Low Dimensional Structures in Zebrafish Neural Recordings

Seth Hirsh, Tun Sheng Tan, Theodore Zhao[a,a,b]

[a]*Department of Physics, University of Washington, Seattle*
[b]*Department of Applied Mathematics, University of Washington, Seattle*

Abstract

Several supervised and unsupervised learning methods are applied to zebrafish brain data to extract out underlying structure in the animal's neural activity. Using the unsupervised learning techniques correlations were found between neural activity and when the fish was moving, while with the supervised learning techniques whether the fish was moving left or right could be predicted well beyond the accuracy achieved by random guessing.

1. Introduction and Overview

Understanding how information flows through a biological neural network is of great interest in neuroscience in addition to computer science and artificial intelligence. Recent advances in brain sheet imaging, have allowed us to record nearly 80% of neural activity for zebrafish larvae [1]. With nearly 100,000 neurons this yields large amounts of high-dimensional data. Due to the simplicity of zebrafish behavior, it is likely that this data lies on a much lower-dimensional manifold. Consequently, these recordings are a prime candidate for data-driven analyses. Here, we apply several supervised and unsupervised learning techniques as a first attempt to extracting some of the underlying structure.

2. Theoretical Background

2.1. Zebrafish Neural Recordings

Zebrafish larvae have been a prime candidate for studying neural activity in vertebrates due to their transparent bodies. This has allowed for the development of a noninvasive neural recording technique known as whole-brain light-sheet imaging. In this method the fish is first injected with the GCaMP5G protein. Whenever neural activity occurs calcium ions are emitted from the neurons, which cause the protein to fluoresce.

This fluorescence can then be measured by scanning over the brain using two lasers (Figure 1). One laser beam scans from the side of the fish while the other beam scans the region between the eyes. With this process, the entire brain can be scanned 40 times per second [2]. Two laser beams are required so as to not interfere with visual stimuli passed to the zebrafish's eyes.

Note that in order to apply this method the zebrafish must remain paralyzed. Thus, to record measurements of a zebrafish's neural activity while "moving" a technique was developed which utilizes the vertebrate's optomotor response (OMR). By projecting vertical gratings (black and red lines in Figure 1), which move forward or backward from the fish's

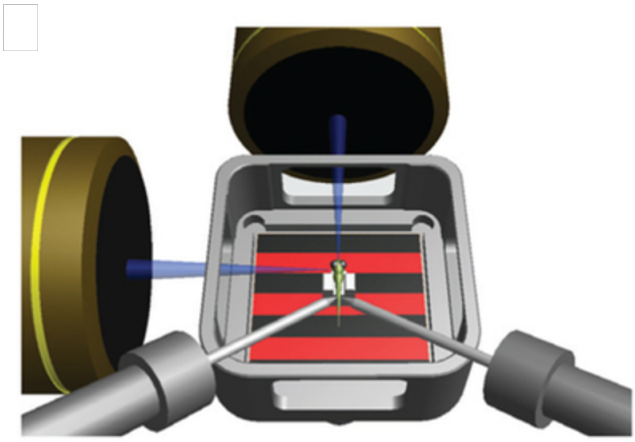


Figure 1: Setup of experiment used to measure neural activity of zebrafish (Vladimirov, 2014).

perspective, the zebrafish can be tricked into believing that it is falling downstream or upstream, respectively. The zebrafish will then attempt to respond to this by swimming so as to remain at rest with respect to the current.

Although the fish is paralyzed, the nerves in its tail will still attempt to propel it. Thus, by placing electrodes in its tail the "movement" of the fish can be independently measured.

For the following analysis, two video recordings were obtained (Figure 2). In the first analysis the video used contains a z-projection of the fish in addition to indicators in the top left which show the moving of the grating (black and white bars) and an expanding and contracting circle, whose radius corresponds to the response measured by the electrodes.

In the second video, the gratings were moved left and right (in contrast to forwards and backwards). Thus, measurements from the tail revealed how much the fish was attempting to move in these two directions.

2.2. Unsupervised Methods

2.2.1. Windowed DMD and Multi-Resolution DMD (mrDMD)

Regular dynamic mode decomposition (DMD) fails when dealing with a multi-scale system. For the zebrafish brain, neurons fire at different frequencies. The brain regulates the involuntary functions of the body (like breathing and heart-beat) and voluntary motions like (blinking and swimming). Those frequencies can vary a lot. In addition, regular dynamic mode decomposition also fails when there is transient behavior.

One method for dealing with this type of behavior is to use a windowed DMD. With this method DMD is applied over a small time interval (window) of the data. This window is then slid in time over the data, with DMD being applied on each window.

