AMATH 563 Multiple Resolution Analysis

Tyler Chen

ABSTRACT

We introduce Multiple Resolution Analysis and the Discrete Wavelet Transform. We then discuss how these might be applied to a data set of global ocean temperatures over the past few decades in order to isolate El Niño. We find that the Oceanic Niño Index can be recovered using just a single spatial point. Finally, we train a neural net to predict the next time the ONI will reach a certain value.

I INTRODUCTION AND OVERVIEW

Multiple Resolution Analysis (MRA) provides a way to view data on multiple scales in both the position and frequency domains simultaneously.

The El Niño Southern Oscillator (ENSO) is an "irregularly periodic varaition in winds and sea surface temperatures over the tropical easter Pacific Ocean". The warm phase is referred to as El Niño and the cold phase is referred to as La Niña. When the ENSO is at its extremes, there is often extreme weather throughout part of the world.

We apply MRA to a ocean surface temperature data set in the hopes of extracting information useful in predicting El Niño.

II THEORETICAL BACKGROUND

Multiple Resolution Analysis (MRA)

Broadly, Multiple Resolution Analysis provides a way to obtain information at multiple scales (as the name suggests). We illustrate this with an example.

Suppose we have an audio recording of a piano. The recording tells us the amplitude of the sounds waves at a given time. What if we want to know which notes were played? We could take the Fourier transform of our recording and get information about the frequencies present. However, unless the notes were played through the entire recording, this would not be very useful.

Multiple resolution analysis lets us get information about what notes were played when. Low frequencies, which have periods comparable to the entire recording can be extracted, but without timing information, while high frequencies can be pinpointed in time. This example illustrates the basic principle of MRA.

Discrete Wavelet Transform

One way to get time frequency information is to split the data into many pieces and take the Fourier transform of each piece. This would tell us information about which frequencies are present, as well as when these frequencies are present. Such a decomposition is often referred to as a spectogram or Short Time Discrete Fourier Transform (STDFT).

In the discrete case it is easy to see that such a transform is just a change of basis. Figure 1 shows a possible basis for $\mathbb{R}^8$. This basis gives us information about 4 different frequencies, as well as information about where these frequencies occur in time.

From this figure it is obvious that these functions are orthogonal, and could be appropriately normalized. It is also clear that such a basis can be generalized to any 2^k-dimensional vectorspace over $\mathbb{R}$. This basis has some advantages over a STDFT, namely that it is able to pick up low frequency modes which have a period longer than the windows in a STDFT.

This approach can be generalized to to a a class of transforms commonly referred to as the Discrete Wavelet Transform. While we show only a class of "wavelets" calleed "Haar Wavelets", many other shapes can be used. Note that the same "wavelet" seen in the first non-constant mode is just shifted and scaled to produce an orthonormal basis. Therefore it should not be surprising that there are many possible wavelets which are used in practice.

Like the Discrete Fourier Transform, there are efficient algorithms for computing the DWT for many wavelets.

Such a transform can be applied to vector-spaces where

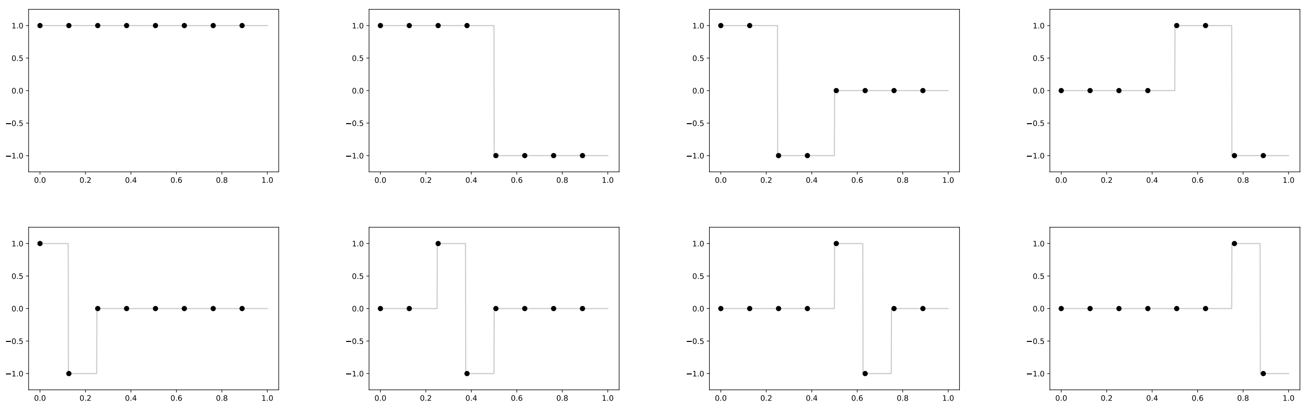


Figure 1: Haar basis with $N = 8$

the vectors are multi-dimensional arrays (for instance a photo). More specifically, suppose our data lives in $\mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$.

