperrinet  wheel   27370 Sep 28 13:13 /tmp/panel_A_None.pdf
-rw-r--r--  1 laurentperrinet  wheel   18909 Sep 28 13:13 /tmp/panel_A_None.png
```

```
-rw-r--r--@ 1 laurentperrinet  wheel   26909 Sep 28 13:13 /tmp/panel_A_HAP.pdf
-rw-r--r--  1 laurentperrinet  wheel   16431 Sep 28 13:13 /tmp/panel_A_HAP.png
-rw-r--r--  1 laurentperrinet  wheel    8816 Sep 28 13:13 /tmp/panel_B_None.pdf
-rw-r--r--  1 laurentperrinet  wheel   39035 Sep 28 13:13 /tmp/panel_B_None.png
-rw-r--r--  1 laurentperrinet  wheel    8813 Sep 28 13:13 /tmp/panel_B_HAP.pdf
-rw-r--r--  1 laurentperrinet  wheel   38743 Sep 28 13:13 /tmp/panel_B_HAP.png
```

In [48]:

```
fname
```

Out[48]:

```
'figure_CNN'
```

In [49]:

```
382+191
```

Out[49]:

```
573
```

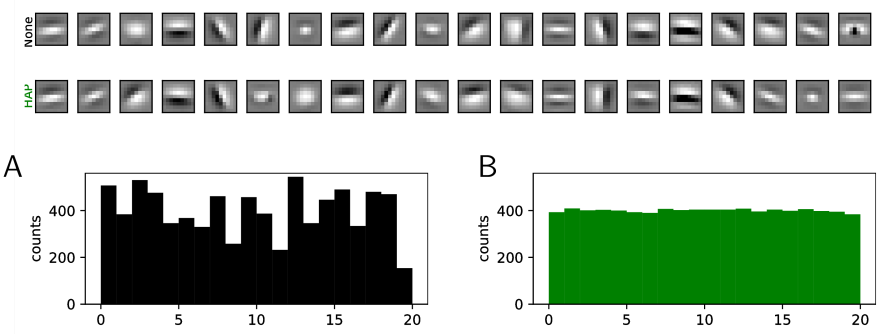
In [50]:

```
%%tikz -f pdf --save {fname}.pdf
\draw[white, fill=white] (0.\linewidth,0) rectangle (1.\linewidth, .382\linewidth) ;
\draw [anchor=north west] (.0\linewidth, .375\linewidth) node {\includegraphics[width=.95\li
\draw [anchor=north west] (.0\linewidth, .300\linewidth) node {\includegraphics[width=.95\li
\draw [anchor=north west] (.0\linewidth, .191\linewidth) node {\includegraphics[width=.45\li
\draw [anchor=north west] (.5\linewidth, .191\linewidth) node {\includegraphics[width=.45\li
\begin{scope}[font=\bf\sffamily\large]
%\draw [anchor=west,fill=white] (.0\linewidth, .382\linewidth) node [above right=-3mm] {$\ma
\draw [anchor=west,fill=white] (.0\linewidth, .191\linewidth) node [above right=-3mm] {$\mat
\draw [anchor=west,fill=white] (.53\linewidth, .191\linewidth) node [above right=-3mm] {$\ma
\end{scope}
```

In [51]:

```
!convert  -density {dpi_export} {fname}.pdf {fname}.jpg
!convert  -density {dpi_export} {fname}.pdf {fname}.png
#!convert  -density {dpi_export} -resize 5400  -units pixelsperinch -flatten  -compress lzw
Image(fname +'.png')
```

Out[51]:



!echo "width=" ; convert {fname}.tiff -format "%[fx:w]" info: !echo ", \nheight=" ; convert {fname}.tiff -format "%[fx:h]" info: !echo ", \nunit=" ; convert {fname}.tiff -format "%U" info:!identify {fname}.tiff

coding¶

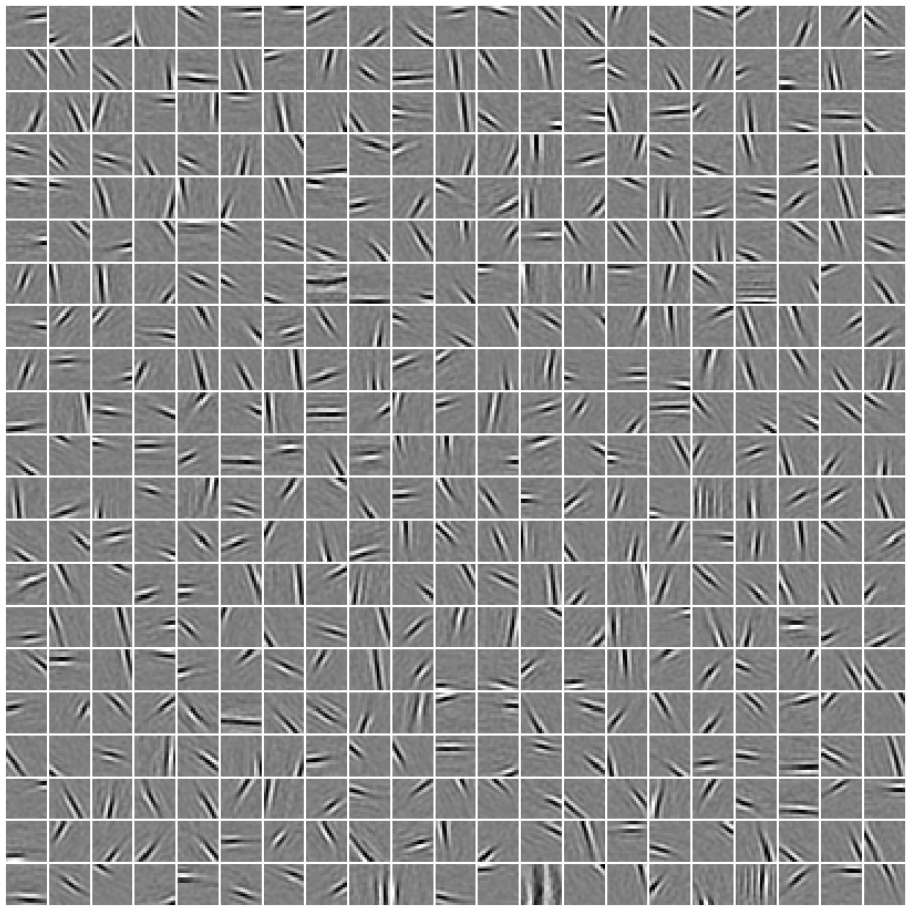
The learning itself is done via a gradient descent but is highly dependent on the coding / decoding algorithm. This belongs to a another function (in the shl_encode.py script)

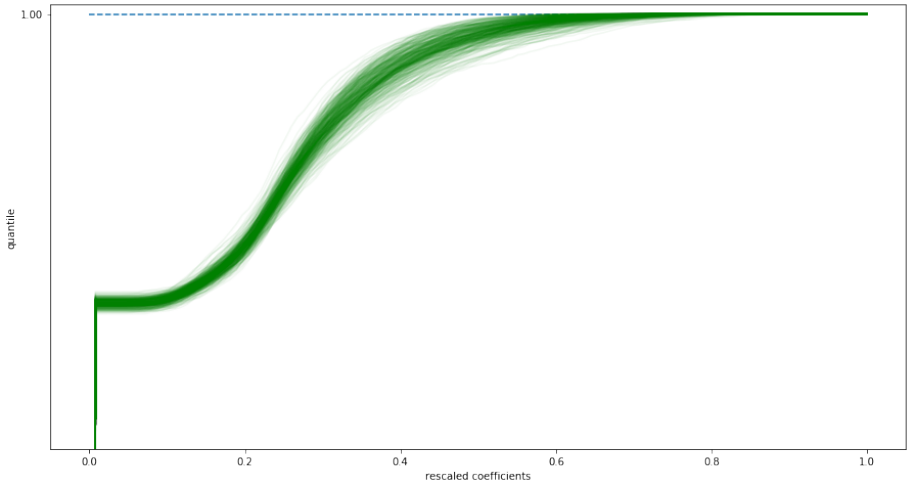
Supplementary controls¶

starting a learning¶

In [52]:

```
shl = SHL(**opts)
list_figures = ['show_dico', 'show_Pcum', 'time_plot_F']
dico = shl.learn_dico(data=data, list_figures=list_figures, matname=tag + '_vanilla')
```




1.00
quantile
0.0
0.2
0.4
0.6
0.8
1.0
rescaled coefficients

In [53]:

```
print('size of dictionary = (number of filters, size of imagelets) = ', dico.dictionary.shap
print('average of filters = ',  dico.dictionary.mean(axis=1).mean(),
      '+/-',  dico.dictionary.mean(axis=1).std())
SE = np.sqrt(np.sum(dico.dictionary**2, axis=1))
print('average energy of filters = ', SE.mean(), '+/-', SE.std())
```

```
size of dictionary = (number of filters, size of imagelets) =  (441, 324)
average of filters =  -3.1637286525674506e-05 +/- 0.0010915831873405475
average energy of filters =  1.0 +/- 3.884971301458624e-17
```

```
[autoreload of IPython.core.ultratb failed: Traceback (most recent call last):
  File "/usr/local/lib/python3.6/site-packages/IPython/extensions/autoreload.py", line 245,
    if py_filename in self.failed:
  File "/usr/local/lib/python3.6/site-packages/IPython/extensions/autoreload.py", line 368,
    module.__dict__['__name__'] = old_name
  File "/usr/local/Cellar/python/3.6.5_1/Frameworks/Python.framework/Versions/3.6/lib/python
    return importlib.reload(module)
  File "/usr/local/Cellar/python/3.6.5_1/Frameworks/Python.framework/Versions/3.6/lib/python
    _bootstrap._exec(spec, module)
  File "<frozen importlib._bootstrap>", line 618, in _exec
  File "<frozen importlib._bootstrap_external>", line 678, in exec_module
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "/usr/local/lib/python3.6/site-packages/IPython/core/ultratb.py", line 128, in <modul
    import IPython.utils.colorable as colorable
AttributeError: module 'IPython' has no attribute 'utils'
]
```

getting help¶

In [54]:

```
help(shl)
```

```
Help on SHL in module shl_scripts.shl_experiments object:

class SHL(builtins.object)
 |  Base class to define SHL experiments:
 |      - initialization
 |      - coding and learning
 |      - visualization
 |      - quantitative analysis
 |
 |  Methods defined here:
 |
 |  __init__(self, height=256, width=256, patch_width=18, N_patches=65536, datapath='../data
 |      Initialize self.  See help(type(self)) for accurate signature.
```