Multi-resolution DMD (mrDMD) is another approach to incorporating transient behavior [3]. Essentially, mrDMD is a recursive application of DMD where at each step low frequencies are separated. In other words, the data x_{mrDMD} is separated into

$$x_{mrDMD} = \sum_{k=1}^{m_1} b_k \psi_k \exp^{\omega_k t} + \sum_{k=m_1+1}^{M} b_k \psi_k \exp^{\omega_k t} \tag{1}$$

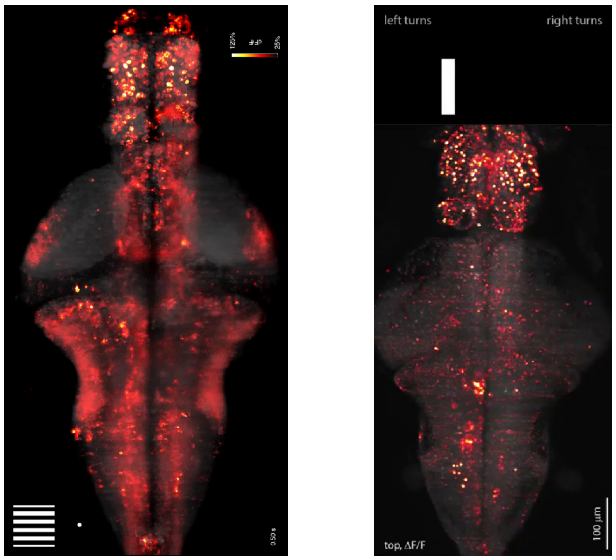


Figure 2: Sample snapshots of zebrafish neural recordings. Left figure contains bars in lower right corresponding grating. Right figure contains bar indicating whether the zebrafish is attempting a left or right turn.

where ω_k is the eigenvalue of the k^{th} mode, and ψ_k is the k^{th} mode, and b is a coefficient dependent on initial conditions. The first term on the right contains the $m+1$st lowest frequency modes and the second term contains the high frequency modes. For the next step the time intervals are split in half and DMD is only applied to the high frequency term. By repeating the process, one can achieve good spatial and temporal separation in the data. This procedure is summarized in Figure 3.

2.3. Supervised Methods

In addition to applying unsupervised learning techniques like DMD to the data, we can also apply supervised machine learning techniques when the data is labeled as different classes. Some of these techniques are briefly introduced below.

2.3.1. Linear Discriminant Analysis and Naive Bayes

Suppose our samples $\{\boldsymbol{x}_i\}$ are statistically independent and suppose each class y_i forms a Gaussian distribution. If we have an equal number of samples in each class, then by applying Bayes' Theorem we can compute the probability that a new sample $\boldsymbol{x}$ belongs to class y_i by

$$P(\boldsymbol{x}|y_i) = 2\pi^{-k/2}|\Sigma|^{-1/2}e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}))} \tag{2}$$

where μ and Σ are the mean value and covariance matrix of the training samples in class y_i, respectively and k is the length of $\boldsymbol{x}$. Thus, by Naive Bayes' model we classify $\boldsymbol{x}$ to belong in class y_j if

$$y_j = \text{argmax}_{y_i} P(\boldsymbol{x}|y_i) \tag{3}$$

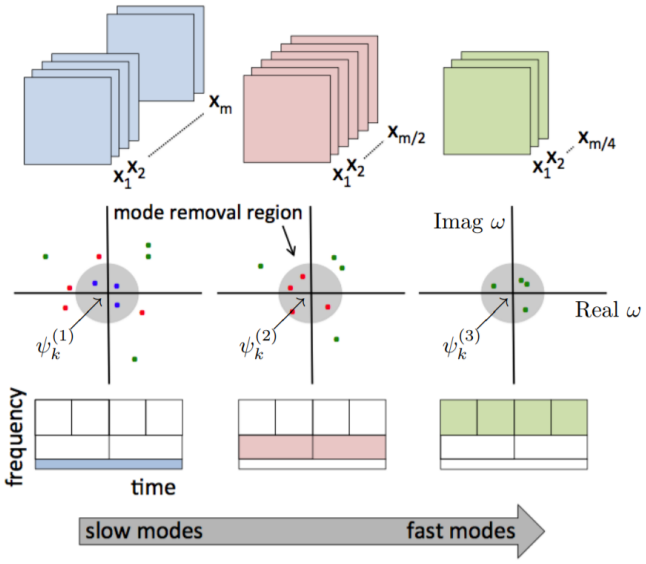


Figure 3: Visualization of mrDMD. At each level, DMD is applied separating fast modes and slow modes. Moving left to right, the time intervals are split in half and the algorithm is applied recursively (Kutz, 2016).

Pictorially, we are essential fitting multivariate normal distributions to each class in the training data and predicting the class of a test sample by determining to which class the sample has the highest probability of belonging. In general this method can produce non-linear decision boundaries.

If the classes are assumed to have the same covariance matrix (which is computed using the data from all of the classes combined), then this decision boundary is linear. This method in which a single covariance matrix is used is known as Linear Discriminant Analysis (LDA).

2.3.2. Classification Tree

The classification tree method uses a decision tree to make predictions. Each node corresponds to a cut along one of the components in the data samples, separating the data into two sets. Cuts are repeatedly applied at each level until each leaf contains data of a single class. The class of a new sample can be predicted by traversing down through the tree and at each node determining to which side of the cut the sample belongs.

3. Algorithm

For both videos, the general procedure for preparing the data was as follows:

1. Remove the impulse response (white circle and bars in video 1 and left/right bars in video 2) of the fish from the video.
2. Convert the RGB frames into grayscale and downscale the images to reduce processing memory usage in the analysis step.

For the first video, DMD and mrDMD were both performed, and the reconstructed videos were compared. After this a windowed DMD was applied. Then, we performed a k-means clustering to the set of DMD modes.