Let $\mathcal{B}_i = \{b_{i1}, b_{i2}, \ldots, b_{ik_i}\}$ be a wavelet basis for $\mathbb{R}^{k_i}$. Then,

$$\mathcal{B} = \mathcal{B}_1 \otimes \mathcal{B}_2 \otimes \cdots \otimes \mathcal{B}_d \tag{1}$$

is a basis for $\mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$, where $\otimes$ denotes the tensor product. In the 2-dimensional case this is simply the collection of all possible outer products of the elements of the bases for each dimension.

Such a change of basis would give information about the position and frequency/wave number of the data in each dimension. For instance, as discussed later, information about the location of a ocean temperate event in both space (longitude and latitude) and time along with information about frequency (in both spatial dimensions and in time) can be gathered.

Such a change of basis is not practical to compute for an arbitrary wavelet basis using naive algorithms since the dimension of the vector-space would generally be quite large. In particular, computing the standard inner product of two vectors of size N requires $\mathcal{O}(2N)$ operations. Even if these vectors can be accessed in constant time, a change of basis would require N inner products to project the data onto each of the N new basis vectors. Therefore it will require something like $\mathcal{O}(N^2)$ operations to compute the change of basis. With vectors with millions of components this is not practical.

There are almost certainly better algorithms for certain bases to help deal with this problem.

SVD

We can also perform MRA using the SVD.

Given any vector of the size of a single snapshot of our data we can project our data onto this vector and see how this "mode" varies in time. That means, if we are able to isolate a vector which represents El Niño we would be able to see hot his varies in time.

Filtering

Filtering is a common technique in signal processing where the frequency content of a signal is altered. A low pass filter maintains the low frequencies of a signal and attenuates the high frequencies of a signal. This can be useful for de-noising a signal or removing a high frequency periodic piece of a signal.

Oceanic Niño Index

The Oceanic Niño Index (ONI) is a 3 month running mean of ERSST.v5 SST anomalies in the Niño 3.4 region (5°N-5°S, 120°-170°W) [1]. This gives a measure of El Niño events.

III ALGORITHM IMPLEMENTATION AND DEVELOPMENT

We create some basic functions to help with out analysis of our data set.[1]

[1] I would have preferred to just put this section in with the computational results section, but I wasn't sure how strict the formatting rules Nathan sent out are.

Discrete Wavelet Transform

We implement a function to generate a wavelet basis of arbitrary size given a mother wavelet. Using these we can construct a basis for our data. However, since the dataset we are working with has over 94 million points, it is not practical at all to use our naive implementation to compute the coefficients in the wavelet basis.

Our function assumes that there are $N = 2^k$ points in the data set for $k \in \mathbb{N}$. A wavelet with period equal to the number of points is then generated. The points are then split, and two wavelets, each with period equal to $N/2$ but starting positions at 0 and $N/2$. This process is repeated. We note that we do define our function recursively, but rather using an appropriately indexed loop.

This function could be used to generate bases for a multi dimensional DWT, however then all of the basis functions would have to be stored.

Singular Value Decomposition

We write a function to take the SVD of a slice of our data. In particular, the array is reshaped so that the spatia axes lie on a single axis prior to taking the SVD.

Filtering

We use Scipy's signal module for filtering.

Prediction

Being able to predict El Niño events would be of great use as it would allow farmers and others whose livelihoods depend on global climate events to prepare for upcoming changes. In past assignments we have explored methods of predicting time-series data including sparse regression and neural nets.

IV COMPUTATIONAL RESULTS

We are given ocean surface temperature data of size $(M, N, T) = (180, 360, 1455)$. Each time point is taken one week apart starting on December 31, 1989 and ending on November 12, 2017.

Discrete Wavelet Transform

We attempt to project a smaller portion of our data. This was inefficient and we did not pursue this approach. Our code is usable in the sense that it will

