Supplement of Geosci. Model Dev., 12, 2091–2105, 2019
https://doi.org/10.5194/gmd-12-2091-2019-supplement

Geoscientific Model Development
Open Access
EGU

Supplement of

LSCE-FFNN-v1: a two-step neural network model for the reconstruction of surface ocean pCO_2 over the global ocean

Anna Denvil-Sommer et al.

Correspondence to: Anna Denvil-Sommer (anna.sommer.lab@gmail.com)

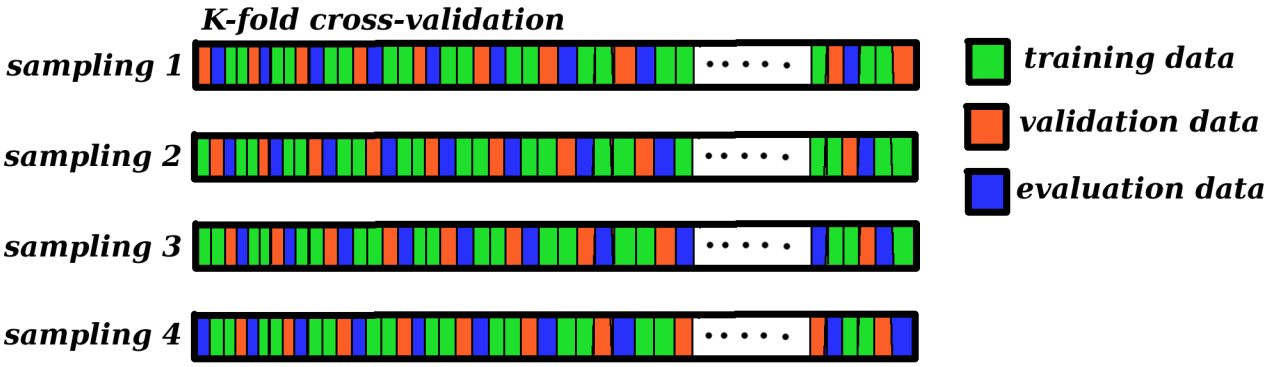

Figure S1: K-fold cross-validation data distribution with 50% of data for training (green), 25% of data for validation (red) and 25% of data for evaluation (blue) for each sampling. Validation and evaluation data are different each time

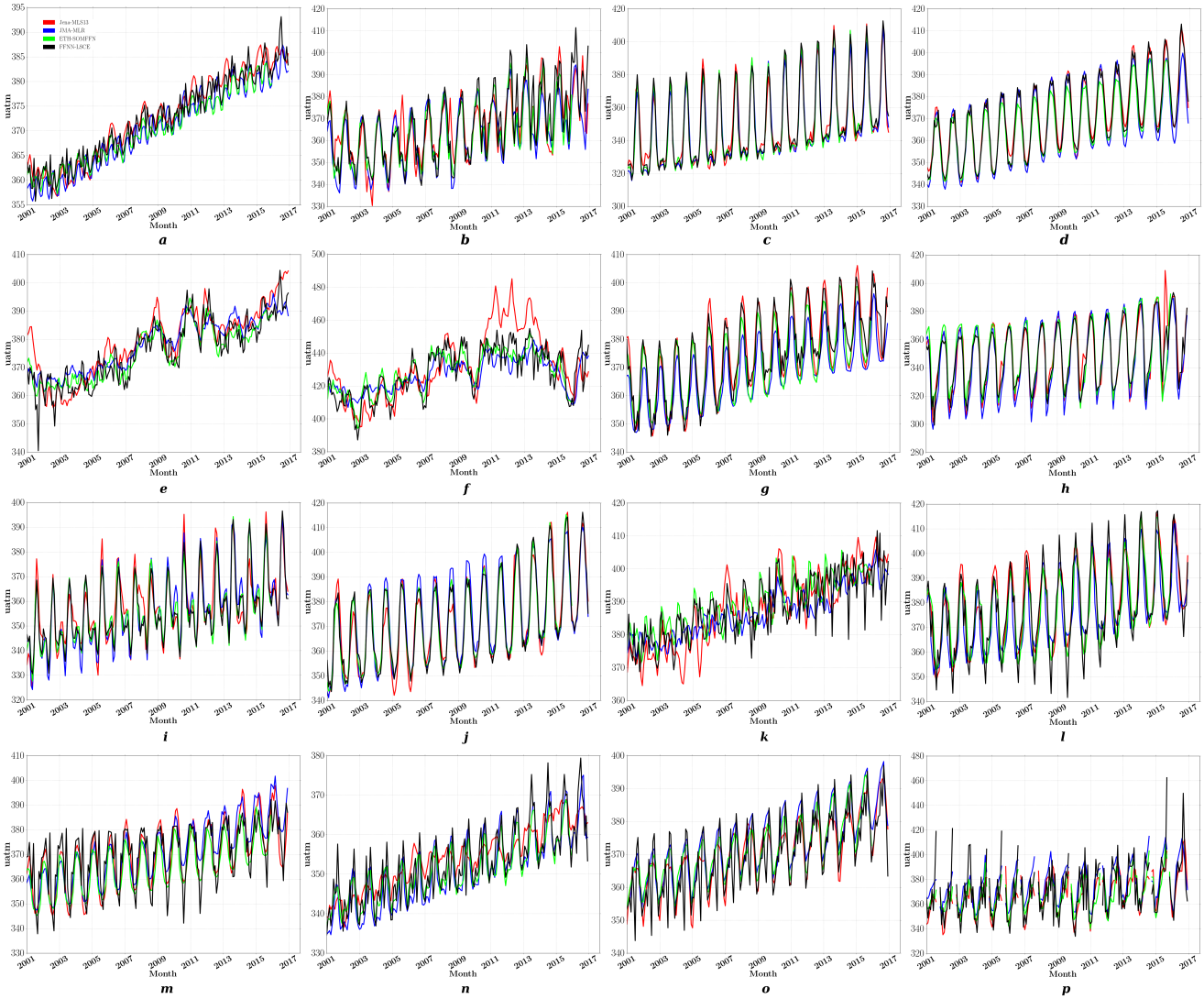

Figure S2: Monthly time series of pCO_2: LSCE-FFNN (black), JMA-MLR (blue), Jena-MLS13 (red), ETH-SOMFFN (green) averaged over: (a) – globe; (b) – biome 2; (c) – biome 3; (d) – biome 4; (e) – biome 5; (f) – biome 6; (g) – biome 7; (h) – biome 9; (i) – biome 10; (j) – biome 11; (k) – biome 12; (l) – biome 13; (m) – biome 14; (n) – biome 15; (o) -biome 16; (p) - biome - 17

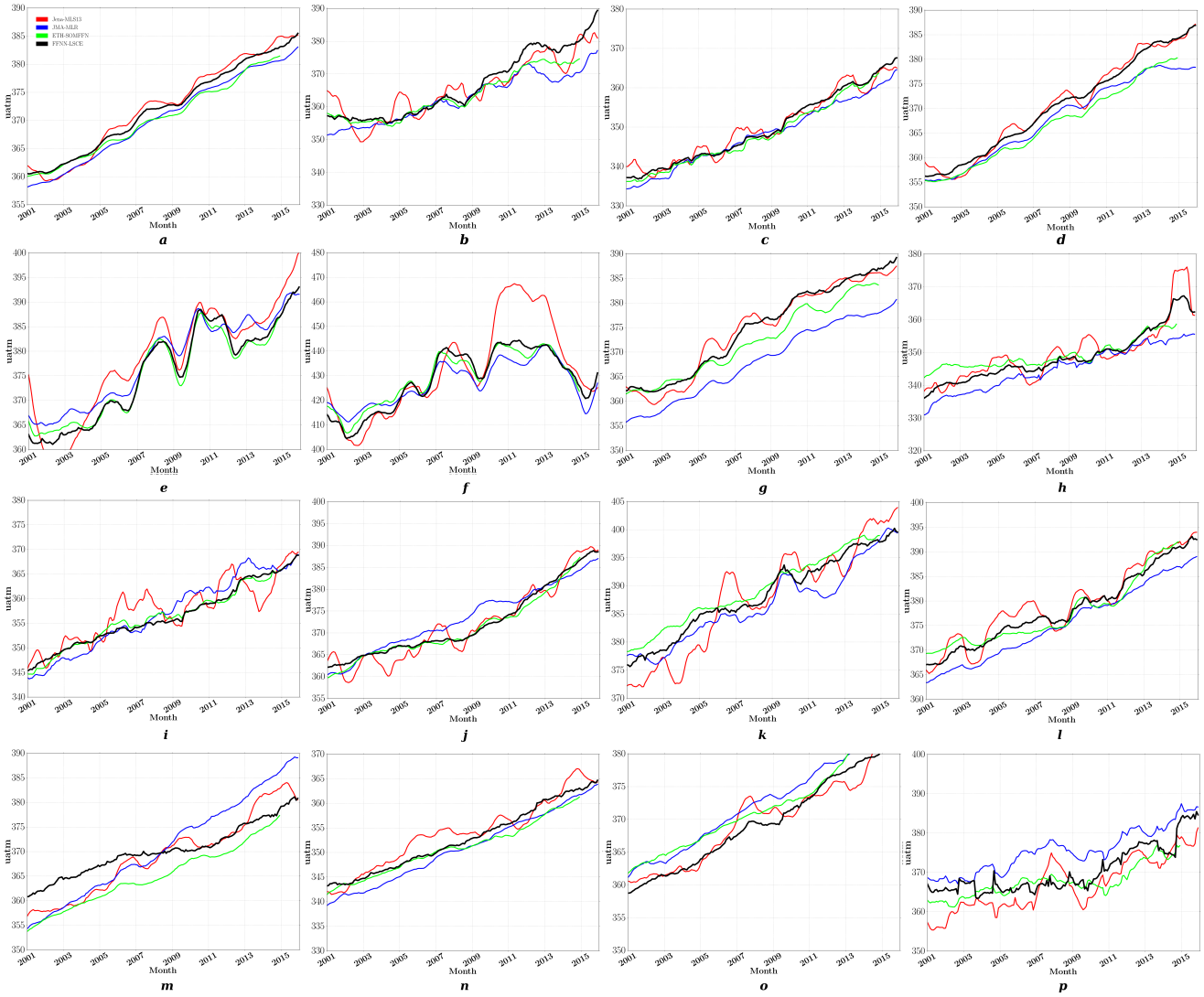

Figure S3: 12-month running mean of pCO_2: LSCE-FFNN (black), JMA-MLR (blue), Jena-MLS13 (red), ETH-SOMFFN (green) averaged over: (a) – globe; (b) – biome 2; (c) – biome 3; (d) – biome 4; (e) – biome 5; (f) – biome 6; (g) – biome 7; (h) – biome 9; (i) – biome 10; (j) – biome 11; (k) – biome 12; (l) – biome 13; (m) – biome 14; (n) – biome 15; (o) – biome 16; (p) – biome - 17