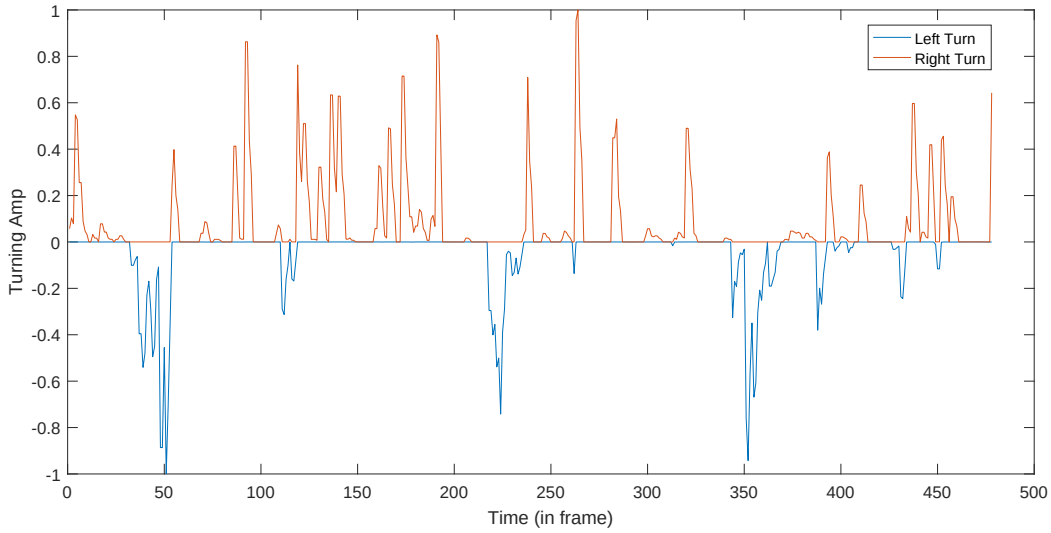


Figure 4: Turning process of the zebrafish.

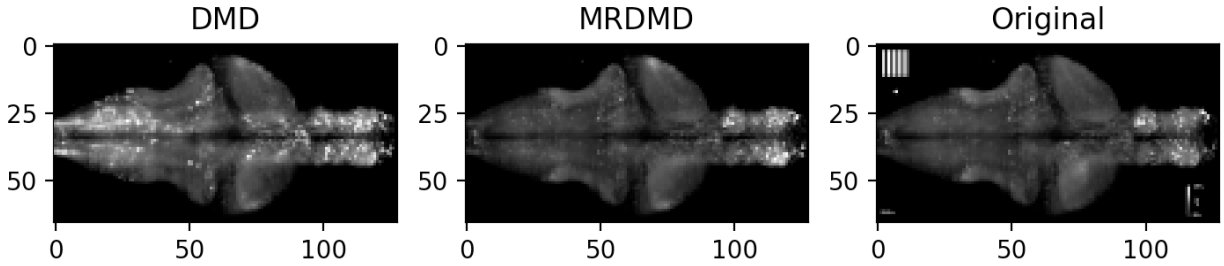


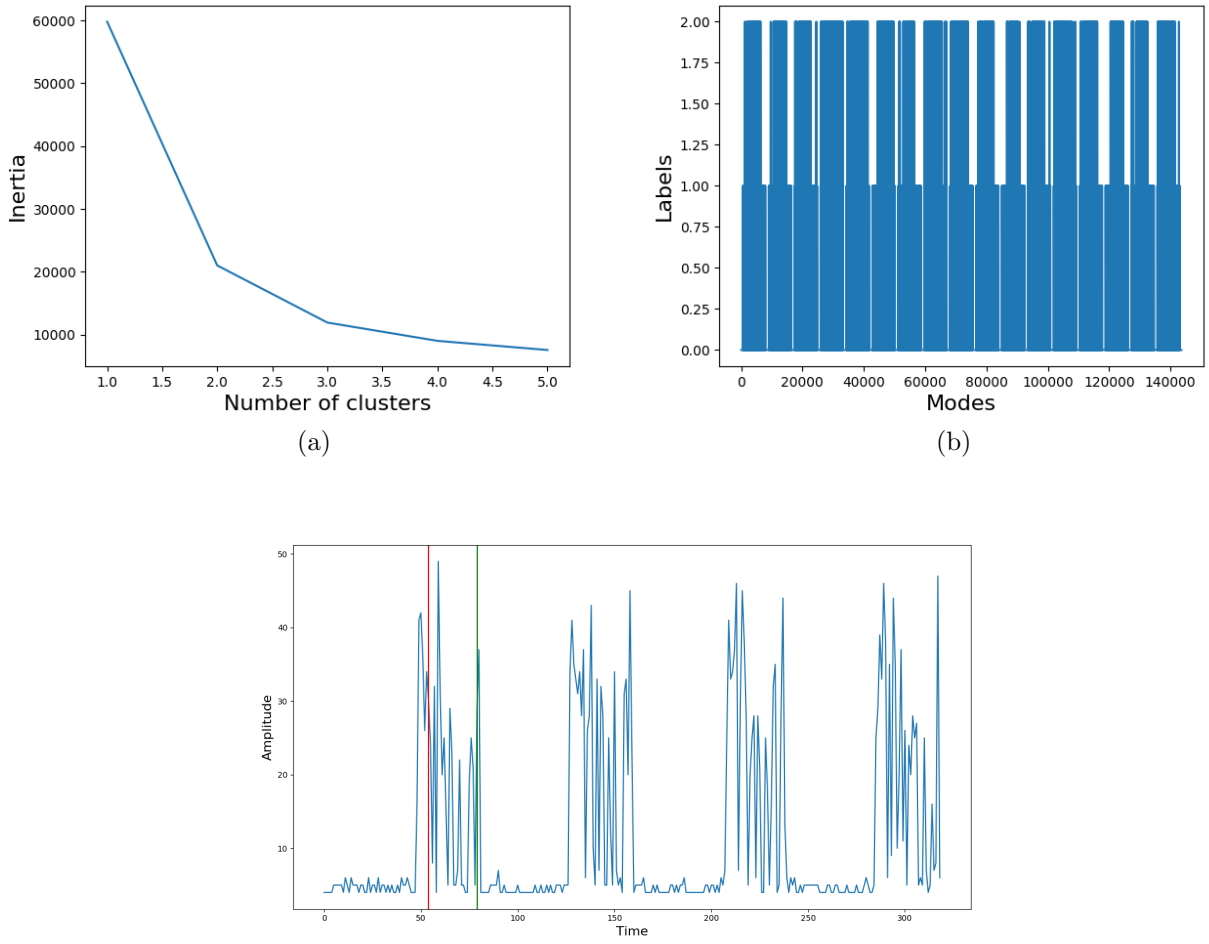
Figure 5: Comparison between DMD (left) and mrDMD (center) in reconstructing the original data (right).