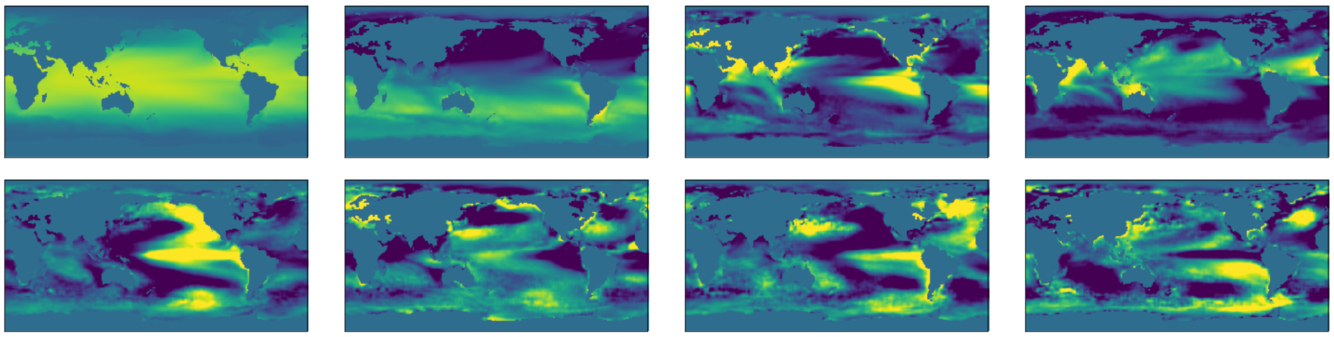


Figure 2: First 8 modes of SVD of full data (all with same color scale)

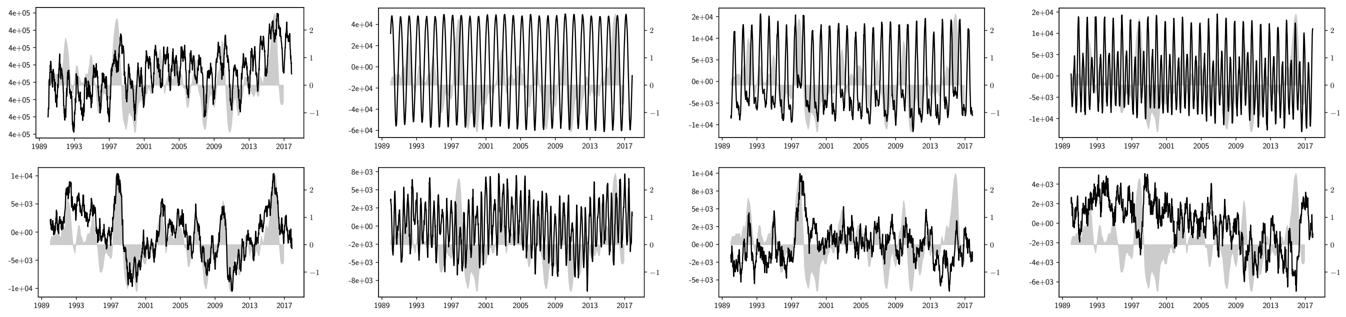


Figure 3: Weights of first 8 modes of SVD against time

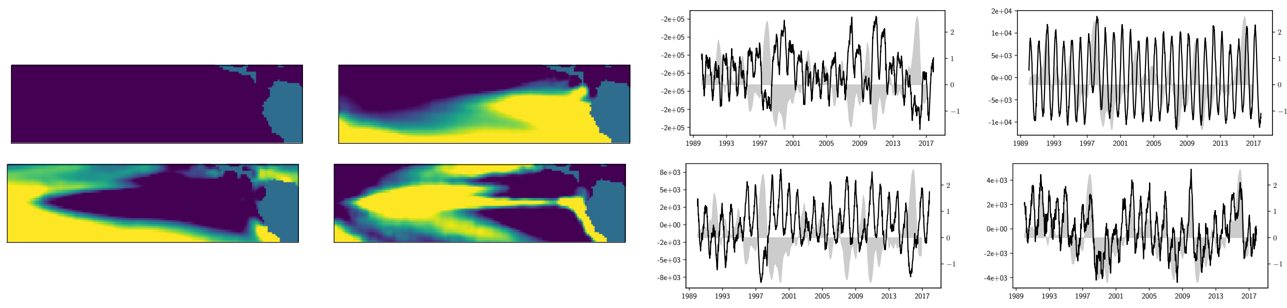


Figure 4: First 4 modes of SVD of data localized in space (all with same color scale) and corresponding weights in time

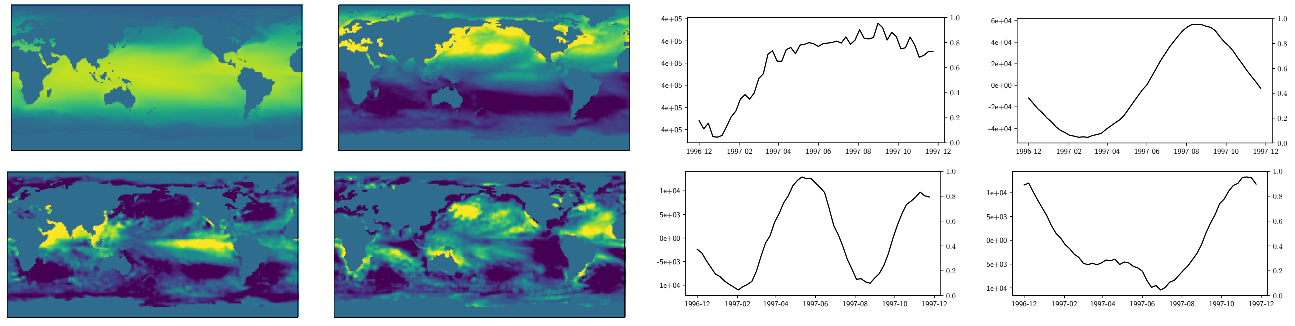


Figure 5: First 4 modes of SVD of data localized in time (all with same color scale) and corresponding weights in time

provide the desired information, however it is still a quadratic time algorithm which seemed to be running too slowly to justify spending more time on this approach.