```
 |
 |  code(self, data, dico, coding_algorithm='mp', matname=None, P_cum=None, fit_tol=None, l0
 |
 |  decode(self, sparse_code, dico)
 |
 |  get_data(self, matname=None, patch_width=None)
 |
 |  learn_dico(self, dictionary=None, precision=None, P_cum=None, data=None, matname=None, n
 |
 |  plot_error(self, dico, **fig_kwargs)
 |
 |  plot_variance(self, sparse_code, **fig_kwargs)
 |
 |  plot_variance_histogram(self, sparse_code, **fig_kwargs)
 |
 |  show_Pcum(self, dico, title=None, verbose=False, n_yticks=21, alpha=0.05, c='g', **fig_k
 |
 |  show_dico(self, dico, data=None, title=None, **fig_kwargs)
 |
 |  show_dico_in_order(self, dico, data=None, title=None, **fig_kwargs)
 |
 |  time_plot(self, dico, variable='kurt', N_nosample=1, **fig_kwargs)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [55]:

```
help(dico)
```

```
Help on SparseHebbianLearning in module shl_scripts.shl_learn object:

class SparseHebbianLearning(builtins.object)
 |  Sparse Hebbian learning
 |
 |  Finds a dictionary (a set of atoms) that can best be used to represent data
 |  using a sparse code.
 |
 |  Parameters
 |  ----------
 |
```

```
|  n_dictionary : int,
|      Number of dictionary elements to extract
|
|  eta : float or dict
|      Gives the learning parameter for the homeostatic gain.
|
|  n_iter : int,
|      total number of iterations to perform
|
|  eta_homeo : float
|      Gives the learning parameter for the homeostatic gain.
|
|  alpha_homeo : float
|      Gives the smoothing exponent  for the homeostatic gain
|      If equal to 1 the homeostatic learning rule learns a linear relation to
|      variance.
|
|  dictionary : array of shape (n_dictionary, n_pixels),
|      initial value of the dictionary for warm restart scenarios
|      Use ``None`` for a new learning.
|
|  fit_algorithm : {'mp', 'lars', 'cd'}
|      see sparse_encode
|
|  batch_size : int,
|      The number of samples to take in each batch.
|
|  l0_sparseness : int, ``0.1 * n_pixels`` by default
|      Number of nonzero coefficients to target in each column of the
|      solution. This is only used by `algorithm='lars'`, `algorithm='mp'`  and
|      `algorithm='omp'`.
|
|  fit_tol : float, 1. by default
|      If `algorithm='lasso_lars'` or `algorithm='lasso_cd'`, `fit_tol` is the
|      penalty applied to the L1 norm.
|      If `algorithm='threshold'`, `fit_tol` is the absolute value of the
|      threshold below which coefficients will be squashed to zero.
|      If `algorithm='mp'` or `algorithm='omp'`, `fit_tol` is the tolerance
|      parameter: the value of the reconstruction error targeted. In this case,
|      it overrides `l0_sparseness`.
|
|  verbose :
|      degree of verbosity of the printed output
|
|  Attributes
|  ----------
```

```
|  dictionary : array, [n_dictionary, n_pixels]
|      dictionary extracted from the data
|
|
|  Notes
|  -----
|  **References:**
|
|  Olshausen BA, Field DJ (1996).
|  Emergence of simple-cell receptive field properties by learning a sparse code for natura
|  Nature, 381: 607-609. (http://redwood.berkeley.edu/bruno/papers/nature-paper.pdf)
|
|  Olshausen BA, Field DJ (1997)
|  Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1?
|  Vision Research, 37: 3311-3325.   (http://redwood.berkeley.edu/bruno/papers/VR.pdf)
|
|  See also
|  --------
|  http://scikit-learn.org/stable/auto_examples/decomposition/plot_image_denoising.html
|
|  Methods defined here:
|
|  __init__(self, fit_algorithm, dictionary=None, precision=None, eta=0.003, beta1=0.9, bet
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  fit(self, X, y=None)
|      Fit the model from data in X.
|
|      Parameters
|      ----------
|      X: array-like, shape (n_samples, n_pixels)
|          Training vector, where n_samples in the number of samples
|          and n_pixels is the number of features.
|
|      Returns
|      -------
|      self : object
|          Returns the instance itself.
|
|  transform(self, X, algorithm=None, l0_sparseness=None, fit_tol=None, alpha_MP=None)
|      Fit the model from data in X.
|
|      Parameters
|      ----------
|      X: array-like, shape (n_samples, n_pixels)
|          Training vector, where n_samples in the number of samples
```

```
|          and n_pixels is the number of features.
|
|      Returns
|      -------
|      self : object
|          Returns sparse code.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __slotnames__ = []
```

loading a database¶

Loading patches, with or without mask:

In [ ]:

```
N_patches = 12
from shl_scripts.shl_tools import show_data

for i, (do_mask, label) in enumerate(zip([False, True], ['Without mask', 'With mask'])):
    data_ = SHL(DEBUG_DOWNSCALE=1, verbose=0, N_patches=N_patches, n_image=1, do_mask=do_mas
    fig, axs = show_data(data_)
    axs[0].set_ylabel(label)
    plt.show()
```

Testing different algorithms¶

In [56]:

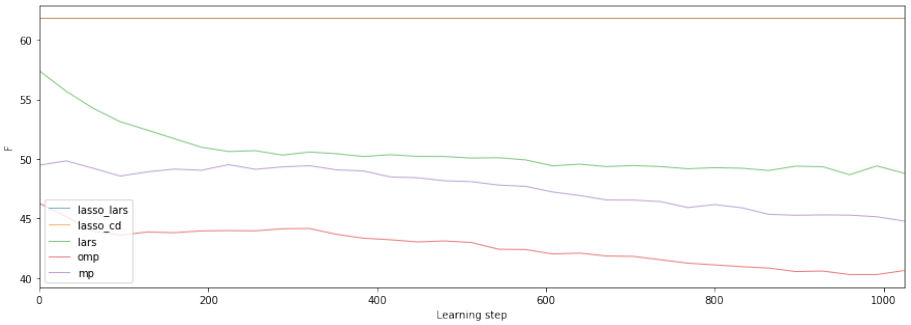
```
fig, ax = None, None

for algorithm in ['lasso_lars', 'lasso_cd', 'lars', 'omp', 'mp']: # 'threshold',
    opts_ = opts.copy()
    opts_.update(homeo_method='None', learning_algorithm=algorithm, verbose=0)
    shl = SHL(**opts_)
    dico= shl.learn_dico(data=data, list_figures=[],
```

```
                    matname=tag + ' - algorithm={}'.format(algorithm))
    fig, ax = shl.time_plot(dico, variable='F', fig=fig, ax=ax, label=algorithm)

ax.legend()
```

```
ERROR! Session/line number was not unique in database. History logging moved to new session
```