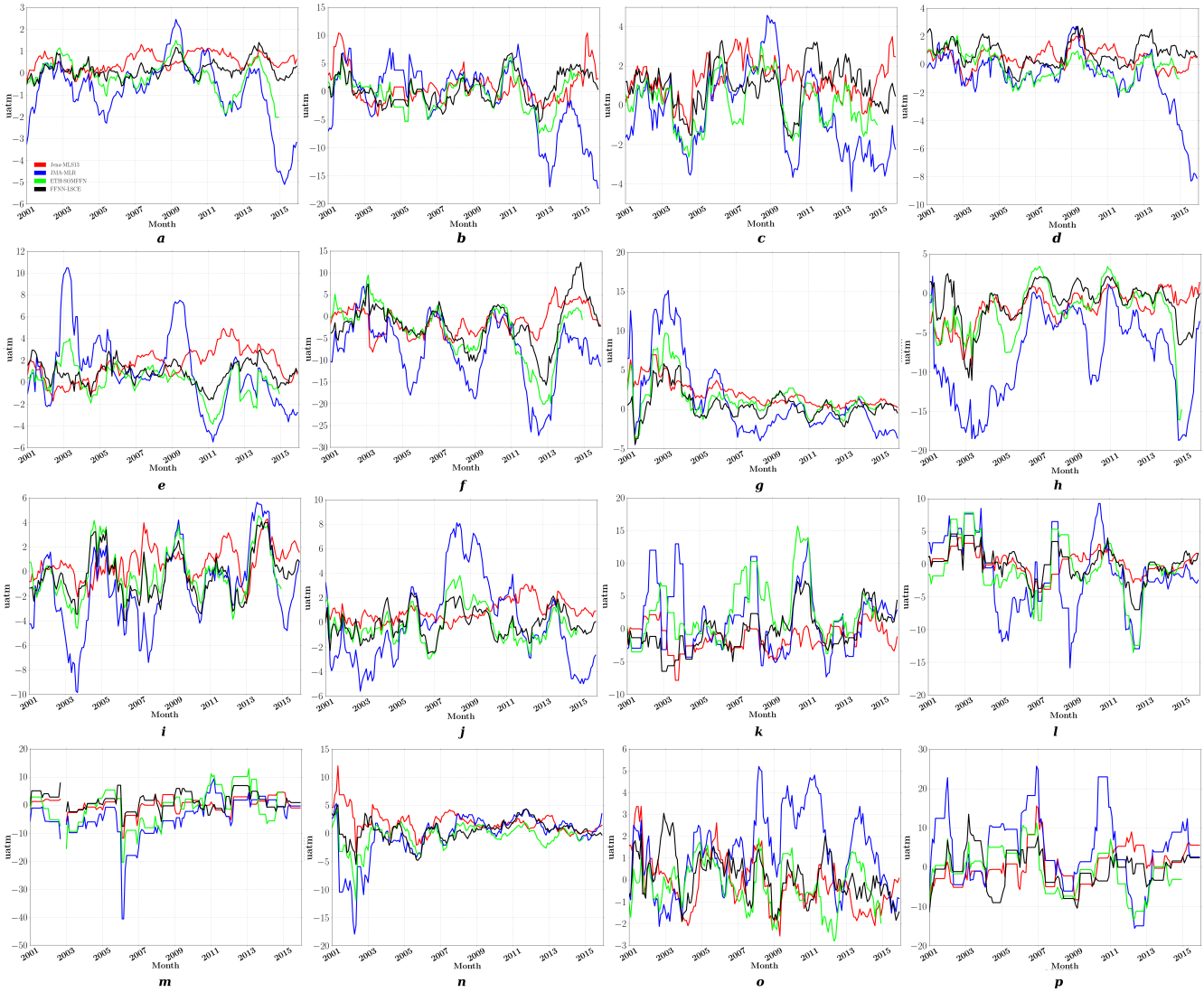

Figure S4: Yearly pCO_2 mismatch (difference of mapping methods and SOCAT data): LSCE-FFNN (black), JMA-MLR (blue), Jena-MLS13 (red), ETH-SOMFFN (green) averaged over: (a) – globe; (b) – biome 2; (c) – biome 3; (d) – biome 4; (e) – biome 5; (f) – biome 6; (g) – biome 7; (h) – biome 9; (i) – biome 10; (j) – biome 11; (k) – biome 12; (l) – biome 13; (m) – biome 14; (n) – biome 15; (o) – biome 16; (p) – biome - 17

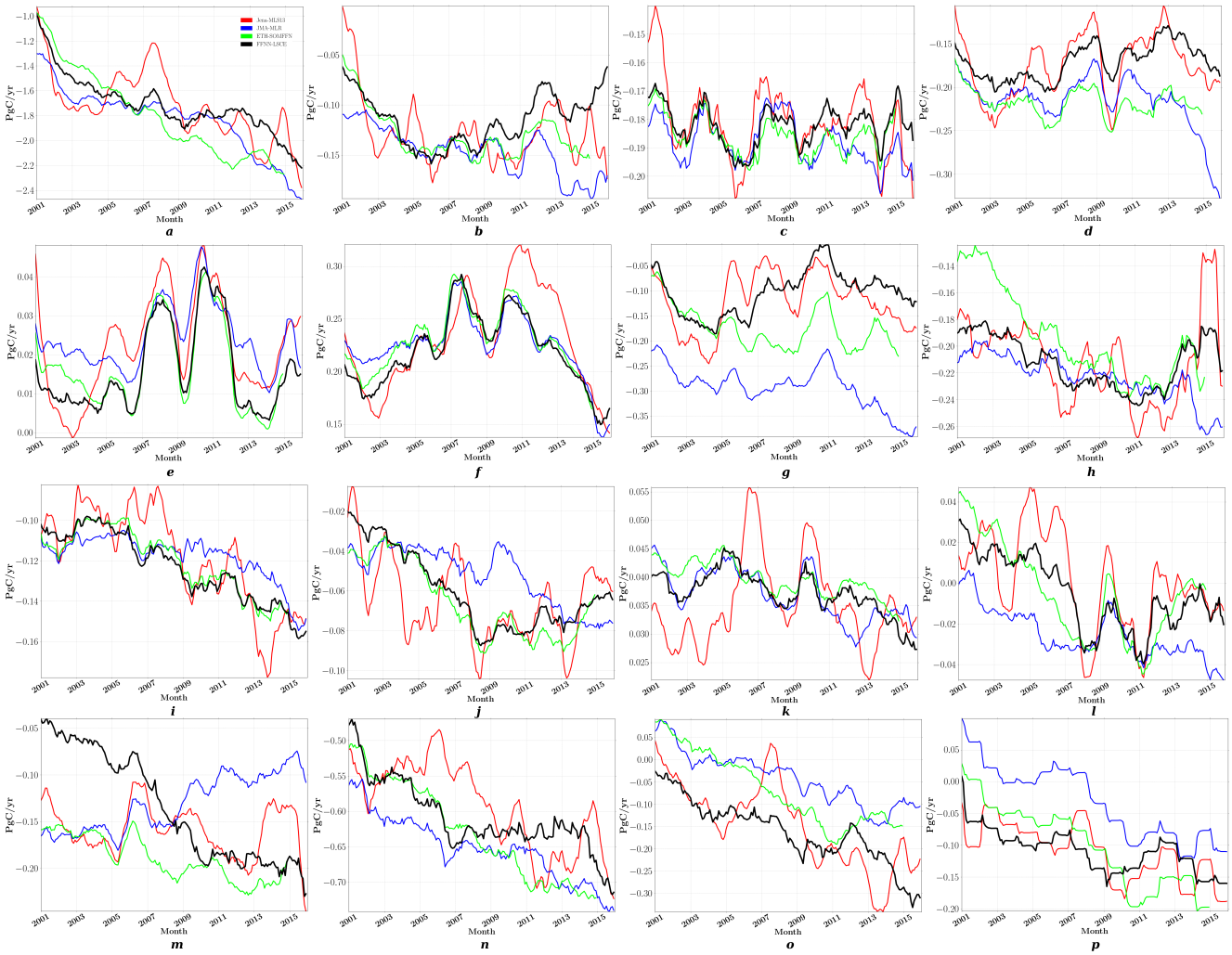

Figure S5: Interannual sea-air CO2 flux (12-month running mean): LSCE-FFNN (black), JMA-MLR (blue), Jena-MLS13 (red), ETH-SOMFFN (green) averaged over: (a) – globe; (b) – biome 2; (c) – biome 3; (d) – biome 4; (e) – biome 5; (f) – biome 6; (g) – biome 7; (h) – biome 9; (i) – biome 10; (j) – biome 11; (k) – biome 12; (l) – biome 13; (m) – biome 14; (n) – biome 15; (o) – biome 16; (p) – biome - 17

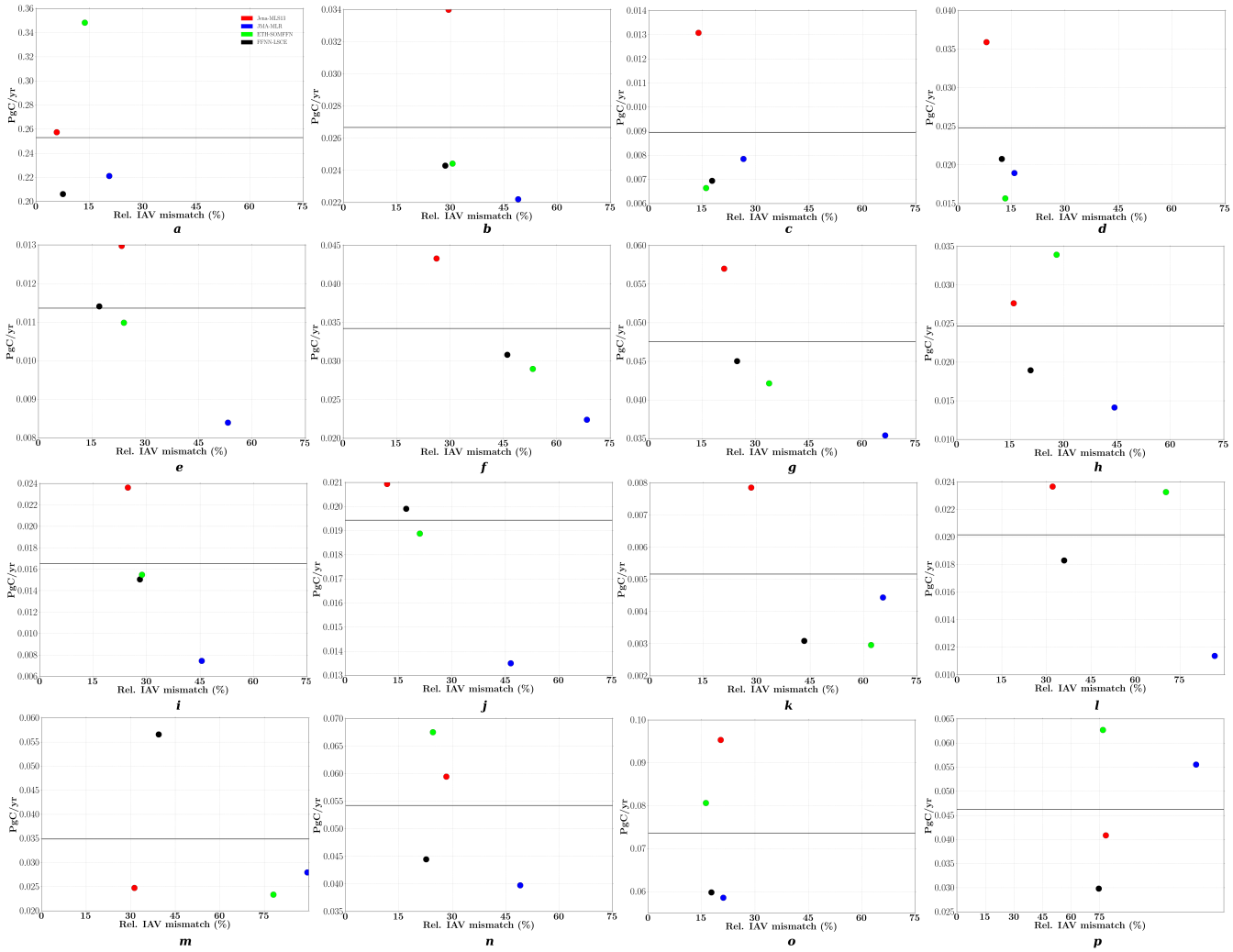

Figure S6: Amplitude of interannual CO_2 flux plotted against the relative IAV mismatch amplitude for LSCE-FFNN (black), JMA-MLR (blue), Jena-MLS13 (red), ETH-SOMFFN (green) averaged over: (a) – globe; (b) – biome 2; (c) – biome 3; (d) – biome 4; (e) – biome 5; (f) – biome 6; (g) – biome 7; (h) – biome 9; (i) – biome 10; (j) – biome 11; (k) – biome 12; (l) – biome 13; (m) – biome 14; (n) – biome 15; (o) – biome 16; (p) – biome – 17. The weighted mean is given as a horizontal lin*e*