Singular Value Decomposition

The SVD of the entire data set is taken. The 8 most dominant modes are shown in Figure 2. The corresponding weights of these modes in time are shown in Figure 3. It is clear that the 5th mode more or less tracks the ONI.

We now perform some rudimentary MRA by observing a spatial slice of the data over the full time range as well as a temporal slice of the data over the whole earth. Neither give particularly interesting results. More specifically, in space we isolated a region of the Pacific Ocean off the coast of South America near the Niño 3.4 region and in time we isolated the year 1997 which was a particularly strong El Niño event.

Figures 4 shows the modes and how they vary in time when we localize in space. We see that the coefficients of many of these modes track the ONI well. Figures 5 shows the modes and how they vary in time when we localize in time. We see that the more important modes of the SVD look more like what we expect the El Niño to look like.

While these results were initially encouraging, as discussed later, it turns out that even a single point from this region is enough to track the ONI and so this is not so interesting.

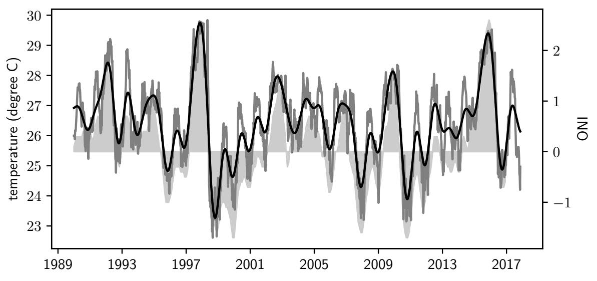
Direct Filtering

The region of the 7th mode of the SVD in the Pacific ocean off the coast of South America looks like what we might expect the El Niño to look like. We isolate the desired portion of this mode and zero the rest of the data. A small Gaussian blur is then applied to clear the harsh boundaries of the clipping.

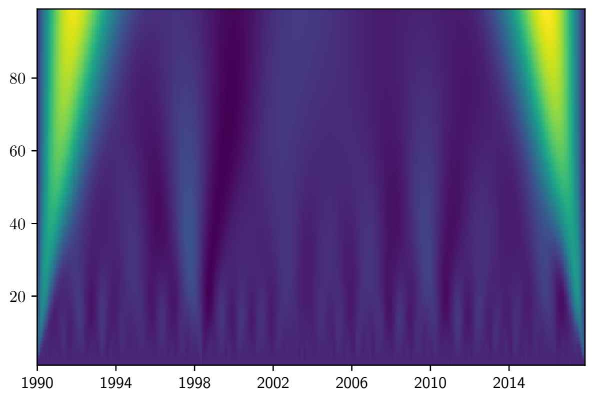
The dataset is then projected onto a normalized version of this mode. We look at the coefficients of this mode in time and find that we are more or less able to recover the The Oceanic Niño Index (ONI) (with a low pass filter).

With this as a motivation we try an even simpler vector taking just a single point $(90, 245)$ corresponding to the position 0°S, 145°W in the center of the Niño 3.4

region. Again a low pass filter is applied and the ONI is recovered. This is shown in Figure 6.



(a) temperature at (90,245) (dark grey), filtered temperature (black) ONI data (light grey fill)



(b) CWT of temperature at (90,245) (ricker wavelet)

Figure 6: Analysis of temperature at (90,245)

Prediction

We train a neural net to predict the next time the ONI will be above a fixed threshold. To build the target data we first detect when the ONI reaches the specified threshold. These points are marked, and then the time to reach the next one of these points is found for each time in the given data set. We then train a neural network with the input being the current global ocean temperatures and the output being the next time until the ONI reaches the specified threshold using 5% of the data for cross validation.

As before, since we do not have much time to train a network, and since we do not understand how to pick the proper network architecture, our results are not notable. That said everything we did was designed to be easily scaleable so that it could be adjusted in the future. This seems like an example of where a convolutional neural net might be useful.

V SUMMARY AND CONCLUSIONS

The Discrete Wavelet Transform seems to provide a clean way of detecting and analyzing multi-scale phenomena without and sort of supervision. Given more time we would like to write an efficient algorithm to compute this transform. At the very least rather than projecting using inner products, slicing could be use for some wavelets. This would be more efficient in terms of floating point operations, and given fast memory access might prove feasible on our computing resources. Alternatively, and perhaps more practically , the algorithm could be implemented for parallel computation. Projecting onto a basis amounts to computing many independent inner products, which is the perfect type of problem for a GPU or large cluster. Parallelization would allow the fast computation of the projection onto any basis.