Out[56]:

```
<matplotlib.legend.Legend at 0x146eaea20>
```



Testing two different dictionary initalization strategies¶

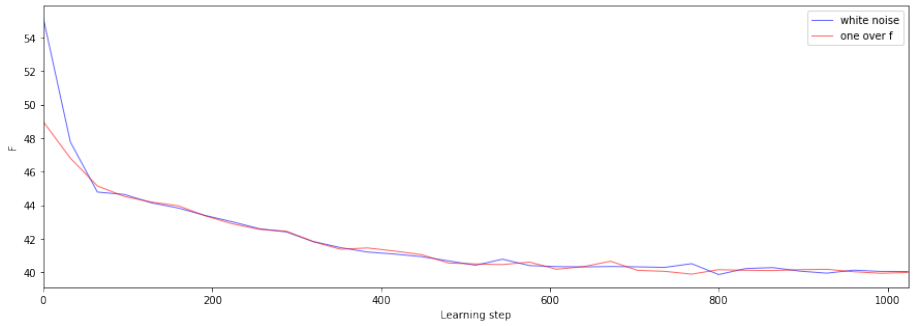
White Noise Initialization + Learning

In [57]:

```
shl = SHL(one_over_F=False, **opts)
dico_w = shl.learn_dico(data=data, matname=tag + '_WHITE', list_figures=[])
shl = SHL(one_over_F=True, **opts)
dico_1oF = shl.learn_dico(data=data, matname=tag + '_OVF', list_figures=[])
fig_error, ax_error = None, None
fig_error, ax_error = shl.time_plot(dico_w, variable='F', fig=fig_error, ax=ax_error, color=
fig_error, ax_error = shl.time_plot(dico_1oF, variable='F', fig=fig_error, ax=ax_error, colo
#ax_error.set_ylim((0, .65))
ax_error.legend(loc='best')
```

Out[57]:

```
<matplotlib.legend.Legend at 0x1468d6048>
```



Testing two different learning rates strategies¶

We use by defaut the strategy of ADAM, see https://arxiv.org/pdf/1412.6980.pdf

In [ ]:

```
shl = SHL(beta1=0., **opts)
dico_fixed = shl.learn_dico(data=data, matname=tag + '_fixed', list_figures=[])
shl = SHL(**opts)
dico_default = shl.learn_dico(data=data, matname=tag + '_default', list_figures=[])
fig_error, ax_error = None, None
fig_error, ax_error = shl.time_plot(dico_fixed, variable='F', fig=fig_error, ax=ax_error, co
fig_error, ax_error = shl.time_plot(dico_default, variable='F', fig=fig_error, ax=ax_error,
#ax_error.set_ylim((0, .65))
ax_error.legend(loc='best')
```

Perspectives¶

Convolutional neural networks¶

In [ ]:

```
from CHAMP.DataLoader import LoadData
from CHAMP.DataTools import LocalContrastNormalization, FilterInputData, GenerateMask
from CHAMP.Monitor import DisplayDico, DisplayConvergenceCHAMP, DisplayWhere

import os
home = os.getenv('HOME')
datapath = os.path.join("/tmp", "database")
path = os.path.join(datapath, "Face_DataBase")
TrSet, TeSet = LoadData('Face', path, decorrelate=False, resize=(65, 65))
to_display = TrSet[0][0, 0:10, :, :, :]
print('Size=', TrSet[0].shape)
DisplayDico(to_display)
```

Training on a face database¶

In [ ]:

```
# MP Parameters
nb_dico = 20
width = 9
dico_size = (width, width)
l0 = 20
seed = 42
# Learning Parameters
eta = .05
nb_epoch = 500

TrSet, TeSet = LoadData('Face', path, decorrelate=False, resize=(65, 65))
N_TrSet, _, _, _ = LocalContrastNormalization(TrSet)
Filtered_L_TrSet = FilterInputData(
    N_TrSet, sigma=0.25, style='Custom', start_R=15)
to_display = Filtered_L_TrSet[0][0, 0:10, :, :, :]
DisplayDico(to_display)

mask = GenerateMask(full_size=(nb_dico, 1, width, width), sigma=0.8, style='Gaussian')
DisplayDico(mask)
```