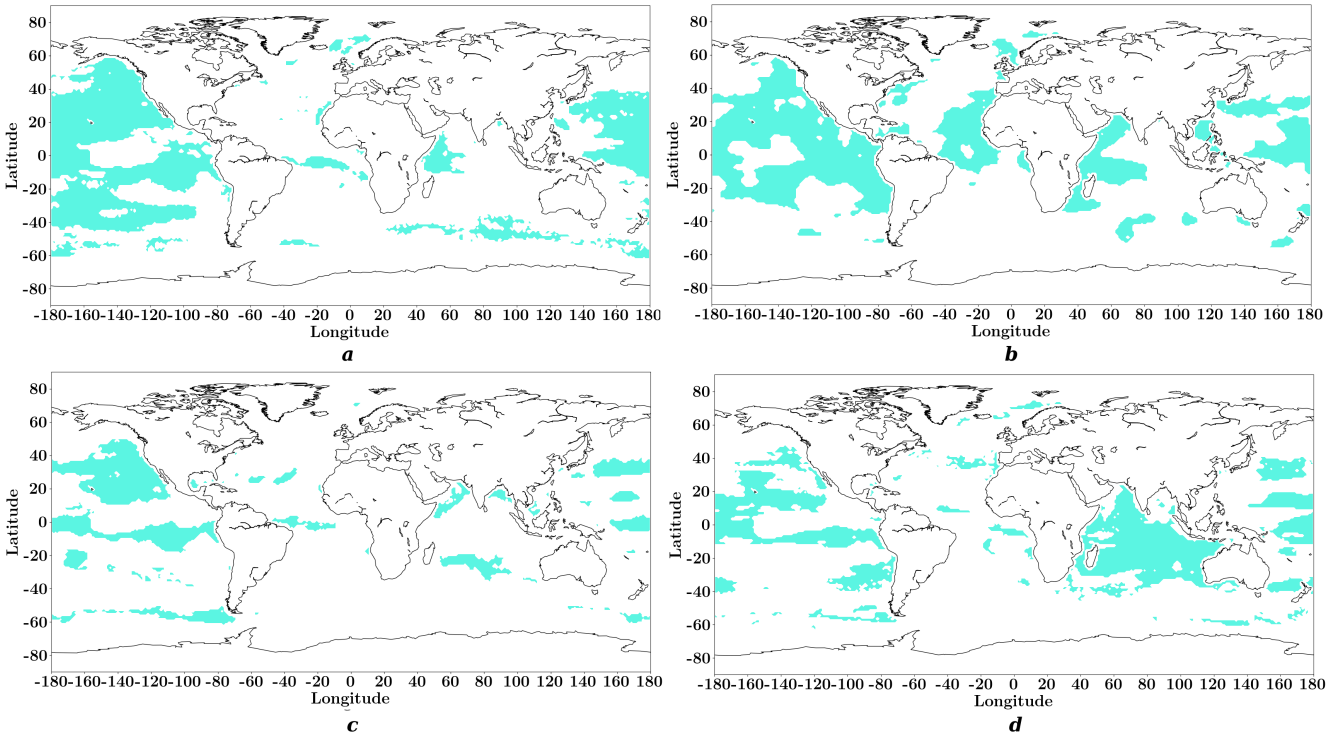

Figure S7: Agreement between sea-air CO_2 flux and atmospheric pCO_2 trends. Blue color indicates the regions where two characteristics have the same sign of linear trend. Atmospheric pCO_2 from Jena inversion and: (a) - LSCE-FFNN; (b) - Jena-NLS13; (c) - ETH-SOMFFN; (d) - JMA-MLR

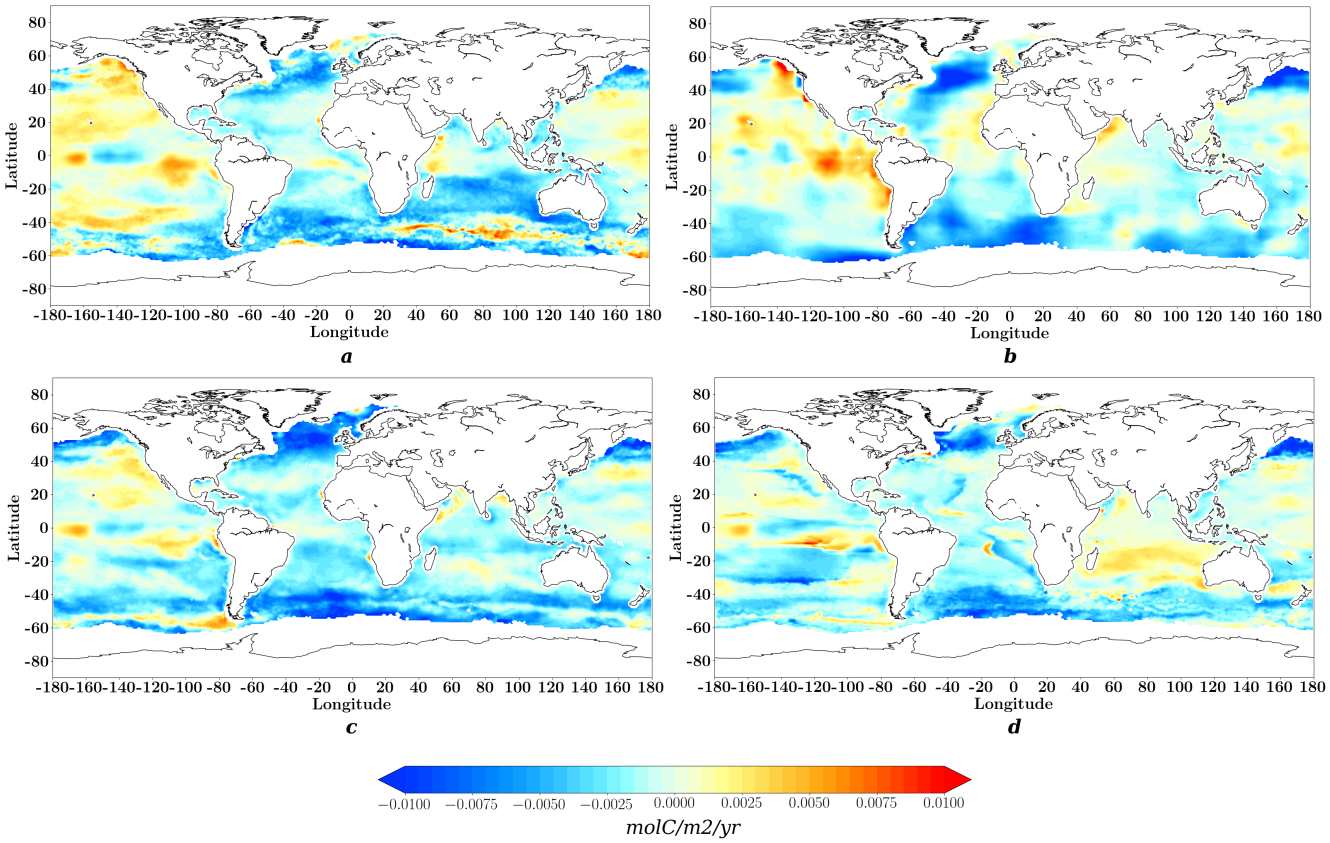

Figure S8: Linear trend in CO_2 fluxes between 2001 and 2015: (a) – LSCE-FFNN; (b) – Jena-MLS13; (c) – ETH-SOMFFN; (d) – JMA-MLR.

Table S1: Bias between reconstructed surface ocean pCO_2 and pCO_2 values from SOCAT v5 database not used in the training algorithm for the period 2001-2016 over the global ocean (except for regions with ice-cover) and for large oceanographic regions.

Model	Latitude boundaries	Bias (µatm)
FFNN Global		-0.24
Arctic	76ºN to 90ºN	-3.95
Subpolar Atlantic	49ºN to 76ºN	-0.23
Subpolar Pacific	49ºN to 76ºN	-0.35
Subtropical Atlantic	18ºN to 49ºN	-0.21
Subtropical Pacific	18ºN to 49ºN	-0.38
Equatorial Atlantic	18ºS to 18ºN	0.22
Equatorial Pacific	18ºS to 18ºN	0.01
South Atlantic	44ºS to 18ºS	-0.71
South Pacific	44ºS to 18ºS	-0.34
Indian Ocean	44S to 30N	-0.39
Southern Ocean	90ºS to 44ºS	-0.38

Table S2: Mean of sea-air CO_2 flux (PgC/yr) over the global ocean and per biomes (Rödenbeck et al., 2015) for 2001-2015.

Biome	LSCE-FFNN	ETH-SOMFFN	Jena-MLS13	JMA-MLR
Globe	-1.55	-1.67	-1.55	-1.74
2	-0.11	-0.12	-0.12	-0.14
3	-0.18	-0.18	-0.18	-0.18
4	-0.17	-0.21	-0.17	-0.21
5	0.016	0.017	0.023	0.024
6	0.22	0.22	0.23	0.23
7	-0.1	-0.17	-0.11	-0.28
9	-0.18	-0.16	-0.17	-0.18
10	-0.12	-0.12	-0.11	-0.11
11	-0.06	-0.06	-0.06	-0.05
12	0.03	0.04	0.03	0.03
13	-0.004	-0.004	0.0016	-0.024
14	-0.13	-0.18	-0.15	-0.13
15	-0.6	-0.63	-0.58	-0.65
16	-0.14	-0.05	-0.13	-0.04