The Oceanic Niño Index seems to be almost completely recoverable from a single point. This suggests that the definition of the index could be tightened. However, perhaps taking recordings over a larger area gives more precision.

Finally, we train a neural net to predict the next time the ONI will reach a certain threshold. Due to the limited time this was not very successful. However, our code is easily scaleable if we were to have more time to invest in the project.

REFERENCES

[1] Cold warm episodes by season. Accessed: May 18, 2017.

VI APPENDIX A

```
#wavelet defined on [0,1]
def Haar_wavelet(x,N):
    return 1 if x<1/2 else -1
Haar_wavelet = np.vectorize(Haar_wavelet)

def sin_wavelet(x,N):
    return 2* np.sin(2*np.pi*x*(N-1)/N)
sin_wavelet = np.vectorize(sin_wavelet)


# generate discrete wavelet basis based on mother wave w over 2^k points
# |w|^2 should be normalized on [0,1]
def generate_wavelet_basis(w,k):
    N = 2**k

    # rows of U are basis
    U = np.zeros((N,N))
    U[0] = np.ones(N) / np.sqrt(N)

    for i in range(k):
        for j in range(2**i):
            index = 2**i+j
            length = 2**(k-i)
            U[index,j*length:(j+1)*length] = w(np.linspace(0,1,length),length) / np.sqrt(
                length)

    return U
```

```
def svd_of_slice(s):
    sst_cut = sst[s]
    M,N,T = np.shape(sst_cut)

    sst_reshaped = np.reshape(sst_cut,(-1,T))
    [u,s,vh] = np.linalg.svd(sst_reshaped, full_matrices=False)

    U = np.reshape(u, (M,N,-1))

    return U,s,vh
```