Training the ConvMP Layer with homeostasis¶

In [ ]:

```
from CHAMP.CHAMP_Layer import CHAMP_Layer

from CHAMP.DataTools import SaveNetwork, LoadNetwork
fname = 'cache_dir_CNN/CHAMP_low_None.pkl'
try:
    L1_mask = LoadNetwork(loading_path=fname)
except:
    L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico,
                      dico_size=dico_size, mask=mask, verbose=2)
    dico_mask = L1_mask.TrainLayer(
        Filtered_L_TrSet, eta=eta, nb_epoch=nb_epoch, seed=seed)
    SaveNetwork(Network=L1_mask, saving_path=fname)

DisplayDico(L1_mask.dictionary)
DisplayConvergenceCHAMP(L1_mask, to_display=['error', 'histo'])
DisplayWhere(L1_mask.where)
```

Training the ConvMP Layer with homeostasis¶

In [ ]:

```
fname = 'cache_dir_CNN/CHAMP_low_HAP.pkl'
try:
    L1_mask = LoadNetwork(loading_path=fname)
except:

    # Learning Parameters
    eta_homeo = 0.0025
    L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico,
                          dico_size=dico_size, mask=mask, verbose=1)
    dico_mask = L1_mask.TrainLayer(
        Filtered_L_TrSet, eta=eta, eta_homeo=eta_homeo, nb_epoch=nb_epoch, seed=seed)
    SaveNetwork(Network=L1_mask, saving_path=fname)

DisplayDico(L1_mask.dictionary)
DisplayConvergenceCHAMP(L1_mask, to_display=['error', 'histo'])
DisplayWhere(L1_mask.where)
```

Reconstructing the input image¶

In [ ]:

```
from CHAMP.DataTools import Rebuilt
import torch
rebuilt_image = Rebuilt(torch.FloatTensor(L1_mask.code), L1_mask.dictionary)
DisplayDico(rebuilt_image[0:10, :, :, :])
```

Training the ConvMP Layer with higher-level filters¶

We train higher-level feature vectors by forcing the network to :

- learn bigger filters,
- represent the information using a bigger dictionary (higher sparseness)
- represent the information with less features (higher sparseness)

In [ ]:

```
fname = 'cache_dir_CNN/CHAMP_high_None.pkl'
try:
    L1_mask = LoadNetwork(loading_path=fname)
except:

    nb_dico = 60
    width = 19
    dico_size = (width, width)
    l0 = 5
    mask = GenerateMask(full_size=(nb_dico, 1, width, width), sigma=0.8, style='Gaussian')
    # Learning Parameters
    eta_homeo = 0.0
```

```
    eta = .05
    nb_epoch = 500
    # learn
    L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico,
                          dico_size=dico_size, mask=mask, verbose=0)
    dico_mask = L1_mask.TrainLayer(
        Filtered_L_TrSet, eta=eta, eta_homeo=eta_homeo, nb_epoch=nb_epoch, seed=seed)
    SaveNetwork(Network=L1_mask, saving_path=fname)


DisplayDico(L1_mask.dictionary)
DisplayConvergenceCHAMP(L1_mask, to_display=['error', 'histo'])
DisplayWhere(L1_mask.where)
```

In [ ]:

```
fname = 'cache_dir_CNN/CHAMP_high_HAP.pkl'
try:
    L1_mask = LoadNetwork(loading_path=fname)
except:

    nb_dico = 60
    width = 19
    dico_size = (width, width)
    l0 = 5
    mask = GenerateMask(full_size=(nb_dico, 1, width, width), sigma=0.8, style='Gaussian')
    # Learning Parameters
    eta_homeo = 0.0025
    eta = .05
    nb_epoch = 500
    # learn
    L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico,
                          dico_size=dico_size, mask=mask, verbose=0)
    dico_mask = L1_mask.TrainLayer(
        Filtered_L_TrSet, eta=eta, eta_homeo=eta_homeo, nb_epoch=nb_epoch, seed=seed)
    SaveNetwork(Network=L1_mask, saving_path=fname)

DisplayDico(L1_mask.dictionary)
DisplayConvergenceCHAMP(L1_mask, to_display=['error', 'histo'])
DisplayWhere(L1_mask.where)
```