The second video shows the neuron firing activity of a zebrafish as a function of its left and right turns. Using the white bars of the image, we classified which direction the zebrafish was turning. We defined the turning amplitude as the fractional height of the left and right bars in the video. By convention, we chose a right turn to have a positive value and a left turn to have a negative value. Note that if the amplitude was below a threshold of 0.005 we classified it as not turning in the frame. A plot of the resultant turning amplitude as a function of time is shown in Figure 4.

4. Computational Results

4.1. Unsupervised Learning Results

Using the fist video, we performed DMD and mrDMD on the brain activity. From the full video and exemplified by Figure 5, we noticed that DMD was not able to reconstruct the original data well. As stated before, this is an expected observation because of the transient behavior in the data. Studying the modes produced by mrDMD, we found that most of the dynamics of the zebrafish in response to the external stimuli was contained in only two of the six recursive layers.



(a) (b)

(c)

Figure 6: (a) Plot of inertia versus number of clusters. (b) Result of k-means (k=3) clustering on windowed DMD. (c) Response of fish versus time. Red and green lines represent the start-time and end-time of a window.

Next, we perform a windowed DMD on the data (with a window of 25 frames and an overlap of 75%) and performed a clustering of the resultant DMD modes. We used the elbow method to determine the number of clusters. In other words, by repeating the k-mean clustering with different k, we observed that the sum of the square of the distances of each data point to the closest cluster center (known as the inertia of the cluster) stops decreasing significantly at $k = 3$ as shown in Figure 6. Thus, we determined that the pixel modes should be clustered into three groups.

We attempted to correlate the window with the radius of the white circle (Figure 6c). As stated before the radius of the circle corresponds to how much the fish is trying to move. Figure 6b shows the modes sorted by starting time and the class they were clustered into (either Class 0, Class 1, or Class 2). There are 17 windows and Figure 6b shows that there are indeed 17 windows. Also, we see that when the number of modes in Class 2 in a window is greater than the number in Class 1, then we can conclude that the window lies in the time interval where the fish is trying to move.

