AMATH 482 Homework 5

Gg Tran

March 14th, 2019

Abstract

Dynamical Mode Decomposition (DMD) is used to approximate a nonlinear dynamical system whose governing equation is unknown. We implement the DMD to perform video extraction. All videos are from YouTube. For each video, the background and foreground correspond, respectively, to the low-rank DMD modes and the sparse data matrix. Overall, across all three sample videos, the algorithm successfully differentiates between the foreground and the background.

I. Introduction and Overview

Dynamical Mode Decomposition (DMD) is a data-driven method to model complex, nonlinear system whose underlying equations are unknown. Using data snapshots in time of the system, the DMD can build the low-dimensional model of the system and make short-timed prediction. The DMD assumes that each data snapshot is collected over regularly spaced interval of time Δt.[2] The DMD incorporates principle component analysis and Fourier modes. The advantage of DMD is that it requires no assumptions about the system.

II. Theoretical Background

Given a nonlinear system that is evolving continuously in time, the DMD approximates such system using a continuous linear system

$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x}$, with $\mathbf{x}(t)$ denoting the state of the system at time t.

The solutions to the system in continuous time can be represented with

$$\mathbf{x}(t) = \sum_{k=1}^{n} \boldsymbol{\phi}_k \exp(\omega_k t)\, b_k = \boldsymbol{\Phi} \exp(\Omega t)\mathbf{b} \quad [1]$$

whereas the solutions at each discretized time can be computed with

$$\mathbf{x}_k = \sum_{j=1}^{r} \boldsymbol{\phi}_j \lambda_j^k b_j = \boldsymbol{\Phi}\Lambda^k \mathbf{b} \quad [1]$$

with $r << n$, $\boldsymbol{\Phi}$ are the DMD modes, Λ^k are the discrete-time eigenvalues, $\mathbf{b}$ is the initial amplitude of each mode. [1] ω_k is the DMD eigenvalues converted to Fourier modes, with $\omega_k = \ln(\lambda_k)/\Delta t$. The growth and decay of these Fourier modes will limit how far in time the model can credibly make prediction. [2]

Koopman operator

Since the DMD seeks to build a linear model that best approximates a nonlinear system, the goal is to find the optimal Koopman operator $\mathbf{A}$ that maps one data snapshot to another. All m data snapshots in time are arranged into matrix $\mathbf{X}$. To reduce approximation error, data matrix $\mathbf{X}$ is split into two matrices with column vectors:

$$\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad ... \quad \mathbf{x_{m-1}}] \text{ and } \mathbf{X'} = [\mathbf{x}_2 \quad \mathbf{x}_3 \quad ... \quad \mathbf{x_m}]$$

From here, $\mathbf{A}$ can be computed by: $\mathbf{A} = \mathbf{X'}\mathbf{X}^\dagger$, with $\mathbf{X}^\dagger$: Moore-Penrose pseudo-inverse.

Koopman operator $\mathbf{A}$ is also used to make prediction, with $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k$.

Steps in building the Dynamic Mode Decomposition algorithm

Do SVD on $\mathbf{X}$ to get $\mathrm{X} = \mathrm{U}\Sigma\mathrm{V}^*$ $\rightarrow$ Pick the number of rank base on the number of dominant modes. Reduce the rank of $\mathrm{X} = \mathrm{U}\Sigma\mathrm{V}^*$ $\rightarrow$ Build $\tilde{A}$ using $\mathbf{X}'$ and the rank-truncated $\mathbf{U}$, $\mathbf{S}$, $\mathbf{V}$ so that $\tilde{A} = \mathbf{U}^*\mathbf{X}'\mathbf{V}\Sigma^{-1}$ $\rightarrow$ Do eigendecomposition on $\tilde{A}$ to obtain eigenvectors $\mathbf{W}$ and diagonal matrix with eigenvalues Λ $\rightarrow$ Construct the $\mathbf{\Phi}$ matrix whose columns are DMD eigenvectors: $\mathbf{\Phi} = \mathbf{X}'\mathbf{V}\Sigma^{-1}\mathbf{W}$ $\rightarrow$ Find the coeeficients b_k of the initial data snapshot $\mathbf{x_1}$ through the pseudo-inverse $\rightarrow$ Build the data matrix with the DMD modes. [1]