Training on MNIST database¶

fname = 'cache_dir/CHAMP_MNIST_HAP.pkl' try: L1_mask = LoadNetwork(loading_path=fname) except: path = os.path.join(datapath, "MNISTtorch") TrSet, TeSet = LoadData('MNIST', data_path=path) N_TrSet, _, _, _ = LocalContrastNormalization(TrSet) Filtered_L_TrSet

= FilterInputData(N_TrSet, sigma=0.25, style='Custom', start_R=15) nb_dico = 60 width = 7 dico_size = (width, width) l0 = 15 # Learning Parameters eta_homeo = 0.0025 eta = .05 nb_epoch = 500 # learn L1_mask = CHAMP_Layer(l0_sparseness=l0, nb_dico=nb_dico, dico_size=dico_size, mask=mask, verbose=2) dico_mask = L1_mask.TrainLayer(Filtered_L_TrSet, eta=eta, eta_homeo=eta_homeo, nb_epoch=nb_epoch, seed=seed) SaveNetwork(Network=L1_mask, saving_path=fname) DisplayDico(L1_mask.dictionary) DisplayConvergenceCHAMP(L1_mask, to_display=['error', 'histo']) DisplayWhere(L1_mask.where)

Computational details¶

caching simulation data¶

A convenience script to run and cache most learning items in this notebooks:

In [ ]:

```
!ls -l {shl.cache_dir}/{tag}*
#!rm {shl.cache_dir}/{tag}*lock*
#!rm {shl.cache_dir}/{tag}*
#!ls -l {shl.cache_dir}/{tag}*
```

In [ ]:

```
%%writefile model.py
#!/usr/bin/env python3
# -*- coding: utf-8 -*
tag = 'ICLR'
from shl_scripts.shl_experiments import SHL, prun
# pre-loading data
datapath = '../../SparseHebbianLearning/database'
opts = dict(eta=0.0033, eta_homeo=0.05, alpha_homeo=2.5, cache_dir='cache_dir_42', datapath=
shl = SHL(**opts)
data = shl.get_data(matname=tag)

# running main simulations
# Figure 1 & 3
N_cv = 10
homeo_methods = ['None', 'OLS', 'HEH', 'HAP', 'EMP']
seed = 42

# running in parallel on a multi-core machine
import sys
try:
    n_jobs = sys.argv[2]
except:
```

```
    n_jobs = 4
    n_jobs = 9
    n_jobs = 10
    n_jobs = 35
    n_jobs = 1

list_figures = []

from shl_scripts.shl_experiments import SHL_set
for homeo_method in homeo_methods:
    opts_ = opts.copy()
    opts_.update(homeo_method=homeo_method)
    experiments = SHL_set(opts_, tag=tag + '_' + homeo_method, N_scan=N_cv)
    experiments.run(variables=['seed'], n_jobs=n_jobs, verbose=0)

# Figure 2-B
variables = ['eta', 'alpha_homeo', 'eta_homeo', 'l0_sparseness']

variables = ['eta', 'alpha_homeo', 'eta_homeo']

for homeo_method in homeo_methods:
    opts_ = opts.copy()
    opts_.update(homeo_method=homeo_method)
    experiments = SHL_set(opts_, tag=tag + '_' + homeo_method)
    experiments.run(variables=variables, n_jobs=n_jobs, verbose=0)

# Annex X.X
for algorithm in ['lasso_lars', 'lasso_cd', 'lars', 'omp', 'mp']: # 'threshold',
    opts_ = opts.copy()
    opts_.update(homeo_method='None', learning_algorithm=algorithm, verbose=0)
    shl = SHL(**opts_)
    dico= shl.learn_dico(data=data, list_figures=[],
                   matname=tag + ' - algorithm={}'.format(algorithm))
```

In [ ]:

```
#%run model.py
```

Version used¶

In [ ]:

```
%load_ext version_information
%version_information numpy, shl_scripts
```

version control¶

In [ ]:

```
!git status
```

In [ ]:

```
!git pull
```

In [ ]:

```
!git commit -am' {tag} : re-running notebooks'
```

In [ ]:

```
!git push
```

exporting the notebook¶

In [ ]:

```
!jupyter nbconvert Annex.ipynb
```

In [ ]:

```
#!jupyter-nbconvert --template report --to pdf Annex.ipynb
```

In [ ]:

```
!pandoc Annex.html -o Annex.pdf
```

In [ ]:

```
!zip Annex.zip Annex.html
```

Done. Thanks for your attention!