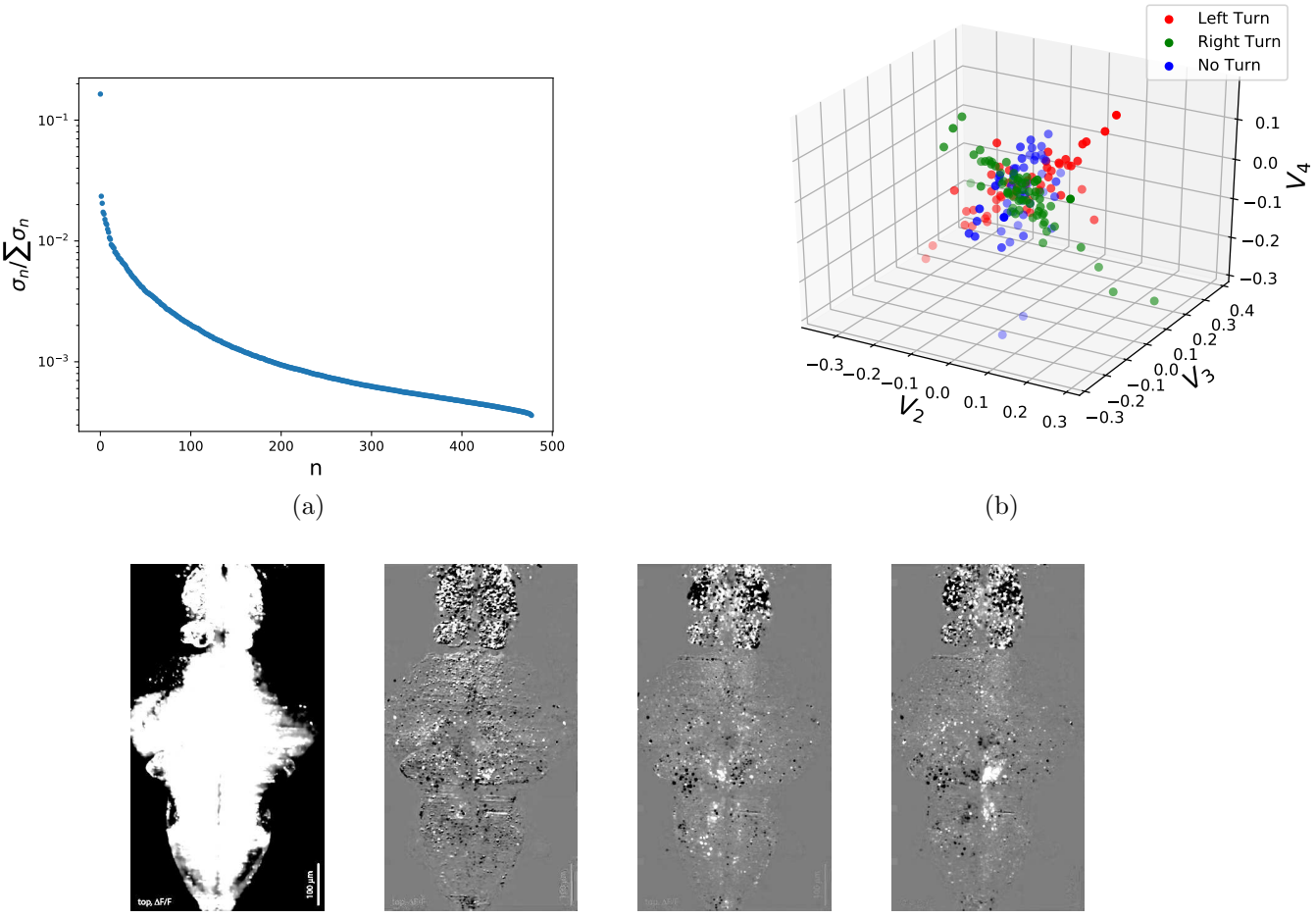
4.2. Supervised Learning Results

Before applying the supervised learning method, we first applied singular value decomposition to the data (SVD). The singular value spectrum is shown in in Figure 7a. The values are decaying fast in the first few modes, which possess the majority of the energy. The first modes with the strongest singular values are the "average frames" indicating the backgrounds. The first four modes are shown in Figure 7c. The coefficients in front of the second, third and fourth SVD modes (which correspond to the second, third and fourth columns of V in $X = U\Sigma V^*$) of these frames are plotted in a 3-D space in Figure 7b. Labeling each sample with its corresponding class, it appears that all three classes lie on separate planes but all share a common center.

We then used LDA, the naive Bayes model, and a decision tree to train and predict the turn direction. We randomly took 80% of the data in each group to be our training set and the other 20% as the test set. This was then performed five times for each model. The prediction results are shown in the confusion matrices in Figure 8. Note that Figure 8 shows the sum of the results over the five trials. We can see that the classification tree gives the best results among the three, with an accuracy of 62% (compared to accuracies of 45% and 48% for LDA and naive Bayes, respectively). This makes sense, since LDA and naive Bayes do not work well when the data shares a common center.

5. Summary and Conclusions

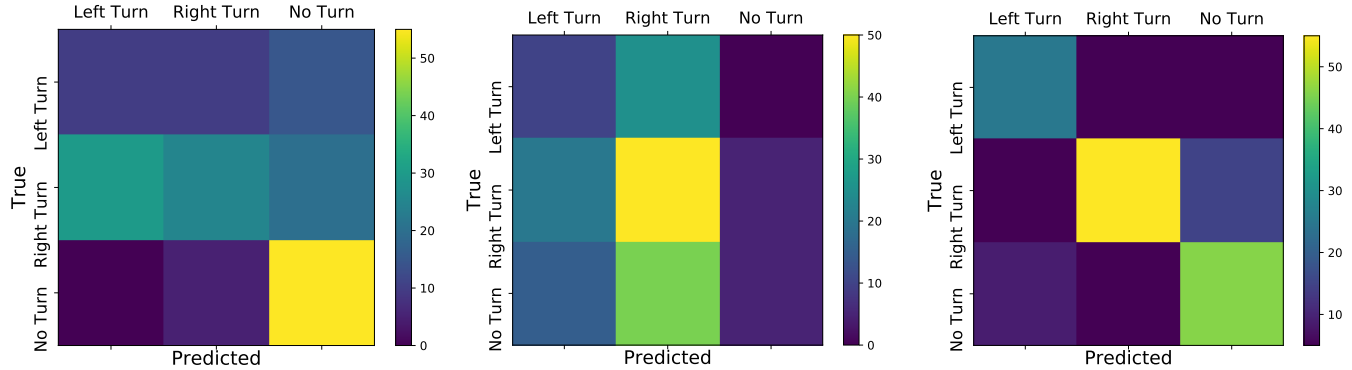
We have introduced several supervised and unsupervised data-driven techniques as methods for understanding some of the low-dimensional structure found in neural activity of the zebrafish. By applying a windowed DMD, correlations were seen between periods of activity and the extracted DMD modes. Using a decision tree we were able to predict whether the fish was moving left, right, or was stationary with 62% accuracy, well beyond the 33% accuracy found from random guessing. Although these results are promising, it is important to apply the developed models to other independent zebrafish data sets to validate these results.



(a) (b)

(c)

Figure 7: (a) Spectrum of singular values σ_n normalized such that the sum of the values is one. (b) Projection of data onto second, third, and fourth singular vectors. (c) First four modes of video using SVD.