VII APPENDIX B

```
k1 = 5
k2 = 5
k3 = 5

trimmed_sst = sst[0:2**k1,0:2**k2,0:2**k3]

U1 = generate_wavelet_basis(Haar_wavelet,k1)
U2 = generate_wavelet_basis(Haar_wavelet,k2)
U3 = generate_wavelet_basis(Haar_wavelet,k3)

N1 = 2**k1
N2 = 2**k2
N3 = 2**k3
```

```
coeffs = np.zeros(N1*N2*N3)

for i in range(N1):
    for j in range(N2):
        for k in range(N3):
            index = i*N2*N3 + j*N3 + k

            if index % 1000 == 0 : print(index)

            # construct 3d wavelet basis
            spatial_basis = np.transpose([U2[j]])@[U1[i]]
            full_basis = np.kron(np.transpose([[U3[k]]]),spatial_basis)

            #normalize
            full_basis /= np.linalg.norm(full_basis)

            full_basis = np.transpose(full_basis, axes=[2,1,0])

            coeffs[index] = np.sum(trimmed_sst*full_basis)
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import scipy as sp
import numpy as np
from scipy.io import netcdf
from scipy import signal
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.ticker import FormatStrFormatter
from matplotlib.animation import FuncAnimation
import datetime

# import netCDF files and set up as numpy arrays
sst_nc = netcdf.netcdf_file('sst.wkmean.1990-present.nc', 'r')
mask_nc = netcdf.netcdf_file('lsmask.nc', 'r')

time = sst_nc.variables['time'][:].copy()
sst = np.transpose(sst_nc.variables['sst'][:].copy(),axes=[1,2,0])
mask = mask_nc.variables['mask'][:].copy()[0,:,:]

sst_nc.close()
mask_nc.close()

N = 360*180;
L = 1400;

M,N,T = np.shape(sst)
max_temp = np.max(sst)
min_temp = np.min(sst)

# apply mask to all entries
sst = sst*np.transpose([mask]*T,axes=[1,2,0])

%matplotlib inline

#%% convert times since 1800 to dates
def time_to_date(t):
    origin_time = datetime.date(1800,1,1)
    delta = datetime.timedelta(days=t)
    return origin_time + delta

dates = list(map(time_to_date,time))

#%% take SVD of 3d array
#<start:svd_slice>
def svd_of_slice(s):
    sst_cut = sst[s]
    M,N,T = np.shape(sst_cut)

    sst_reshaped = np.reshape(sst_cut,(-1,T))
    [u,s,vh] = np.linalg.svd(sst_reshaped, full_matrices=False)

    U = np.reshape(u, (M,N,-1))

    return U,s,vh
#<end:svd_slice>

#%% load ONI data from NOAA

ONI = np.array([0.1, 0.2, 0.3, 0.3, 0.3, 0.3, 0.3, 0.4, 0.4, 0.3, 0.4, 0.4, #1990
0.4, 0.3, 0.2, 0.3, 0.5, 0.6, 0.7, 0.6, 0.6, 0.8, 1.2, 1.5,
1.7, 1.6, 1.5, 1.3, 1.1, 0.7, 0.4, 0.1, -0.1, -0.2, -0.3, -0.1,
0.1, 0.3, 0.5, 0.7, 0.7, 0.6, 0.3, 0.3, 0.2, 0.1, 0.0, 0.1,
0.1, 0.1, 0.2, 0.3, 0.4, 0.4, 0.4, 0.4, 0.6, 0.7, 1.0, 1.1,
1.0, 0.7, 0.5, 0.3, 0.1, 0.0, -0.2, -0.5, -0.8, -1.0, -1.0, -1.0,
-0.9, -0.8, -0.6, -0.4, -0.3, -0.3, -0.3, -0.3, -0.4, -0.4, -0.4, -0.5,
-0.5, -0.4, -0.1, 0.3, 0.8, 1.2, 1.6, 1.9, 2.1, 2.3, 2.4, 2.4,
2.2, 1.9, 1.4, 1.0, 0.5, -0.1, -0.8, -1.1, -1.3, -1.4, -1.5, -1.6,
-1.5, -1.3, -1.1, -1.0, -1.0, -1.0, -1.1, -1.1, -1.2, -1.3, -1.5, -1.7,
-1.7, -1.4, -1.1, -0.8, -0.7, -0.6, -0.6, -0.5, -0.5, -0.6, -0.7, -0.7,
-0.7, -0.5, -0.4, -0.3, -0.3, -0.1, -0.1, -0.1, -0.2, -0.3, -0.3, -0.3,
-0.1, 0.0, 0.1, 0.2, 0.4, 0.7, 0.8, 0.9, 1.0, 1.2, 1.3, 1.1,
0.9, 0.6, 0.4, 0.0, -0.3, -0.2, 0.1, 0.2, 0.3, 0.3, 0.4, 0.4,
0.4, 0.3, 0.2, 0.2, 0.2, 0.3, 0.5, 0.6, 0.7, 0.7, 0.7, 0.7,
0.6, 0.6, 0.4, 0.4, 0.3, 0.1, -0.1, -0.1, -0.1, -0.3, -0.6, -0.8,
-0.8, -0.7, -0.5, -0.3, 0.0, 0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 0.9,
0.7, 0.3, 0.0, -0.2, -0.3, -0.4, -0.5, -0.8, -1.1, -1.4, -1.5, -1.6,
-1.6, -1.4, -1.2, -0.9, -0.8, -0.5, -0.4, -0.3, -0.3, -0.4, -0.6, -0.7,
-0.8, -0.7, -0.5, -0.2, 0.1, 0.4, 0.5, 0.5, 0.7, 1.0, 1.3, 1.6,
1.5, 1.3, 0.9, 0.4, -0.1, -0.6, -1.0, -1.4, -1.6, -1.7, -1.7, -1.6,
-1.4, -1.1, -0.8, -0.6, -0.5, -0.4, -0.5, -0.7, -0.9, -1.1, -1.1, -1.0,
-0.8, -0.6, -0.5, -0.4, -0.2, 0.1, 0.3, 0.3, 0.3, 0.2, 0.0, -0.2,
-0.4, -0.3, -0.2, -0.2, -0.3, -0.3, -0.4, -0.4, -0.3, -0.2, -0.2, -0.3,
-0.4, -0.4, -0.2, 0.1, 0.3, 0.2, 0.1, 0.0, 0.2, 0.4, 0.6, 0.7,