17	-0.025	-0.024	-0.029	-0.0006

```
#code to reconstruct the pCO2 climatology

#!/usr/bin/env python

# -*- coding: utf-8 -*-

import numpy as npy
import math
import keras.callbacks
from scipy.io import netcdf as nc
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD, RMSprop

# read a simple 1d field (timserie for example)
def readnc_1d(ncfile,varname):
        fid = nc.netcdf_file(ncfile, 'r')
        out = npy.array(fid.variables[varname][:]).squeeze()
        fid.close()
        return out

# read a simple 3d field
def readnc(ncfile,varname):
        fid = nc.netcdf_file(ncfile, 'r')
        out = npy.array(fid.variables[varname][:,:,:]).squeeze()
        fid.close()
        return out

## Write 2d field in a regular lon/lat file
## fill with lon,lat and field
def write_2d_reg_file(ncfile,lon_array,lat_array,time,var,varname):
            try:
            from netCDF4 import Dataset
            except:
                    print 'netCDF4 not available' ; exit()

        fid = Dataset(ncfile, 'w', format='NETCDF3_CLASSIC')
        fid.description = 'file created by Sommer A (anna.sommer.lab@gmail.com)'
        # dimensions
        fid.createDimension('lat', lat_array.shape[0])
        fid.createDimension('lon', lon_array.shape[0])
        fid.createDimension('time', None)
        # variables
        latitudes  = fid.createVariable('lat', 'f4', ('lat',))
        longitudes = fid.createVariable('lon', 'f4', ('lon',))
        times      = fid.createVariable('time', 'f4', ('time',))
        variable   = fid.createVariable(varname, 'f8', ('time','lat','lon',))
        # data
        latitudes[:]    = lat_array
```

```
        longitudes[:]   = lon_array
        times[:]        = time
        variable[0,:,:] = var
        # close
        fid.close()
        return None

#reconstruction is done for all 12 month
month_estimate = 'all'
t = 0

if month_estimate == 'all':

 for number_month in npy.arange(1,13,1):
    #read data climatology data of predictors and interpolated data (all on the 1x1 grid)
    if number_month < 10:
       directory = '/home/biomac2/sommer/Data/Taka_Inerpolated/Taka_Interp_0' + str(number_month)
+ '.nc'
       directory1_SSH = '/home/biomac2/sommer/Data/Data_Predict_Taka/SSH_Clim_predict_0'+
str(number_month) + '.nc'
       directory1_SSS = '/home/biomac2/sommer/Data/Data_Predict_Taka/SSS_Clim_predict_0' +
str(number_month) + '.nc'
       directory1_SST = '/home/biomac2/sommer/Data/Data_Predict_Taka/SST_Clim_predict_0' +
str(number_month) + '.nc'
       directory2 = '/home/biomac2/sommer/Data/Data_Predict_Taka/CHL_Clim_predict_0' +
str(number_month) + '.nc'
       directory4 = '/home/biomac2/sommer/Data/Data_Predict_Taka/MLD_Clim_predict_0' +
str(number_month) + '.nc'
       directory5 = '/home/biomac2/sommer/Data/Sea_Ice_Clim/Sea_Ice_' + str(number_month) + '.nc'
    else:
       directory = '/home/biomac2/sommer/Data/Taka_Inerpolated/Taka_Interp_' + str(number_month) +
'.nc'
       directory1_SSH = '/home/biomac2/sommer/Data/Data_Predict_Taka/SSH_Clim_predict_'+
str(number_month) + '.nc'
       directory1_SSS = '/home/biomac2/sommer/Data/Data_Predict_Taka/SSS_Clim_predict_' +
str(number_month) + '.nc'
       directory1_SST = '/home/biomac2/sommer/Data/Data_Predict_Taka/SST_Clim_predict_' +
str(number_month) + '.nc'
       directory2 = '/home/biomac2/sommer/Data/Data_Predict_Taka/CHL_Clim_predict_' +
str(number_month) + '.nc'
       directory4 = '/home/biomac2/sommer/Data/Data_Predict_Taka/MLD_Clim_predict_' +
str(number_month) + '.nc'
       directory5 = '/home/biomac2/sommer/Data/Sea_Ice_Clim/Sea_Ice_' + str(number_month) + '.nc'

    if t == 0:
       lon_SOCAT = readnc_1d(directory2,'lon')
       lat_SOCAT = readnc_1d(directory2,'lat')
       lon_SST = readnc_1d(directory1_SST,'lon')
       lat_SST = readnc_1d(directory1_SST,'lat')
```

```
    salinity_Clim = npy.zeros((12,len(lat_SST),len(lon_SST)))
    temperature_Clim = npy.zeros((12,len(lat_SST),len(lon_SST)))
    ssh_Clim = npy.zeros((12,len(lat_SST),len(lon_SST)))
    CHL_Clim = npy.zeros((12,len(lat_SST),len(lon_SST)))
    CHL_Clim_test = npy.zeros((12,len(lat_SOCAT),len(lon_SOCAT)))
    pCO2_Clim = npy.zeros((12,len(lat_SST),len(lon_SST)))
    MLD_Clim = npy.zeros((12,len(lat_SST),len(lon_SST)))
    Sea_Ice = npy.zeros((12,len(lat_SST),len(lon_SST)))
    Sea_Ice_test = npy.zeros((12,len(lat_SOCAT),len(lon_SOCAT)))

    a3 = int(lat_SST[0] - lat_SOCAT[0])
    lat_SOCAT = lat_SOCAT[a3:]


  salinity_Clim[t,:,:] = readnc(directory1_SSS,'SSS')
  temperature_Clim[t,:,:] = readnc(directory1_SST,'SST')
  ssh_Clim[t,:,:] = readnc(directory1_SSH,'SSH')
  CHL_Clim_test[t,:,:] = readnc(directory2,'CHL')
  pCO2_Clim[t,:,:] = readnc(directory,'pCO2')
  MLD_Clim[t,:,:] = readnc(directory4,'MLD')
  Sea_Ice_test[t,:,:] = readnc(directory5,'mask')

  CHL_Clim[t,:,:] = CHL_Clim_test[t,a3:,:]
  Sea_Ice[t,:,:] = Sea_Ice_test[t,a3:,:]

  #application of ice mask
  for i in npy. arange(0,len(lat_SOCAT),1):
     for j in npy. arange(0,len(lon_SOCAT),1):
        if math.isnan(Sea_Ice[t,i,j]) == True:
          salinity_Clim[t,i,j] = npy.nan
          temperature_Clim[t,i,j] = npy.nan
          ssh_Clim[t,i,j] = npy.nan
          pCO2_Clim[t,i,j] = npy.nan
          CHL_Clim[t,i,j] = npy.nan

  #add CHL data as "0" in the region where there are not CHL measurements
  for i in npy. arange(0,len(lat_SOCAT),1):
     for j in npy. arange(0,len(lon_SOCAT),1):
        if math.isnan(CHL_Clim[t,i,j]) == True and math.isnan(salinity_Clim[t,i,j]) == False and
math.isnan(temperature_Clim[t,i,j]) == False:
          CHL_Clim[t,i,j] = 0.

  for i in npy. arange(0,len(lat_SOCAT),1):
     for j in npy. arange(0,len(lon_SOCAT),1):
        if math.isnan(pCO2_Clim[t,i,j]) == True or math.isnan(salinity_Clim[t,i,j]) == True or
math.isnan(temperature_Clim[t,i,j]) == True or math.isnan(ssh_Clim[t,i,j]) == True or
math.isnan(CHL_Clim[t,i,j]) == True or math.isnan(MLD_Clim[t,i,j]) == True:
          salinity_Clim[t,i,j] = npy.nan
          temperature_Clim[t,i,j] = npy.nan
```

```
            ssh_Clim[t,i,j] = npy.nan
            CHL_Clim[t,i,j] = npy.nan
            pCO2_Clim[t,i,j] = npy.nan
            MLD_Clim[t,i,j] = npy.nan

    #transfer of CHL and MLD to log
    if t == 0:
       CHL_log = npy.zeros((len(CHL_Clim),len(lat_SOCAT),len(lon_SOCAT)))
       MLD_log = npy.zeros((len(CHL_Clim),len(lat_SOCAT),len(lon_SOCAT)))

    for i in npy. arange(0,len(lat_SOCAT),1):
        for j in npy. arange(0,len(lon_SOCAT),1):
            if CHL_Clim[t,i,j] <> 0.:
               CHL_log[t,i,j] = npy.log10(CHL_Clim[t,i,j])

    for i in npy. arange(0,len(lat_SOCAT),1):
        for j in npy. arange(0,len(lon_SOCAT),1):
            MLD_log[t,i,j] = npy.log10(MLD_Clim[t,i,j])

    t = t + 1

#!!!!!normalisation of data used as predictors in training algorithm!!!!!!!!!!
salinity_region = (salinity_Clim - npy.nanmean(salinity_Clim))/npy.nanstd(salinity_Clim)
temperature_region = (temperature_Clim -
npy.nanmean(temperature_Clim))/npy.nanstd(temperature_Clim)
ssh_region = (ssh_Clim - npy.nanmean(ssh_Clim))/npy.nanstd(ssh_Clim)
CHL_region = (CHL_log)/npy.nanstd(CHL_log)
pCO2_region = (pCO2_Clim - npy.nanmean(pCO2_Clim))/npy.nanstd(pCO2_Clim)
MLD_region = (MLD_log - npy.nanmean(MLD_log))/npy.nanstd(MLD_log)

lat_region = lat_SOCAT
lon_region = lon_SOCAT

pCO2_list = npy.zeros(100000000)
salinity_list = npy.zeros(100000000)
temperature_list = npy.zeros(100000000)
ssh_list = npy.zeros(100000000)
CHL_list = npy.zeros(100000000)
MLD_list = npy.zeros(100000000)
lat_list = npy.zeros(100000000)
lon_list1 = npy.zeros(100000000)
lon_list2 = npy.zeros(100000000)

lat_degree = npy.zeros(100000000)
lon_degree = npy.zeros(100000000)

#data for prediction

data_reconstr_1 = npy.zeros((100000,8))
```

```
data_reconstr_2 = npy.zeros((100000,8))
data_reconstr_3 = npy.zeros((100000,8))
data_reconstr_4 = npy.zeros((100000,8))
data_reconstr_5 = npy.zeros((100000,8))
data_reconstr_6 = npy.zeros((100000,8))
data_reconstr_7 = npy.zeros((100000,8))
data_reconstr_8 = npy.zeros((100000,8))
data_reconstr_9 = npy.zeros((100000,8))
data_reconstr_10 = npy.zeros((100000,8))
data_reconstr_11 = npy.zeros((100000,8))
data_reconstr_12 = npy.zeros((100000,8))

p = 0

for t in npy.arange(0,12,1):
    k = p
    for i in npy.arange(0,len(lat_region),1):
        for j in npy.arange(0,len(lon_region),1):
            if math.isnan(pCO2_region[t,i,j]) == False and math.isnan(salinity_region[t,i,j]) == False and
math.isnan(temperature_region[t,i,j]) == False and math.isnan(ssh_region[t,i,j]) == False and
math.isnan(CHL_region[t,i,j]) == False and math.isnan(MLD_region[t,i,j]) == False:
                pCO2_list[p] = pCO2_region[t,i,j]
                salinity_list[p] = salinity_region[t,i,j]
                temperature_list[p] = temperature_region[t,i,j]
                ssh_list[p] = ssh_region[t,i,j]
                CHL_list[p] = CHL_region[t,i,j]
                MLD_list[p] = MLD_region[t,i,j]
                lat_list[p] = npy.sin(lat_region[i] * npy.pi/180.)
                lon_list1[p] = npy.cos(lon_region[i] * npy.pi/180.)
                lon_list2[p] = npy.sin(lon_region[i] * npy.pi/180.)
                lat_degree[p] = lat_region[i]
                lon_degree[p] = lon_region[i]
                p = p + 1
    if t == 0:
        data_reconstr_1 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
    if t == 1:
        data_reconstr_2 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
    if t == 2:
        data_reconstr_3 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
    if t == 3:
        data_reconstr_4 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
```

```
    if t == 4:
        data_reconstr_5 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
    if t == 5:
        data_reconstr_6 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
    if t == 6:
        data_reconstr_7 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
    if t == 7:
        data_reconstr_8 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
    if t == 8:
        data_reconstr_9 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
    if t == 9:
        data_reconstr_10 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
    if t == 10:
        data_reconstr_11 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))
    if t == 11:
        data_reconstr_12 =
npy.column_stack((salinity_list[k:p],temperature_list[k:p],ssh_list[k:p],CHL_list[k:p],MLD_list[k:p],la
t_list[k:p],lon_list1[k:p],lon_list2[k:p]))

pCO2_list = pCO2_list[:p]
salinity_list = salinity_list[:p]
temperature_list = temperature_list[:p]
ssh_list = ssh_list[:p]
CHL_list = CHL_list[:p]
MLD_list = MLD_list[:p]
lat_list = lat_list[:p]
lon_list1 = lon_list1[:p]
lon_list2 = lon_list2[:p]
lat_degree = lat_degree[:p]
lon_degree = lon_degree[:p]

data_predictors =
npy.column_stack((salinity_list,temperature_list,ssh_list,CHL_list,MLD_list,lat_list,lon_list1,lon_list2)
)
```

```
index1 = npy.zeros(len(pCO2_list))

p = 0
for i in npy.arange(0,len(pCO2_list),4):
    index1[p] = i
    p = p + 1

index1 = index1[:p]

data_train1 = npy.delete(data_predictors, index1, axis = 0)
lat_list_test = npy.delete(lat_list, index1)
lat_degree_test = npy.delete(lat_degree, index1)
lon_degree_test = npy.delete(lon_degree, index1)
pCO2_list_train1 = npy.delete(pCO2_list, index1)

#data for training, evaluation and validation

data_eval = npy.zeros((len(index1),8))
pCO2_list_eval = npy.zeros(len(index1))

for i in npy.arange(0,len(index1),1):
    data_eval[i,:] = data_predictors[int(index1[i]),:]
    pCO2_list_eval[i] = pCO2_list[int(index1[i])]

index2 = npy.zeros(len(pCO2_list_train1))

p = 0
for i in npy.arange(0,len(pCO2_list_train1),3):
    index2[p] = i
    p = p + 1

index2 = index2[:p]

data_train = npy.delete(data_train1, index2, axis = 0)
pCO2_list_train = npy.delete(pCO2_list_train1, index2)

data_val = npy.zeros((len(index2),8))
pCO2_list_val = npy.zeros(len(index2))
lat_degree_val = npy.zeros(len(index2))
lon_degree_val = npy.zeros(len(index2))

for i in npy.arange(0,len(index2),1):
    data_val[i,:] = data_train1[int(index2[i]),:]
    pCO2_list_val[i] = pCO2_list_train1[int(index2[i])]
    lat_degree_val[i] = lat_degree_test[int(index2[i])]
    lon_degree_val[i] = lon_degree_test[int(index2[i])]

print 'Size of data', data_predictors.shape, data_eval.shape, data_val.shape
```

```
#!!!!!!!!!CREATION OF MODEL!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
earlyStopping=keras.callbacks.EarlyStopping(monitor='val_loss', patience=30, verbose=0,
mode='auto')

model = Sequential()
model.add(Dense(20, input_dim=8, init='glorot_uniform'))
model.add(Activation('tanh'))
model.add(Dense(30, input_dim=20, init='glorot_uniform'))
model.add(Activation('tanh'))
model.add(Dense(25, input_dim=30, init='glorot_uniform'))
model.add(Activation('tanh'))
model.add(Dense(10, input_dim=25, init='glorot_uniform'))
model.add(Activation('tanh'))
model.add(Dense(1, init='glorot_uniform'))
model.add(Activation('linear'))
sgd = SGD(lr=0.01, decay=0.0, momentum=0.0, nesterov=False)
RMSprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
L = model.compile(loss='mse',optimizer=RMSprop)
print L
H = model.fit(data_train,
pCO2_list_train,nb_epoch=1600,callbacks=[earlyStopping],batch_size=20,validation_data=(data_val,p
CO2_list_val))
score = model.evaluate(data_eval, pCO2_list_eval,show_accuracy = True, batch_size=20)
print model.get_weights()

#reconstruction of climatology month by month
preds1 = model.predict(data_reconstr_1)
preds2 = model.predict(data_reconstr_2)
preds3 = model.predict(data_reconstr_3)
preds4 = model.predict(data_reconstr_4)
preds5 = model.predict(data_reconstr_5)
preds6 = model.predict(data_reconstr_6)
preds7 = model.predict(data_reconstr_7)
preds8 = model.predict(data_reconstr_8)
preds9 = model.predict(data_reconstr_9)
preds10 = model.predict(data_reconstr_10)
preds11 = model.predict(data_reconstr_11)
preds12 = model.predict(data_reconstr_12)

lat = lat_SOCAT
lon = lon_SOCAT
pCO2_matrix = npy.zeros((12,len(lat),len(lon)))

for t in npy.arange(0,12,1):
    p = 0
    preds = preds_list[t]
    for i in npy.arange(0,len(lat),1):
        for j in npy.arange(0,len(lon),1):
```

```
            if math.isnan(salinity_Clim[t,i,j]) == False and math.isnan(temperature_Clim[t,i,j]) == False
and math.isnan(ssh_Clim[t,i,j]) == False and math.isnan(CHL_Clim[t,i,j]) == False and
math.isnan(MLD_Clim[t,i,j]) == False:
              if  t == 0:
                 pCO2_matrix[t,i,j] = preds1[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 1:
                 pCO2_matrix[t,i,j] = preds2[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 2:
                 pCO2_matrix[t,i,j] = preds3[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 3:
                 pCO2_matrix[t,i,j] = preds4[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 4:
                 pCO2_matrix[t,i,j] = preds5[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 5:
                 pCO2_matrix[t,i,j] = preds6[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 6:
                 pCO2_matrix[t,i,j] = preds7[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 7:
                 pCO2_matrix[t,i,j] = preds8[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 8:
                 pCO2_matrix[t,i,j] = preds9[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 9:
                 pCO2_matrix[t,i,j] = preds10[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 10:
                 pCO2_matrix[t,i,j] = preds11[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              if  t == 11:
                 pCO2_matrix[t,i,j] = preds12[p,0] * npy.nanstd(pCO2_Clim) + npy.nanmean(pCO2_Clim)
              p = p + 1
            else:
              pCO2_matrix[t,i,j] = npy.nan

   #write reconstructed pCO2 in NetCDF
   time = 0
   if t+1 < 10:
     a = '0' + str(t+1)
     print a
   else:
     a = str(t+1)

   write_2d_reg_file('/home/users/asommer/Figure_MLP_sknn/Clim_MLP_TF_'
+a+'.nc',lon_SOCAT,lat_SOCAT,time,pCO2_matrix[t,:,:],'pCO2')


   #estimation of accuracy compared with original pCO2 field
   print '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
   print 'Total'
   print 'Correlation', numpy.corrcoef(pCO2_Clim[t,:,:],pCO2_matrix[t,:,:])
   print 'RMS', npy.sqrt(npy.nanmean((pCO2_Clim[t,:,:]-pCO2_matrix[t,:,:])**2))
```

```
    s1 = 0
    s_ref = 0
    s_data = 0
    for i in range(0,len(lat),1):
        for j in range(0,len(lon),1):
            if math.isnan(pCO2_Clim[t,i,j]) == False and math.isnan(pCO2_matrix[t,i,j]) == False:
                x = 6357 * abs(math.cos(npy.pi/180.*lat[i]) * npy.pi/180. * (lon[j+1] - lon[j]))
                y = 6357 * abs(npy.pi/180. * (lat[i+1] - lat[i]))
                s1 = s1 + x * y
                s_ref = s_ref + x * y * pCO2_Clim[t,i,j]
                s_data = s_data + x * y * pCO2_matrix[t,i,j]

    s_ref1 = s_ref/s1
    s_data1 = s_data/s1
    data1 = pCO2_Clim[t,:,:] - s_ref1
    data2 = pCO2_matrix[t,:,:] - s_data1

    s = 0
    s_RMS = 0
    s1 = 0
    s2 = 0
    for i in range(lat.shape[0]-1):
        for j in range(lon.shape[0]-1):
            if math.isnan(pCO2_Clim[t,i,j]) == False and math.isnan(pCO2_matrix[t,i,j]) == False:
                s = s + data1[i,j]*data2[i,j]
                s_RMS = s_RMS + (data1[i,j] - data2[i,j])**2
                s1 = s1 + data1[i,j]*data1[i,j]
                s2 = s2 + data2[i,j]*data2[i,j]

    data3 = s/(sqrt(s1)*sqrt(s2))

    print 'Spatial Correlation',data3
    print data3**2

#estimation of accuracy by comparing with data that did not participate in training algorithm
preds_conc =
npy.concatenate((preds1[:,0],preds2[:,0],preds3[:,0],preds4[:,0],preds5[:,0],preds6[:,0],preds7[:,0],preds
8[:,0],preds9[:,0],preds10[:,0],preds11[:,0],preds12[:,0]),axis = 0)

pCO2_test1 = preds_conc
pCO2_test2 = npy.zeros(len(index1))
for i in npy.arange(0,len(index1),1):
    pCO2_test2[i] = pCO2_test1[int(index1[i])]

print '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
print'Validation'
print 'Personr',scipy.stats.pearsonr(pCO2_list_eval,pCO2_test2)
print 'RMS', npy.sqrt(npy.nanmean((pCO2_list_eval-pCO2_test2)**2))
slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(pCO2_list_eval,pCO2_test2)
```

```
print 'R2', r_value**2
print 'Bias', npy.nanmean(pCO2_list_eval) - npy.nanmean(pCO2_test2)

s1 = 0
s_ref = 0
s_data = 0
for i in range(0,len(lat_degree_val)-1,1):
    if math.isnan(pCO2_list_val[i]) == False and math.isnan(pCO2_test2[i]) == False:
       x = 6357 * abs(math.cos(npy.pi/180.*lat_degree_val[i]) * npy.pi/180. * (lon_degree_val[i] + 1. -
lon_degree_val[i]))
       y = 6357 * abs(npy.pi/180. * (lat_degree_val[i] + 1. - lat_degree_val[i]))
       s1 = s1 + x * y
       s_ref = s_ref + x * y * pCO2_list_val[i]
       s_data = s_data + x * y * pCO2_test2[i]

s_ref1 = s_ref/s1
s_data1 = s_data/s1
data11 = pCO2_list_val - s_ref1
data22 = pCO2_test2 - s_data1

s = 0
s_RMS = 0
s1 = 0
s2 = 0
for i in range(lat_degree_val.shape[0]-1):
    if math.isnan(pCO2_list_val[i]) == False and math.isnan(pCO2_test2[i]) == False:
       s = s + data11[i]*data22[i]
       s_RMS = s_RMS + (data11[i] - data22[i])**2
       s1 = s1 + data11[i]*data11[i]
       s2 = s2 + data22[i]*data22[i]

data3 = s/(sqrt(s1)*sqrt(s2))

print 'Spatial Correlation',data3
print data3**2
```

```
#code provide pCO2 reconstruction based on Feed-Forward Neural Network using Keras, Python
#model reconstruct pCO2 anomaly, pCO2 climatology added before write results in NetCDF

#!/usr/bin/env python

# -*- coding: utf-8 -*-

import numpy as npy
import math
import keras.callbacks
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD, RMSprop
from scipy.io import netcdf as nc

# read a simple 1d field (timserie for example)
def readnc_1d(ncfile,varname):
        fid = nc.netcdf_file(ncfile, 'r')
        out = npy.array(fid.variables[varname][:]).squeeze()
        fid.close()
        return out

# read a simple 3d field
def readnc(ncfile,varname):
        fid = nc.netcdf_file(ncfile, 'r')
        out = npy.array(fid.variables[varname][:,:,:]).squeeze()
        fid.close()
        return out