(a) Naive Bayes (b) Linear Discriminant (c) Decision Tree

Figure 8: Prediction performance for Naive Bayes, LDA, and decision tree.

[1] M. B. Ahrens, M. B. Orger, D. N. Robson, J. M. Li, and P. J. Keller, "Whole-brain functional imaging at cellular resolution using light-sheet microscopy," *Nat Meth*, vol. 10, pp. 413–420, 05 2013.

[2] N. Vladimirov, Y. Mu, T. Kawashima, D. V. Bennett, C.-T. Yang, L. L. Looger, P. J. Keller, J. Freeman, and M. B. Ahrens, "Light-sheet functional imaging in fictively behaving zebrafish," *Nature methods*, 2014.

[3] J. N. Kutz, X. Fu, and S. L. Brunton, "Multi-Resolution Dynamic Mode Decomposition," *ArXiv e-prints*, June 2015.

[4] J. Kutz, X. Fu, and S. Brunton, "Multi-resolution dynamic mode decomposition; 2016," *Preprint. Available.*

[5] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[6] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online; accessed 2017-03-16].

[7] R. Taylor, "Multi-resolution dmd," Aug 2016.

Appendix A. Python functions used

1. skimage.color.rgb2gray - convert 3D RGB image to 2D grayscale image.
2. numpy.dot - returns matrix product of two given matrices
3. scipy.linalg.svd - given a data matrix returns the correspond left singular vectors, singular values, and right singular vectors.
4. numpy.linalg.pinv - given a data matrix returns the Moore-Penrose pseudoinverse.
5. numpy.linalg.eig - given a data matrix returns the corresponding eigenvalues and eigenvectors.
6. cv2.VideoCapture - loads an mp4 video file into python.
7. imageio.get_reader - loads MOV and avi video files into python.

Appendix B. MATLAB functions used

1. permute - change the order of dimensions (convert from phython video form into matlab video form)
2. rgb2gray - transform the frames to gray scale
3. imshow - display the image matrix as an image
4. reshape - reshape the dimension of matrices
5. svd - perform SVD on a matrix
6. randperm - randomly change the order of intergers 1:n
7. fitctree - fit classification tree using training data and labels
8. fitNaiveBayes - fit Naive Bayes model using training data and labels

9. predict - predict the classes of test data using the learning model from previous model fitting
10. classify - using linear discriminant analysis to classify the test data from training set and labels
11. ind2vec - convert indices to vectors
12. plotconfusion - plot classification confusion matrix using prediction and true values