0.6, 0.6, 0.6, 0.8, 1.0, 1.2, 1.5, 1.8, 2.1, 2.4, 2.5, 2.6,
2.5, 2.2, 1.7, 1.0, 0.5, 0.0, -0.3, -0.6, -0.7, -0.7, -0.7, -0.6])

ONI_dates = []
for y in range(1990,2017):
    for m in range(1,13):
        ONI_dates.append(datetime.date(y,m,1))

ONI_interp_f = sp.interpolate.interp1d(np.linspace(0,1,len(ONI)),ONI)
ONI_interp = ONI_interp_f(np.linspace(0,1,len(dates)))

#%% Take SVD of full data
```

```
U,s,vh = svd_of_slice(np.s_[:,:,:])


#%% Plot singular values
mpl.rcParams['text.usetex'] = True
plt.figure()
plt.scatter(np.arange(16),s[:16],color='k')
plt.yscale('log')
plt.savefig('img/svd/'+str(T)+'/s.pdf',bbox_inches='tight')

#%% Plot SVD modes

for j in range(8):
    plt.figure(figsize=(6,3))
    ax1 = plt.gca()
    ax2 = ax1.twinx()

    # plot ONI data
    ax2.fill_between(ONI_dates,0,ONI,color='.8')

    # plot filtered data
    ax1.set_zorder(ax2.get_zorder()+1)
    ax1.patch.set_visible(False)
    ax1.plot(dates,s[j]*vh[j],color='k')
    ax1.yaxis.set_major_formatter(FormatStrFormatter('%1.0e'))

    plt.savefig('img/svd/'+str(T)+'/vh_'+str(j)+'.pdf',bbox_inches='tight')


    plt.figure(figsize=(6,3))

    ax1 = plt.gca()

    ax1.invert_yaxis()
    ax1.axes.get_xaxis().set_ticks([])
    ax1.axes.get_yaxis().set_ticks([])

    m = plt.imshow(U[:,:,j] , vmin=-5e-3,vmax=9e-3)
    m.set_rasterized(True)
#    plt.axis('image')

    plt.savefig('img/svd/'+str(T)+'/u_'+str(j)+'.pdf',bbox_inches='tight')

#%% Try part of 5th SVD mode

X = np.copy(U[:,:,5])

for i in range(180):
    for j in range(360):
        if np.abs(i-89)**3/20**3+np.abs(j-230)**3/30**3 >=1:
            X[i,j] = 0

X = sp.ndimage.filters.gaussian_filter(X,3)

coeffs = np.zeros(T)
for t in range(T):
    coeffs[t] = np.sum(X*sst[:,:,t])
```

```
plt.figure(figsize=(6,3))
plt.gca().invert_yaxis()
plt.gca().axes.get_xaxis().set_ticks([])
plt.gca().axes.get_yaxis().set_ticks([])

m = plt.imshow(X)
m.set_rasterized(True)
#    plt.axis('image')

plt.savefig('img/el_nino_mode.pdf',bbox_inches='tight')


#%% Try single single point instead
coeffs = sst[90,215]

#%% Plot raw projection

plt.figure(figsize=(6,3))
ax = plt.gca()
plt.ylabel('temperature (degree C)')
plt.xlabel('date')

ax2 = ax.twinx()
ax2.fill_between(ONI_dates,0,ONI,color='.8')
ax.plot(dates,coeffs/100,color='k')
plt.ylabel('ONI')

ax.set_zorder(ax2.get_zorder()+1)
ax.patch.set_visible(False)
#ax.yaxis.set_major_formatter(FormatStrFormatter('%1.0e'))
plt.savefig('img/el_nino_coeff.pdf',bbox_inches='tight')

#%% Plot Filtered projection

# define low pass filter
b, a = signal.butter(4, .036)

plt.figure(figsize=(6,3))
ax1 = plt.gca()
plt.ylabel('temperature (degree C)')

ax2 = ax1.twinx()

# plot ONI data
ax2.fill_between(ONI_dates,0,ONI,color='.8')
plt.ylabel('ONI')

# plot filtered data
ax1.set_zorder(ax2.get_zorder()+1)
ax1.patch.set_visible(False)
#ax1.yaxis.set_major_formatter(FormatStrFormatter('%1.0e'))
ax1.plot(dates,coeffs/100,color='.5')
ax1.plot(dates,sp.signal.filtfilt(b, a, coeffs/100, method="gust"),color='k')

plt.savefig('img/el_nino_filtered.pdf',bbox_inches='tight')

#%% Take DWT

widths = np.arange(1,100)
```

```
cwtmatr = signal.cwt(coeffs, signal.ricker, widths)
m = plt.pcolormesh(dates,widths,cwtmatr)
m.set_rasterized(True)
plt.savefig('img/el_nino_wavelet_transform.pdf',bbox_inches='tight')


#%% Try Time Slice

U,s,vh = svd_of_slice(np.s_[:,:,361:413])

plt.figure()
plt.scatter(np.arange(16),s[:16],color='k')
plt.yscale('log')
plt.show()
#plt.savefig('img/svd/'+str(T)+'/s.pdf',bbox_inches='tight')

for j in range(8):
    plt.figure(figsize=(6,3))
    ax1 = plt.gca()
    ax2 = ax1.twinx()

    # plot ONI data
#    ax2.fill_between(ONI_dates,0,ONI,color='.8')

    # plot filtered data
    ax1.set_zorder(ax2.get_zorder()+1)
    ax1.patch.set_visible(False)
    ax1.plot(dates[361:413],s[j]*vh[j],color='k')