## Write 2d field in a regular lon/lat file
## fill with lon,lat and field
def write_2d_reg_file(ncfile,lon_array,lat_array,time,var,varname):
            try:
            from netCDF4 import Dataset
            except:
                    print 'netCDF4 not available' ; exit()

        fid = Dataset(ncfile, 'w', format='NETCDF3_CLASSIC')
        fid.description = 'file created by Sommer A (anna.sommer.lab@gmail.com)'
        # dimensions
        fid.createDimension('lat', lat_array.shape[0])
        fid.createDimension('lon', lon_array.shape[0])
        fid.createDimension('time', None)
        # variables
        latitudes  = fid.createVariable('lat', 'f4', ('lat',))
        longitudes = fid.createVariable('lon', 'f4', ('lon',))
        times      = fid.createVariable('time', 'f4', ('time',))
        variable   = fid.createVariable(varname, 'f8', ('time','lat','lon',))
        # data
```

```
    latitudes[:]    = lat_array
    longitudes[:]   = lon_array
    times[:]        = time
    variable[0,:,:] = var
    # close
    fid.close()
    return None


#read SOCAT pCO2 tarjet data and data used as predictors localized at SOCAT positions
direct0 =
"/home/biomac2/sommer/Data/pCO2_SOCAT_Estimated_New/SOCAT_pCO2_200101_201612.nc"
direct1_SSS =
"/home/biomac2/sommer/Data/SST_SSH_SSS_SOCAT_New/SOCAT_SSS_200101_201612.nc"
direct1_SST =
"/home/biomac2/sommer/Data/SST_SSH_SSS_SOCAT_New/SOCAT_SST_200101_201612.nc"
direct1_SSH =
"/home/biomac2/sommer/Data/SST_SSH_SSS_SOCAT_New/SOCAT_SSH_200101_201612.nc"
direct2 =
"/home/biomac2/sommer/Data/Chlorophyll_SOCAT_New/SOCAT_CHL_200101_201612.nc"
direct3 = "/home/biomac2/sommer/Data/MLD_SOCAT_New/SOCAT_MLD_200101_201612.nc"
direct4 =
"/home/biomac2/sommer/Data/Clim_MLP_TF_SOCAT_pos/Clim2_MLP_SOCAT_pos_200101_2016
12.nc"
direct5 = "/home/biomac2/sommer/Data/aCO2_SOCAT_New/SOCAT_aCO2_200101_201612.nc"

lon_SOCAT = readnc_1d(direct0,'lon')
lat_SOCAT = readnc_1d(direct0,'lat')

pCO2_SOCAT = readnc(direct0,'pCO2')
salinity_SOCAT = readnc(direct1_SSS,'SSS')
temperature_SOCAT = readnc(direct1_SST,'SST')
ssh_SOCAT = readnc(direct1_SSH,'SSH')
CHL_SOCAT = readnc(direct2,'CHL')
MLD_SOCAT = readnc(direct3,'MLD')
pCO2_Taka_SOCAT = readnc(direct4,'pCO2')
pCO2_atm_SOCAT = readnc(direct5,'aCO2')

#read climatology data for predictors
directory1_SSH =
"/home/biomac2/sommer/Data/Data_Predict_Taka_New/SSH_Clim_predict_2001_2016.nc"
directory1_SSS =
"/home/biomac2/sommer/Data/Data_Predict_Taka_New/SSS_Clim_predict_2001_2016.nc"
directory1_SST =
"/home/biomac2/sommer/Data/Data_Predict_Taka_New/SST_Clim_predict_2001_2016.nc"
directory2 =
"/home/biomac2/sommer/Data/Data_Predict_Taka_New/CHL_Clim_predict_2001_2016.nc"
directory4 =
"/home/biomac2/sommer/Data/Data_Predict_Taka_New/MLD_Clim_predict_2001_2016.nc"
```

```
directory5 =
"/home/biomac2/sommer/Data/Data_Predict_Taka_New/CO2_atm_Clim_predict_2001_2016.nc"
directory6 = "/home/biomac2/sommer/Data/Sea_Ice_aver_New/Sea_Ice_200101_201612.nc"

salinity_Clim_Mean = readnc(directory1_SSS,'SSS')
temperature_Clim_Mean = readnc(directory1_SST,'SST')
ssh_Clim_Mean = readnc(directory1_SSH,'SSH')
CHL_Clim_Mean = readnc(directory2,'CHL')
MLD_Clim_Mean = readnc(directory4,'MLD')
pCO2_atm_Clim_Mean = readnc(directory5,'aCO2')
Sea_Ice = readnc(directory6,'mask')

#regulation all data on the same grid
CHL_Clim_Mean = CHL_Clim_Mean[:,7:,:]
Sea_Ice = Sea_Ice[:,7:,:]
pCO2_atm_Clim_Mean = pCO2_atm_Clim_Mean[:,7:,:]
MLD_Clim_Mean = MLD_Clim_Mean[:,7:,:]

#estimation of anomaly of main predictors that will be also used as predictors
# + pCO2 anomaly that we will reconstruct
salinity_SOCAT_anom = npy.zeros((len(pCO2_SOCAT),len(lat_SOCAT),len(lon_SOCAT)))
temperature_SOCAT_anom = npy.zeros((len(pCO2_SOCAT),len(lat_SOCAT),len(lon_SOCAT)))
ssh_SOCAT_anom = npy.zeros((len(pCO2_SOCAT),len(lat_SOCAT),len(lon_SOCAT)))
CHL_SOCAT_anom = npy.zeros((len(pCO2_SOCAT),len(lat_SOCAT),len(lon_SOCAT)))
MLD_SOCAT_anom = npy.zeros((len(pCO2_SOCAT),len(lat_SOCAT),len(lon_SOCAT)))
pCO2_atm_SOCAT_anom = npy.zeros((len(pCO2_SOCAT),len(lat_SOCAT),len(lon_SOCAT)))

k = 0
l = 0
for i in npy.arange(0,len(pCO2_SOCAT),1):
    salinity_SOCAT_anom[i,:,:] = salinity_SOCAT[i,:,:] - salinity_Clim_Mean[i-k*12,:,:]
    temperature_SOCAT_anom[i,:,:] = temperature_SOCAT[i,:,:] - temperature_Clim_Mean[i-k*12,:,:]
    ssh_SOCAT_anom[i,:,:] = ssh_SOCAT[i,:,:] - ssh_Clim_Mean[i-k*12,:,:]
    CHL_SOCAT_anom[i,:,:] = CHL_SOCAT[i,:,:] - CHL_Clim_Mean[i-k*12,:,:]
    MLD_SOCAT_anom[i,:,:] = MLD_SOCAT[i,:,:] - MLD_Clim_Mean[i-k*12,:,:]
    pCO2_atm_SOCAT_anom[i,:,:] = pCO2_atm_SOCAT[i,:,:] - pCO2_atm_Clim_Mean[i-k*12,:,:]
    pCO2_SOCAT[i,:,:] = pCO2_SOCAT[i,:,:] - pCO2_Taka_SOCAT[i,:,:]
    l = l + 1
    if l == 12:
       l = 0
       k = k + 1

#read data for pCO2 reconstruction (prediction)
direct1_SSS =
"/home/biomac2/sommer/Data/SST_SSH_SSS_aver_New/Averaged_SSS_200101_201612.nc"
direct1_SST =
"/home/biomac2/sommer/Data/SST_SSH_SSS_aver_New/Averaged_SST_200101_201612.nc"
direct1_SSH =
"/home/biomac2/sommer/Data/SST_SSH_SSS_aver_New/Averaged_SSH_200101_201612.nc"
```

```
direct2 = "/home/biomac2/sommer/Data/Chlorophyll_GlobColour_New/CHL_200101_201612.nc"
direct3 = "/home/biomac2/sommer/Data/MLD_aver_New/Averaged_MLD_200101_201612.nc"
direct4 = "/home/biomac2/sommer/Data/Clim_MLP_TF_2001_2016/Clim2_MLP_TF.nc"
direct5 =
"/home/biomac2/sommer/Data/CO2_atm_Inerpolated_Jena/CO2_atm_Interp_200101_201612.nc"

salinity_Clim_predict = readnc(direct1_SSS,'SSS')
temperature_Clim_predict = readnc(direct1_SST,'SST')
ssh_Clim_predict = readnc(direct1_SSH,'SSH')
CHL_Clim_predict = readnc(direct2,'CHL')
MLD_Clim_predict = readnc(direct3,'MLD')
pCO2_Taka = readnc(direct4,'pCO2')
pCO2_atm = readnc(direct5,'aCO2')
lon_SST = readnc_1d(direct1_SSS,'lon')
lat_SST = readnc_1d(direct1_SSS,'lat')

MLD_Clim_predict = MLD_Clim_predict[:,7:,:]
CHL_Clim_predict = CHL_Clim_predict[:,7:,:]
pCO2_atm = pCO2_atm[:,7:,:]

#estimation of anomaly of predictors for reconstruction
salinity_anom_predict = npy.zeros((len(salinity_Clim_predict),len(lat_SST),len(lon_SST)))
temperature_anom_predict = npy.zeros((len(salinity_Clim_predict),len(lat_SST),len(lon_SST)))
ssh_anom_predict = npy.zeros((len(salinity_Clim_predict),len(lat_SST),len(lon_SST)))
CHL_anom_predict = npy.zeros((len(salinity_Clim_predict),len(lat_SST),len(lon_SST)))
MLD_anom_predict = npy.zeros((len(salinity_Clim_predict),len(lat_SST),len(lon_SST)))
pCO2_atm_anom = npy.zeros((len(salinity_Clim_predict),len(lat_SST),len(lon_SST)))

k = 0
l = 0
for i in npy.arange(0,len(salinity_Clim_predict),1):
    salinity_anom_predict[i,:,:] = salinity_Clim_predict[i,:,:] - salinity_Clim_Mean[i-k*12,:,:]
    temperature_anom_predict[i,:,:] = temperature_Clim_predict[i,:,:] - temperature_Clim_Mean[i-
k*12,:,:]
    ssh_anom_predict[i,:,:] = ssh_Clim_predict[i,:,:] - ssh_Clim_Mean[i-k*12,:,:]
    CHL_anom_predict[i,:,:] = CHL_Clim_predict[i,:,:] - CHL_Clim_Mean[i-k*12,:,:]
    MLD_anom_predict[i,:,:] = MLD_Clim_predict[i,:,:] - MLD_Clim_Mean[i-k*12,:,:]
    pCO2_atm_anom[i,:,:] = pCO2_atm[i,:,:] - pCO2_atm_Clim_Mean[i-k*12,:,:]
    l = l + 1
    if l == 12:
      l = 0
      k = k + 1

#transfer of CHL and MLD to log
CHL_log = npy.zeros((len(CHL_SOCAT),len(lat_SOCAT),len(lon_SOCAT)))
MLD_log = npy.zeros((len(MLD_SOCAT),len(lat_SST),len(lon_SST)))

for t in npy.arange(0,len(CHL_SOCAT),1):
    for i in npy. arange(0,len(lat_SOCAT),1):
```

```
        for j in npy. arange(0,len(lon_SOCAT),1):
            CHL_log[t,i,j] = npy.log10(CHL_SOCAT[t,i,j])

for t in npy.arange(0,len(MLD_SOCAT),1):
    for i in npy. arange(0,len(lat_SST),1):
        for j in npy. arange(0,len(lon_SST),1):
            MLD_log[t,i,j] = npy.log10(MLD_SOCAT[t,i,j])

CHL_anom_log = CHL_SOCAT_anom
MLD_anom_log = MLD_SOCAT_anom

#application of ice mask
for t in npy. arange(0,len(Sea_Ice),1):
    for i in npy. arange(0,len(lat_SOCAT),1):
        for j in npy. arange(0,len(lon_SOCAT),1):
            if Sea_Ice[t,i,j] == 0:
                salinity_SOCAT[t,i,j] = npy.nan
                temperature_SOCAT[t,i,j] = npy.nan
                ssh_SOCAT[t,i,j] = npy.nan
                pCO2_Taka_SOCAT[t,i,j] = npy.nan
                CHL_log[t,i,j] = npy.nan
                MLD_log[t,i,j] = npy.nan
                pCO2_atm_SOCAT[t,i,j] = npy.nan
                pCO2_SOCAT[t,i,j] = npy.nan

#we put 0 where there is not CHL data but other predictors have their values
for t in npy. arange(0,len(CHL_log),1):
    for i in npy. arange(0,len(lat_SOCAT),1):
        for j in npy. arange(0,len(lon_SOCAT),1):
            if math.isnan(CHL_log[t,i,j]) == True and math.isnan(salinity_SOCAT[t,i,j]) == False and
math.isnan(temperature_SOCAT[t,i,j]) == False:
                CHL_log[t,i,j] = 0.
                CHL_anom_log[t,i,j] = 0.

#!!!!!normalisation of data used as predictors in training algorithm!!!!!!!!!!
salinity_region = (salinity_SOCAT - npy.nanmean(salinity_SOCAT))/npy.nanstd(salinity_SOCAT)
temperature_region = (temperature_SOCAT -
npy.nanmean(temperature_SOCAT))/npy.nanstd(temperature_SOCAT)
ssh_region =(ssh_SOCAT - npy.nanmean(ssh_SOCAT))/npy.nanstd(ssh_SOCAT)
CHL_region = (CHL_log)/npy.nanstd(CHL_log)
pCO2_region = (pCO2_SOCAT - npy.nanmean(pCO2_SOCAT))/npy.nanstd(pCO2_SOCAT)
MLD_region = (MLD_log - npy.nanmean(MLD_log))/npy.nanstd(MLD_log)
pCO2_Taka_region = (pCO2_Taka_SOCAT -
npy.nanmean(pCO2_Taka_SOCAT))/npy.nanstd(pCO2_Taka_SOCAT)
pCO2_atm_region = (pCO2_atm_SOCAT -
npy.nanmean(pCO2_atm_SOCAT))/npy.nanstd(pCO2_atm_SOCAT)

salinity_anom_region = (salinity_SOCAT_anom -
npy.nanmean(salinity_SOCAT_anom))/npy.nanstd(salinity_SOCAT_anom)
```

```
temperature_anom_region = (temperature_SOCAT_anom -
npy.nanmean(temperature_SOCAT_anom))/npy.nanstd(temperature_SOCAT_anom)
ssh_anom_region = (ssh_SOCAT_anom -
npy.nanmean(ssh_SOCAT_anom))/npy.nanstd(ssh_SOCAT_anom)
CHL_anom_region = (CHL_anom_log)/npy.nanstd(CHL_anom_log)
MLD_anom_region = (MLD_anom_log -
npy.nanmean(MLD_anom_log))/npy.nanstd(MLD_anom_log)
pCO2_atm_anom_region = (pCO2_atm_SOCAT_anom -
npy.nanmean(pCO2_atm_SOCAT_anom))/npy.nanstd(pCO2_atm_SOCAT_anom)