Appendix C. Python codes

```
# This scripts runs DMD and MRDMD & compares them

import numpy as np
import matplotlib
#matplotlib.use('Qt4Agg')
import matplotlib.pyplot as plt
from scipy.fftpack import dct

# Required for dmd
from skimage.color import rgb2gray
from skimage.transform import downscale_local_mean
from dmd import *
from visualize import *

# Load Video
# First index is frame, then height, width, and RGB
video = load('ZebrafishBrain.mp4')
numFrames = video.shape[0]

# Get Brain portion
brain = getBrain(video)

# Get Dot portion
circle = video.copy()
circle[:,:120,:200,:] = 0
circle[:, :, 125:,:] = 0
circle[:, 220:, :,:] = 0

DEBUG = False
if DEBUG:
    plt.imshow(brain[50])
    plt.show()

################################################################################
scale = 10
test = downscale_local_mean(rgb2gray(video[0,:,:,:]), (scale, scale))
height = test.shape[0]
width = test.shape[1]

# row = Time, col = Image
print("Downscaling images")
```

```
gray_video = np.array([downscale_local_mean(rgb2gray(frame), (scale, scale)).flatten() for
gray_brain = np.array([downscale_local_mean(rgb2gray(frame), (scale, scale)).flatten() for
gray_circle = np.array([downscale_local_mean(rgb2gray(frame), (scale, scale)).flatten() fo

binary_cirlce = gray_circle.copy()
binary_cirlce[binary_cirlce < 0.1] = 0
binary_cirlce[binary_cirlce >= 0.1] = 1

# col = Time, row = Image
print("Transposing matrices")
gray_video = gray_video.T
gray_brain = gray_brain.T
gray_circle = gray_circle.T
binary_cirlce = binary_cirlce.T

# if DEBUG:
#   print("Shape = ", gray_video.shape)
#   im = plt.imshow(gray_video[:, 40].reshape(height, width), cmap='gray')
#   plt.show()

# extract input-output matrices
X = gray_brain[:,:-1]
Y = gray_brain[:,1:]
t = np.linspace(0, numFrames, numFrames)

#-----------------------------------------------------------------
# Regular DMD
r = reducedRank(X)
print("Rank reduced ", r)
mu, Phi = dmd(X,Y)
Psi = timeEvolve(X[:,0], t, mu, Phi)
D_dmd = dot(Phi, Psi)
# animateDMD(gray_video, D_dmd, numFrames, height, width)

#-----------------------------------------------------------------
# Multi-resolution DMD
# DMD on brain
nodes1 = mrdmd(gray_brain)
D_mrdmd1 = [dot(*stitch(nodes1, i)) for i in range(5)]
rD_rec = sum(D_mrdmd1)
print("rD_rec = ", rD_rec.shape)

# DMD on circle (NOT REQUIRED)
# nodes2 = mrdmd(binary_cirlce)
# D_mrdmd2 = [dot(*stitch(nodes2, i)) for i in range(5)]

# total = np.abs(D_mrdmd1) + np.abs(D_mrdmd2)
# animateMRDMD(gray_video, total, numFrames, height, width)
#-----------------------------------------------------------------
# Animate MRDMD
animateMRDMD(gray_video, D_mrdmd1, numFrames, height, width)
# animateMRDMD(gray_video, D_mrdmd2, numFrames, height, width)

# Compare DMD and MRDMD
```

```
animateCompare(gray_video, D_dmd, rD_rec, numFrames, height, width)


# This scripts plots the radius of the white circle as a function of time

import numpy as np
import matplotlib
matplotlib.use('Qt4Agg')
import matplotlib.pyplot as plt

from visualize import load
from skimage.color import rgb2gray
from skimage.draw import circle_perimeter

from scipy.fftpack import fft, dct

# Return the radius of the circle in a video
def getRadius(video):
    # Get only the circle
    circle = video.copy()
    circle[:,:120,:200,:] = 0
    circle[:, :, 125:,:] = 0
    circle[:, 220:, :,:] = 0
    radii = []

    # Binarize image
    print("Start grayscaling")
    binary_cirlce = [rgb2gray(frames) for frames in circle]
    print("Convert to array")
    binary_cirlce = np.array(binary_cirlce)
    print("Start binarizing")
    binary_cirlce[binary_cirlce < 0.1] = 0
    binary_cirlce[binary_cirlce >= 0.1] = 1


    print("Calculating radius")
    return [radius(image) for image in binary_cirlce]

# Returns the radius of the circle in an image
def radius(image):
    stdy = (max(np.nonzero(image)[0]) - min(np.nonzero(image)[0]))//2
    stdx = (max(np.nonzero(image)[1]) - min(np.nonzero(image)[1]))//2
    return int(round(stdx + stdy)/2)

# Animate the detected circle and return the radius
def visualizeRadius(video, debug=True):
    numFrames = video.shape[0]

    # Get only the circle
    circle = video.copy()
    circle[:,:120,:200,:] = 0
    circle[:, :, 125:,:] = 0
    circle[:, 220:, :,:] = 0
    radii = []
```

```
    # Binarize image
    print("Start grayscaling")
    binary_cirlce = [rgb2gray(frames) for frames in circle]
    print("Convert to array")
    binary_cirlce = np.array(binary_cirlce)
    print("Start binarizing")
    binary_cirlce[binary_cirlce < 0.1] = 0
    binary_cirlce[binary_cirlce >= 0.1] = 1

    for t in range(numFrames):
        if debug:
            # Zoom in view
            image = circle[120:220, :125, t]
        else:
            image = circle[:,:, t]

        # Compute circle parameters
        y_pos = int(np.average(np.nonzero(image)[0]))
        x_pos = int(np.average(np.nonzero(image)[1]))
        radius = radius(image)
        radii.append(radius)

        if debug:
            # Draw circles on images
            cx, cy = circle_perimeter(x_pos, y_pos, radius)
            image[cy, cx] = 0.5
            plt.imshow(image, cmap = 'gray')
            plt.pause(0.01)
    if debug:
        plt.figure()
        plt.show()

    return radii


# Main Function

# Load Video
video = load('/home/weyl000/Documents/Assignment/zebrafish582/ZebrafishBrain.mp4')
radii = getRadius(video)
plt.figure()
plt.subplot(1,3,1)
plt.plot(radii)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Impulse of Zebrafish versus Time")
plt.subplot(1,3,2)
plt.plot(np.abs(fft(radii)))
plt.xlabel("Frequency")
plt.ylabel("Amplitude")
plt.title("Absolute Value of Fast Fourier Transform")
plt.subplot(1,3,3)
plt.plot(dct(radii))
```

```
plt.xlabel("Frequency")
plt.ylabel("Amplitude")
plt.title("Discrete Cosine Transform")
plt.show()

np.save("radius.npy", radii)
```