    ax1.yaxis.set_major_formatter(FormatStrFormatter('%1.0e'))

    plt.savefig('img/svd/shortT/vh_'+str(j)+'.pdf',bbox_inches='tight')


    plt.figure(figsize=(6,3))

    ax1 = plt.gca()

    ax1.invert_yaxis()
    ax1.axes.get_xaxis().set_ticks([])
    ax1.axes.get_yaxis().set_ticks([])

    m = plt.imshow(U[:,:,j] , vmin=-5e-3,vmax=9e-3)
    m.set_rasterized(True)
    plt.axis('image')
    plt.savefig('img/svd/shortT/u_'+str(j)+'.pdf',bbox_inches='tight')

#%% Try Spatial Slice

U,s,vh = svd_of_slice(np.s_[70:110,150:300,:])

plt.figure()
plt.scatter(np.arange(16),s[:16],color='k')
plt.yscale('log')
plt.show()
#plt.savefig('img/svd/'+str(T)+'/s.pdf',bbox_inches='tight')

for j in range(8):
    plt.figure(figsize=(6,3))
    ax1 = plt.gca()
    ax2 = ax1.twinx()

    # plot ONI data
    ax2.fill_between(ONI_dates,0,ONI,color='.8')

    # plot filtered data
    ax1.set_zorder(ax2.get_zorder()+1)
    ax1.patch.set_visible(False)
    ax1.plot(dates,s[j]*vh[j],color='k')
    ax1.yaxis.set_major_formatter(FormatStrFormatter('%1.0e'))

    plt.savefig('img/svd/smallMN/vh_'+str(j)+'.pdf',bbox_inches='tight')


    plt.figure(figsize=(6,2))

    ax1 = plt.gca()

    ax1.invert_yaxis()
    ax1.axes.get_xaxis().set_ticks([])
    ax1.axes.get_yaxis().set_ticks([])

    m = plt.imshow(U[:,:,j] , vmin=-5e-3,vmax=9e-3)
    m.set_rasterized(True)
    plt.axis('image')
    plt.savefig('img/svd/smallMN/u_'+str(j)+'.pdf',bbox_inches='tight')

#%% Set up Data for Neural Net

ONI_threshold = 2;

# classify points as above or below threshold
classes = np.sign(ONI_interp-ONI_threshold)

# get points where ONI first goes above ONI_threshold
transition_ind=np.where(np.convolve(classes,[1,-1],mode='valid')>0)

time_to_jump = np.zeros(T)
time_to_jump[transition_ind] = np.ones(len(transition_ind))

# count backwards from ones
current_time_to_jump = 0
jumping = False
for j in range(T):
    if time_to_jump[-j]==1:
        current_time_to_jump=1
        jumping=True
    elif jumping:
        time_to_jump[-j]=current_time_to_jump
        current_time_to_jump+=1

last_known_jump_index = np.where(time_to_jump!=0)[0][-1]

#%% Train Neural Network

import keras
from keras import optimizers
from keras.models import Model,Sequential,load_model
from keras.layers import Input,Dense,Reshape,Activation
from keras import backend as K
from keras.utils.generic_utils import get_custom_objects
from keras.utils import plot_model

def rad_bas(x):
    return K.exp(-x**2)
get_custom_objects().update({’rad_bas’: Activation(rad_bas)})

def tan_sig(x):
    return 2/(1+K.exp(-2*x))-1
get_custom_objects().update({’tan_sig’: Activation(tan_sig)})

input_data = np.reshape(sst,(-1,T))[:,:,last_known_jump_index]
target_data = time_to_jump[:last_known_jump_index]

#%%
model = Sequential()
model.add(Dense(M*N, activation=’tan_sig’, use_bias=True, input_shape=(M*N,)))
model.add(Dense(M*N, activation=’sigmoid’, use_bias=True))
model.add(Dense(M*N, activation=’linear’, use_bias=True))
model.add(Dense(1))


adam1 = optimizers.Adam(lr=.005, beta_1=0.9, beta_2=0.999, epsilon=None, decay=1e-5, amsgrad=
    True, clipvalue=0.5)
model.compile(loss=’mean_squared_error’, optimizer=adam1, metrics=[’accuracy’])


model.fit(input_data, target_data, epochs=1000, batch_size=1000, shuffle=True,
    validation_split=0.05)

#%%
def progress_bar(percent):
    length = 40
    pos = round(length*percent)
#     clear_output(wait=True)
    return(’[’+’’*pos+’ ’*(length-pos)+’]  ’+str(int(100*percent))+’%’)

#%% Animate Full Data

%matplotlib auto

fig, ax = plt.subplots(figsize=(5, 3))
fig.gca().invert_yaxis()
#ax.axis(’image’)

mesh = ax.pcolormesh(sst[:,:,0]*mask,vmin=min_temp,vmax=max_temp)
fig.colorbar(mesh)

def animate(i):
    mesh.set_array((sst[:,:,i]).reshape(-1))
    plt.title(progress_bar(i/T),name=’ubuntu mono’)


anim = FuncAnimation(
    fig, animate, interval=6, frames=T-1)

plt.draw()
plt.show()
```