lat_region = lat_SOCAT
lon_region = lon_SOCAT

MLD_Clim_predict[:,0,:] = npy.nan
MLD_Clim_predict[:,len(lat_SST) - 1,:] = npy.nan

#transfer of CHL and MLD used for resonstruction to log
CHL_log_predict = npy.zeros((len(CHL_Clim_predict),len(lat_SOCAT),len(lon_SOCAT)))
MLD_log_predict = npy.zeros((len(MLD_Clim_predict),len(lat_SST),len(lon_SST)))

for t in npy.arange(0,len(CHL_Clim_predict),1):
    for i in npy. arange(0,len(lat_SOCAT),1):
        for j in npy. arange(0,len(lon_SOCAT),1):
            CHL_log_predict[t,i,j] = npy.log10(CHL_Clim_predict[t,i,j])

for t in npy.arange(0,len(MLD_Clim_predict),1):
    for i in npy. arange(0,len(lat_SST),1):
        for j in npy. arange(0,len(lon_SST),1):
            MLD_log_predict[t,i,j] = npy.log10(MLD_Clim_predict[t,i,j])

MLD_anom_predict[:,0,:] = npy.nan
MLD_anom_predict[:,len(lat_SOCAT) - 1,:] = npy.nan

CHL_anom_log_predict = CHL_anom_predict
MLD_anom_log_predict = MLD_anom_predict

#application of ice mask to data used in reconstruction
k = 0
for t in npy. arange(0,len(Sea_Ice),1):
    for i in npy. arange(0,len(lat_SOCAT),1):
        for j in npy. arange(0,len(lon_SOCAT),1):
            if Sea_Ice[t,i,j] == 0:
                salinity_Clim_predict[t,i,j] = npy.nan
                temperature_Clim_predict[t,i,j] = npy.nan
                ssh_Clim_predict[t,i,j] = npy.nan
                pCO2_Taka[t - k *12,i,j] = npy.nan
                CHL_log_predict[t,i,j] = npy.nan
                MLD_log_predict[t,i,j] = npy.nan
                pCO2_atm[t,i,j] = npy.nan
```

```
    l = l + 1
    if l == 12:
      l = 0
      k = k + 1

for t in npy. arange(0,len(CHL_log_predict),1):
    for i in npy. arange(0,len(lat_SOCAT),1):
        for j in npy. arange(0,len(lon_SOCAT),1):
            if math.isnan(CHL_log_predict[t,i,j]) == True and math.isnan(salinity_Clim_predict[t,i,j]) ==
False and math.isnan(temperature_Clim_predict[t,i,j]) == False:
                CHL_log_predict[t,i,j] = 0.
                CHL_anom_log_predict[t,i,j] = 0.

#!!!!!normalization of data used in reconstruction (prediction)!!!!!
salinity_Clim_predict = (salinity_Clim_predict -
npy.nanmean(salinity_SOCAT))/npy.nanstd(salinity_SOCAT)
temperature_Clim_predict = (temperature_Clim_predict -
npy.nanmean(temperature_SOCAT))/npy.nanstd(temperature_SOCAT)
ssh_Clim_predict = (ssh_Clim_predict - npy.nanmean(ssh_SOCAT))/npy.nanstd(ssh_SOCAT)
CHL_Clim_predict = (CHL_log_predict)/npy.nanstd(CHL_log)
MLD_Clim_predict = (MLD_log_predict - npy.nanmean(MLD_log))/npy.nanstd(MLD_log)
Taka_Clim_predict = (pCO2_Taka -
npy.nanmean(pCO2_Taka_SOCAT))/npy.nanstd(pCO2_Taka_SOCAT)
pCO2_atm = (pCO2_atm - npy.nanmean(pCO2_atm_SOCAT))/npy.nanstd(pCO2_atm_SOCAT)

salinity_anom_predict = (salinity_anom_predict -
npy.nanmean(salinity_SOCAT_anom))/npy.nanstd(salinity_SOCAT_anom)
temperature_anom_predict = (temperature_anom_predict -
npy.nanmean(temperature_SOCAT_anom))/npy.nanstd(temperature_SOCAT_anom)
ssh_anom_predict = (ssh_anom_predict -
npy.nanmean(ssh_SOCAT_anom))/npy.nanstd(ssh_SOCAT_anom)
CHL_anom_log_predict = (CHL_anom_log_predict)/npy.nanstd(CHL_anom_log)
MLD_anom_log_predict = (MLD_anom_log_predict -
npy.nanmean(MLD_anom_log))/npy.nanstd(MLD_anom_log)
pCO2_atm_anom = (pCO2_atm_anom -
npy.nanmean(pCO2_atm_SOCAT_anom))/npy.nanstd(pCO2_atm_SOCAT_anom)


#training algorithm for month April

#creation of list of tarjets and data used as predictors
#we use data from April as well as March and May to increase number of data for training and keep
some signal of sesonal variabiliy
pCO2_list = npy.zeros(100000000)
salinity_list = npy.zeros(100000000)
temperature_list = npy.zeros(100000000)
ssh_list = npy.zeros(100000000)
CHL_list = npy.zeros(100000000)
MLD_list = npy.zeros(100000000)
```

```
pCO2_Taka_list = npy.zeros(100000000)
pCO2_atm_list = npy.zeros(100000000)
lat_list = npy.zeros(100000000)
lon_list1 = npy.zeros(100000000)
lon_list2 = npy.zeros(100000000)
lat_list_degree = npy.zeros(100000000)
lon_list_degree = npy.zeros(100000000)
salinity_anom_list = npy.zeros(100000000)
temperature_anom_list = npy.zeros(100000000)
ssh_anom_list = npy.zeros(100000000)
CHL_anom_list = npy.zeros(100000000)
MLD_anom_list = npy.zeros(100000000)
pCO2_atm_anom_list = npy.zeros(100000000)
year_list = npy.zeros(100000000)
month_list = npy.zeros(100000000)

p = 0

#!!!!!!chose the first year-1 !!!!!!
year_list_const = 2000

#the loop starts from "3" that corresponds to fourth month April
for t in npy.arange(3,len(pCO2_region),12):
    year_list_const = year_list_const + 1
    if t <> 0:
      for k in npy.arange(t-1,t+2,1):
         for i in npy.arange(0,len(lat_region),1):
            for j in npy.arange(0,len(lon_region),1):
               if math.isnan(pCO2_region[k,i,j]) == False and math.isnan(salinity_region[k,i,j]) == False
and math.isnan(temperature_region[k,i,j]) == False and math.isnan(ssh_region[k,i,j]) == False and
math.isnan(CHL_region[k,i,j]) == False and math.isnan(MLD_region[k,i,j]) == False and
math.isnan(pCO2_Taka_region[k,i,j]) == False and math.isnan(pCO2_atm_region[k,i,j]) == False:
                  pCO2_list[p] = pCO2_region[k,i,j]
                  salinity_list[p] = salinity_region[k,i,j]
                  temperature_list[p] = temperature_region[k,i,j]
                  ssh_list[p] = ssh_region[k,i,j]
                  CHL_list[p] = CHL_region[k,i,j]
                  MLD_list[p] = MLD_region[k,i,j]
                  pCO2_atm_list[p] = pCO2_atm_region[k,i,j]
                  salinity_anom_list[p] = salinity_anom_region[k,i,j]
                  temperature_anom_list[p] = temperature_anom_region[k,i,j]
                  ssh_anom_list[p] = ssh_anom_region[k,i,j]
                  CHL_anom_list[p] = CHL_anom_region[k,i,j]
                  MLD_anom_list[p] = MLD_anom_region[k,i,j]
                  pCO2_Taka_list[p] = pCO2_Taka_region[k,i,j]
                  pCO2_atm_anom_list[p] = pCO2_atm_anom_region[k,i,j]
                  lat_list[p] = npy.sin(lat_region[i]*npy.pi/180.)
                  lon_list1[p] = npy.cos(lon_region[j]*npy.pi/180.)
                  lon_list2[p] = npy.sin(lon_region[j]*npy.pi/180.)
```

```
                lat_list_degree[p] = lat_region[i]
                lon_list_degree[p] = lon_region[j]
                if k == t-1:
                  month_list[p] = 3.
                  year_list[p] = year_list_const - 1
                if k == t:
                  month_list[p] = 4.
                  year_list[p] = year_list_const
                if k == t+1:
                  month_list[p] = 5.
                  year_list[p] = year_list_const
                p = p + 1

pCO2_list = pCO2_list[:p]
salinity_list = salinity_list[:p]
temperature_list = temperature_list[:p]
ssh_list = ssh_list[:p]
CHL_list = CHL_list[:p]
MLD_list = MLD_list[:p]
pCO2_atm_list = pCO2_atm_list[:p]
salinity_anom_list = salinity_anom_list[:p]
temperature_anom_list = temperature_anom_list[:p]
ssh_anom_list = ssh_anom_list[:p]
CHL_anom_list = CHL_anom_list[:p]
MLD_anom_list = MLD_anom_list[:p]
pCO2_Taka_list = pCO2_Taka_list[:p]
pCO2_atm_anom_list = pCO2_atm_anom_list[:p]
lat_list = lat_list[:p]
lon_list1 = lon_list1[:p]
lon_list2 = lon_list2[:p]
lat_list_degree = lat_list_degree[:p]
lon_list_degree = lon_list_degree[:p]
month_list = month_list[:p]
year_list = year_list[:p]

#matrix of preditors that will be used in training algorithm
data_predictors =
npy.column_stack((salinity_list,temperature_list,ssh_list,CHL_list,MLD_list,pCO2_atm_list,lat_list,lo
n_list1,lon_list2,salinity_anom_list,temperature_anom_list,ssh_anom_list,CHL_anom_list,MLD_anom
_list,pCO2_atm_anom_list))

lat_region1 = lat_SOCAT
lon_region1 = lon_SOCAT

#creation of list of data used for reconstruction
data_reconstr = npy.zeros((int(len(salinity_Clim_predict)/12.),1000000,15))
reconstr_numb = npy.zeros(int(len(salinity_Clim_predict)/12.))

p = 0
```

```
tt = 0
for t in npy.arange(3,len(salinity_Clim_predict),12):
    p = 0
    salinity_Clim_predict_list = npy.zeros(1000000)
    temperature_Clim_predict_list = npy.zeros(1000000)
    ssh_Clim_predict_list = npy.zeros(1000000)
    CHL_Clim_predict_list = npy.zeros(1000000)
    MLD_Clim_predict_list = npy.zeros(1000000)
    pCO2_atm_Clim_predict_list = npy.zeros(1000000)
    salinity_anom_predict_list = npy.zeros(1000000)
    temperature_anom_predict_list = npy.zeros(1000000)
    ssh_anom_predict_list = npy.zeros(1000000)
    CHL_anom_predict_list = npy.zeros(1000000)
    MLD_anom_predict_list = npy.zeros(1000000)
    pCO2_atm_anom_predict_list = npy.zeros(1000000)
    Taka_Clim_predict_list = npy.zeros(1000000)
    lat_predict_list = npy.zeros(1000000)
    lon_predict_list1 = npy.zeros(1000000)
    lon_predict_list2 = npy.zeros(1000000)
    month_predict_list = npy.zeros(1000000)
    for i in npy.arange(0,len(lat_region),1):
        for j in npy.arange(0,len(lon_region),1):
            if math.isnan(salinity_Clim_predict[t,i,j]) == False and
math.isnan(temperature_Clim_predict[t,i,j]) == False and math.isnan(ssh_Clim_predict[t,i,j]) == False
and math.isnan(CHL_Clim_predict[t,i,j]) == False and math.isnan(MLD_Clim_predict[t,i,j]) == False
and math.isnan(pCO2_atm[t,i,j]) == False:
                salinity_Clim_predict_list[p] = salinity_Clim_predict[t,i,j]
                temperature_Clim_predict_list[p] = temperature_Clim_predict[t,i,j]
                ssh_Clim_predict_list[p] = ssh_Clim_predict[t,i,j]
                CHL_Clim_predict_list[p] = CHL_Clim_predict[t,i,j]
                MLD_Clim_predict_list[p] = MLD_Clim_predict[t,i,j]
                pCO2_atm_Clim_predict_list[p] = pCO2_atm[t,i,j]
                salinity_anom_predict_list[p] = salinity_anom_predict[t,i,j]
                temperature_anom_predict_list[p] = temperature_anom_predict[t,i,j]
                ssh_anom_predict_list[p] = ssh_anom_predict[t,i,j]
                CHL_anom_predict_list[p] = CHL_anom_log_predict[t,i,j]
                MLD_anom_predict_list[p] = MLD_anom_log_predict[t,i,j]
                pCO2_atm_anom_predict_list[p] = pCO2_atm_anom[t,i,j]
                Taka_Clim_predict_list[p] = Taka_Clim_predict[3,i,j]