Appendix D. Matlab codes

```
video=permute(file,[2,3,4,1]);%Change into MATLAB form
imshow(video(:,:,:,50))
T=size(video,4);
%Finding the boarder lines of videos of different view
for j=100:200
    if video(j,50,1,1)>0
        HoriLine=j;
        break
    end
end
for j=200:400
    if video(100,j,1,1)>0
        LeftLine=j;
        break
    end
end
for j=100:300
    if video(100,end-j,1,1)>0
        RightLine=size(video,2)-j;
        break
    end
end

%Seperating the videos
TV=video(1:HoriLine-1,LeftLine+1:RightLine-1,:,:);
LV=video(HoriLine+1:end,1:LeftLine-1,:,:);
MV=video(HoriLine+1:end,LeftLine+1:RightLine-1,:,:);
RV=video(HoriLine+1:end,RightLine+1:end,:,:);

%Get the height change of the turning bars
TurnBars=video(1:HoriLine-1,1:LeftLine-1,:,:);
LT=zeros(1,T);
RT=zeros(1,T);
for t=1:T
    LT(t)=sum(sum(TurnBars(35:end-5,80:120,1,t)));
end
for t=1:T
    RT(t)=sum(sum(TurnBars(35:end-5,200:240,1,t)));
end
%Turning amptitude in percentage
LT=LT/max(LT);
RT=RT/max(RT);
plot(1:T,-LT,1:T,RT);
xlabel('Time (in frames)');
```

```
ylabel('Amplitude');

%Reshape the videos into matrices
%Top one
Top=[];
xt=size(TV,2);
yt=size(TV,1);
for t=1:T
    G=reshape(rgb2gray(TV(:,:,:,t)),xt*yt,1);
    Top=[Top,G];
end
Top=double(Top);
%Left one
Left=[];
xl=size(LV,2);
yl=size(LV,1);
for t=1:T
    G=reshape(rgb2gray(LV(:,:,:,t)),xl*yl,1);
    Left=[Left,G];
end
Left=double(Left);
%Middle one
Middle=[];
xm=size(MV,2);
ym=size(MV,1);
for t=1:T
    G=reshape(rgb2gray(MV(:,:,:,t)),xm*ym,1);
    Middle=[Middle,G];
end
Middle=double(Middle);
%Right one
Right=[];
xr=size(RV,2);
yr=size(RV,1);
for t=1:T
    G=reshape(rgb2gray(RV(:,:,:,t)),xr*yr,1);
    Right=[Right,G];
end
Right=double(Right);

%SVD on the matrices
[Ut,St,Vt]=svd(Top-mean(Top),'econ');
[Ul,Sl,Vl]=svd(Left-mean(Left),'econ');
[Um,Sm,Vm]=svd(Middle-mean(Middle),'econ');
[Ur,Sr,Vr]=svd(Right-mean(Right),'econ');

subplot(2,2,1);
semilogy(diag(St),'ko');
xlabel('Mode (Top Video)');
ylabel('Singular Value (log-plot)');
subplot(2,2,2);
semilogy(diag(Sl),'ko');
xlabel('Mode (Left Video)');
ylabel('Singular Value (log-plot)');
```

```
subplot(2,2,3);
semilogy(diag(Sm),'ko');
xlabel('Mode (Middle Video)');
ylabel('Singular Value (log-plot)');
subplot(2,2,4);
semilogy(diag(Sr),'ko');
xlabel('Mode (Right Video)');
ylabel('Singular Value (log-plot)');

%Plot the modes
subplot(1,4,1);
imshow(int8(reshape(Sl(1,1).*Ul(:,1),yl,xl)));
subplot(1,4,2);
imshow(int8(reshape(Sl(2,2).*Ul(:,2),yl,xl)));
subplot(1,4,3);
imshow(int8(reshape(Sl(3,3).*Ul(:,3),yl,xl)));
subplot(1,4,4);
imshow(int8(reshape(Sl(4,4).*Ul(:,4),yl,xl)));


%Classification of turning
%Find left, right and still frames
idx_L = find(LT > 0.15);
idx_R = find(RT > 0.3);
idx_S = find(LT+RT < 0.00005);
nL = length(idx_L);
nR = length(idx_R);
nS = length(idx_S);

%Left video frames L,R,S
L_left = Left(:,idx_L);
R_left = Left(:,idx_R);
S_left = Left(:,idx_S);

Turn_L = [L_left, R_left, S_left];
[U_turn, S_turn, V_turn] = svd(Turn_L,'econ');
plot3(V_turn(2,1:nL),V_turn(3,1:nL),V_turn(4,1:nL),'ro',V_turn(2,nL+1:nL+nR),V_turn(3,nL+1
xlabel('V2');
ylabel('V3');
zlabel('V4');

%Learning data with labels
LearningData = [V_turn(2,1:nL)',V_turn(3,1:nL)',V_turn(4,1:nL)';V_turn(2,nL+1:nL+nR)',V_tur
Label = [-ones(nL,1);ones(nR,1);zeros(nS,1)];

%Classification
%Randomly choose 80% as training
qL = randperm(nL);
qR = randperm(nR);
qS = randperm(nS);
train = [V_turn(2,qL(1:round(0.8*nL)))',V_turn(3,qL(1:round(0.8*nL)))',V_turn(4,qL(1:round
turn_class = zeros(size(train,1),1);
turn_class(1:round(0.8*nL)) = -1;
turn_class(round(0.8*nL)+1:round(0.8*nL)+round(0.8*nR)) = 1;
```

```
test = [V_turn(2,qL(round(0.8*nL)+1:end))',V_turn(3,qL(round(0.8*nL)+1:end))',V_turn(4,qL(
test_true = [-ones(nL-round(0.8*nL),1);ones(nR-round(0.8*nR),1);zeros(nS-round(0.8*nS),1)];

%Linear discriminant
pre = classify(test,train,turn_class);
bar(pre);
a = ind2vec(pre'+2);
b = ind2vec(test_true'+2);
plotconfusion(b,a)

%Naive Bayes
nb = fitNaiveBayes(train,turn_class);
pre = predict(nb,test);
bar(pre);
a = ind2vec(pre'+2);
b = ind2vec(test_true'+2);
plotconfusion(b,a)

%Classification tree
tree = fitctree(train,turn_class);
pre = predict(tree,test);
a = ind2vec(pre'+2);
b = ind2vec(test_true'+2);
plotconfusion(b,a)

%Five trials
pre = [pre1;pre2;pre3;pre4;pre5];
a = ind2vec(pre'+2);
b = ind2vec([test_true;test_true;test_true;test_true;test_true]'+2);
plotconfusion(b,a)
```