                lat_predict_list[p] = npy.sin(lat_region1[i]*npy.pi/180.)
                lon_predict_list1[p] = npy.cos(lon_region1[j]*npy.pi/180.)
                lon_predict_list2[p] = npy.sin(lon_region1[j]*npy.pi/180.)
                month_predict_list[p] = 4.
                p = p + 1
    reconstr_numb[tt] = p
    matrix_reconst =
npy.column_stack((salinity_Clim_predict_list,temperature_Clim_predict_list,ssh_Clim_predict_list,CH
L_Clim_predict_list,MLD_Clim_predict_list,pCO2_atm_Clim_predict_list,lat_predict_list,lon_predict
_list1,lon_predict_list2,salinity_anom_predict_list,temperature_anom_predict_list,ssh_anom_predict_li
```

```
st,CHL_anom_predict_list,MLD_anom_predict_list,pCO2_atm_anom_predict_list))
    #creation of matrix for predictors used in reconstruction. For each time step (tt) there is an ensemble
of all predictors to reconstruct pCO2 for April of considered year (tt)
    data_reconstr[tt,:,:] = matrix_reconst
    tt = tt + 1

    salinity_Clim_predict_list = salinity_Clim_predict_list[:p]
    temperature_Clim_predict_list = temperature_Clim_predict_list[:p]
    ssh_Clim_predict_list = ssh_Clim_predict_list[:p]
    CHL_Clim_predict_list = CHL_Clim_predict_list[:p]
    MLD_Clim_predict_list = MLD_Clim_predict_list[:p]
    pCO2_atm_Clim_predict_list = pCO2_atm_Clim_predict_list[:p]
    salinity_anom_predict_list = salinity_anom_predict_list[:p]
    temperature_anom_predict_list = temperature_anom_predict_list[:p]
    ssh_anom_predict_list = ssh_anom_predict_list[:p]
    CHL_anom_predict_list = CHL_anom_predict_list[:p]
    MLD_anom_predict_list = MLD_anom_predict_list[:p]
    Taka_Clim_predict_list = Taka_Clim_predict_list[:p]
    pCO2_atm_anom_predict_list = pCO2_atm_anom_predict_list[:p]
    lat_predict_list = lat_predict_list[:p]
    lon_predict_list1 = lon_predict_list1[:p]
    lon_predict_list2 = lon_predict_list2[:p]
    month_predict_list = month_predict_list[:p]

#k-fold cross-validation FFNN. 50% of data for training, 25% of data for validation and 25% for
evaluation. => 4 samplings/ 4 tests
for numb_of_model in npy.arange(0,4,1):
    index1 = npy.zeros(len(pCO2_list), dtype=npy.int)
    p = 0
    for i in npy.arange(numb_of_model,len(pCO2_list),4):
        index1[p] = i
        p = p + 1
    index1 = index1[:p]
    data_train1 = npy.delete(data_predictors, index1, axis = 0)
    pCO2_list_train1 = npy.delete(pCO2_list, index1)
    lon_list_degree1 = npy.delete(lon_list_degree, index1)
    lat_list_degree1 = npy.delete(lat_list_degree, index1)
    year_list1 = npy.delete(year_list, index1)
    month_list1 = npy.delete(month_list, index1)
    data_eval = npy.zeros((len(index1),15)) #data for evaluation (do not participate in training)
    pCO2_list_eval = npy.zeros(len(index1)) #data for evaluation (do not participate in training)
    for i in npy.arange(0,len(index1),1):
        data_eval[i,:] = data_predictors[index1[i],:]
        pCO2_list_eval[i] = pCO2_list[index1[i]]
    index2 = npy.zeros(len(pCO2_list_train1), dtype=npy.int)
    p = 0
    for i in npy.arange(numb_of_model,len(pCO2_list_train1),3):
        index2[p] = i
        p = p + 1
```

```
    index2 = index2[:p]
    data_train = npy.delete(data_train1, index2, axis = 0)
    pCO2_list_train = npy.delete(pCO2_list_train1, index2)
    lon_train = npy.delete(lon_list_degree1, index2) #data for training
    lat_train = npy.delete(lat_list_degree1, index2) #data for training

    data_val = npy.zeros((len(index2),15))
    pCO2_list_val = npy.zeros(len(index2))
    pCO2_list_val_tot = npy.zeros(len(index2))
    lon_degree_val = npy.zeros(len(index2))
    lat_degree_val = npy.zeros(len(index2))
    year_list_val = npy.zeros(len(index2))
    month_list_val = npy.zeros(len(index2))
    for i in npy.arange(0,len(index2),1):
        data_val[i,:] = data_train1[index2[i],:] #data for validation (add to validate a model during the
iterative procedure of training)
        pCO2_list_val[i] = pCO2_list_train1[index2[i]] #data for validation (add to validate a model
during the iterative procedure of training)
        lon_degree_val[i] = lon_list_degree1[index2[i]]
        lat_degree_val[i] = lat_list_degree1[index2[i]]
        year_list_val[i] = year_list1[index2[i]]
        month_list_val[i] = month_list1[index2[i]]

    print 'Version', numb_of_model
    print 'Training data!!!!!!!!!!!!!!!!!!!!!!!!!!!!', data_train.shape, pCO2_list_train.shape
    print 'Validation data!!!!!!!!!!!!!!!!!!!!!!!!!!', data_val.shape, pCO2_list_val.shape
    print 'Evaluation data!!!!!!!!!!!!!!!!!!!!!!!!!!', data_eval.shape, pCO2_list_eval.shape


    #!!!!!!!!!CREATION OF MODEL!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    earlyStopping=keras.callbacks.EarlyStopping(monitor='val_loss', patience=30, verbose=0,
mode='auto')
    model = Sequential()
    model.add(Dense(30, input_dim=15, init='glorot_uniform'))
    model.add(Activation('tanh'))
    model.add(Dense(25, init='glorot_uniform'))
    model.add(Activation('tanh'))
    model.add(Dense(20, init='glorot_uniform'))
    model.add(Activation('tanh'))
    model.add(Dense(1, init='glorot_uniform'))
    model.add(Activation('linear'))
    sgd = SGD(lr=0.01, decay=0.0, momentum=0.0, nesterov=False)
    rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
    L = model.compile(loss='mse',optimizer=rmsprop)
    print model.get_weights()
    H = model.fit(data_train,
pCO2_list_train,nb_epoch=1600,callbacks=[earlyStopping],batch_size=20,validation_data=(data_val,p
CO2_list_val))
    score = model.evaluate(data_eval, pCO2_list_eval, batch_size=20)
```

```
    print model.get_weights()
    print model.summary()
    print model.get_config()

    #write basic loss function and loss function of validation data (form earlystopping)
    fo =
open("/home/users/asommer/Cost_Function_FFN_2001_2016/New_Cost_Function_Corr_Interannual_
season_tot_Apr_test5_1_" + str(numb_of_model) + ".txt","wb")
    for i in npy.arange(0,len(H.history['loss']),1):
        fo.write(str(i)+';'+str(npy.log10(H.history['loss'][i]))+';'+str(npy.log10(H.history['val_loss'][i]))+
"\n")
    fo.close()

    #reconstruction based on training data tot test model capacity
    preds_test = model.predict(data_train)
    preds_test1 = preds_test[:,0]

    print 'TEST VALIDATION!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
    print 'Personr',scipy.stats.pearsonr(pCO2_list_train,preds_test1)
    print 'RMS', npy.sqrt(npy.nanmean((pCO2_list_train-preds_test1)**2))
    slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(pCO2_list_train,preds_test1)
    print 'R2', r_value**2
    print 'Bias', npy.nanmean(pCO2_list_train) - npy.nanmean(preds_test1)
    print 'TEST VALIDATION FIN!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'

    std_SOCAT = npy.nanstd(pCO2_SOCAT)
    mean_SOCAT = npy.nanmean(pCO2_SOCAT)
    std_Taka = npy.nanstd(pCO2_Taka)
    mean_Taka = npy.nanmean(pCO2_Taka)

    #!!!!!!!!!!!!!!!!!!!!!RECONSTRUCTION!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    pCO2_SOCAT_list1 = npy.zeros(100000)
    pCO2_reconstr_list1 = npy.zeros(100000)

    year_list_record =
['2001','2002','2003','2004','2005','2006','2007','2008','2009','2010','2011','2012','2013','2014','2015','201
6']

    lat = lat_SOCAT
    lon = lon_SOCAT

    j_SOCAT = 0
    j_Val = 0

    pCO2_matrix_time = npy.zeros((len(year_list_record),len(lat),len(lon)))

    for r in npy.arange(0,len(year_list_record),1):
        pCO2_matrix = npy.zeros((len(lat),len(lon)))
        preds1 = model.predict(data_reconstr[r,:int(reconstr_numb[r]),:])
```

```
        preds_list = preds1[:,0]
        p = 0
        for i in npy.arange(0,len(lat),1):
            for j in npy.arange(0,len(lon),1):
                if math.isnan(salinity_Clim_predict[r*12+3,i,j]) == False and
math.isnan(temperature_Clim_predict[r*12+3,i,j]) == False and
math.isnan(ssh_Clim_predict[r*12+3,i,j]) == False and math.isnan(CHL_Clim_predict[r*12+3,i,j]) ==
False and math.isnan(MLD_Clim_predict[r*12+3,i,j]) == False and math.isnan(pCO2_atm[r*12+3,i,j])
== False:
                   pCO2_matrix[i,j] = preds_list[p] * std_SOCAT + mean_SOCAT + pCO2_Taka[3,i,j]
                   p = p + 1
                else:
                   pCO2_matrix[i,j] = npy.nan


        #write reconstructed pCO2 in NetCDF
        #"test1" - first run of the model; "number of model" - K-fold sampling: 0,1,2,3
        time = 0

write_2d_reg_file('/home/biomac2/sommer/Data/pCO2_FFN_2step_20012016/pCO2_TF_Corr_Apr_t
est1_' + str(numb_of_model) + '_' + year_list_record[r] + '.nc',lon,lat,time,pCO2_matrix,'pCO2')

        #reconstructed pCO2 at the postion of validation data
        for i in npy.arange(0,len(lat),1):
            for j in npy.arange(0,len(lon),1):
                for kk in npy.arange(0,len(lat_degree_eval),1):
                    if month_list_eval[kk] == 4 and year_list_eval[kk] == int(year_list_record[r]):
                       if lat_degree_eval[kk] == lat[i] and lon_degree_eval[kk] == lon[j]:
                          if math.isnan(pCO2_matrix[i,j]) == False:
                             pCO2_train_val_extr[j_Val] = pCO2_matrix[i,j]
                             pCO2_percip_val_extr[j_Val] = pCO2_list_eval[kk] * std_SOCAT + mean_SOCAT
+ pCO2_Taka[3,i,j]
                             j_Val = j_Val + 1

    print 'Number validation', j_Val
    pCO2_train_val_extr = pCO2_train_val_extr[:j_Val]
    pCO2_percip_val_extr = pCO2_percip_val_extr[:j_Val]

    pCO2_list_val = pCO2_train_val_extr
    pCO2_test2 = pCO2_percip_val_extr

    #ESTIMATION OF STATISTICAL RESULTS OF MODEL (MODEL
ACCURACY)!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    print 'TOTAL'
    print '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
    print'Validation'
    print 'Personr',scipy.stats.pearsonr(pCO2_list_val,pCO2_test2)
    print 'RMS', npy.sqrt(npy.nanmean((pCO2_list_val-pCO2_test2)**2))
    slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(pCO2_list_val,pCO2_test2)
```

```
print 'R2', r_value**2
print 'Bias', npy.nanmean(pCO2_list_val) - npy.nanmean(pCO2_test2)
```