III. Algorithm Implementation and Development

- Building the data matrix: represent the video as a sequence of picture frames $\rightarrow$ Reshape each frame as a column vector $\rightarrow$ Split the data matrix `X` into `X1` and `X2` (shifted data).
- Apply DMD algorithm to reconstruct the data using DMD modes. We use the `DMD.m` function (see Appendix B) from Kutz's textbook to construct `Xdmd`. We find 2 singular values that are the dominant modes, but we set rank `r = 10` as this yield better video reconstruction.
- Split the `Xdmd` into background matrix and foreground matrix. `XSparseDmd` represents the foreground. `XdmdLowRank` represents the background. We construct `XdmdLowRank` using `abs(omega) < 0.5`. This cutoff is chosen based on manually looking at all the `omega` values and deciding which cutoff range is closest to 0 while still captures all the background.
- Get the real value of `XSparseDmd` $\rightarrow$ Put all residual negative values in `XSparseDmd` into `R` $\rightarrow$ Subtract `R` from `XSparseDmd`.
- Reconstruct the background and foreground frames.

IV. Computational Results

Overall, with `rank = 10`, across all 3 videos, we are able to separate the foreground (right) from the background (left). There are still some contours of the foreground residing in the extracted background, but those contours are completely unmoving.

In the corgi surveillance video (figure 1), we can easily separate out the foreground (the moving corgi) and the background (the kitchen). In the video, the only thing moving is the corgi, while the surrounding background (the kitchen) is completely still. However, while we can extract all the corgi's movement into a separate foreground, we are unable to extract a clearer shape of the moving corgi. We surmise that this might due to the poor quality of the surveillance camera: The video is very blurry, showing poor sharpness and contrast between objects.

In the boxing surveillance video (figure 2), the foreground is the two moving boxers, and the background is the audience. There are some slight movements in the background (e.g the referee walking back and forth). However the algorithm extracts out a very clear representation and movement of the two fighters.

In the corgi twerking video (figure 3), the camera is moving so both the background and the foreground are moving. However the video quality is great, with clear sharpness and contrast between objects. The algorithm does a pretty good job separating out the main background, while still captures the shaking movement of the corgi. In some frames the background (the room) is also moving back and forth through time, so the algorithm mistakenly identifies some parts of the background as being the foreground.

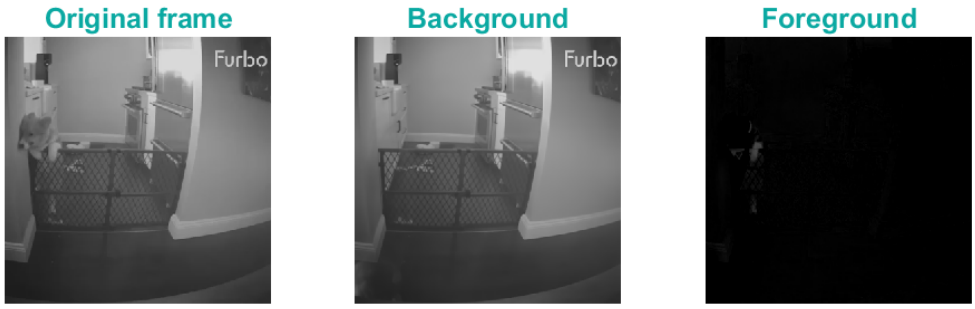

Figure 1: Video surveillance featuring a corgi trying to climb through the fence, frame 11. The only thing moving in the video is the corgi. The video has poor quality, and the images of the corgi are very blurry. Nonetheless the algorithm correctly captures all the movement of the dog into the foreground matrices.

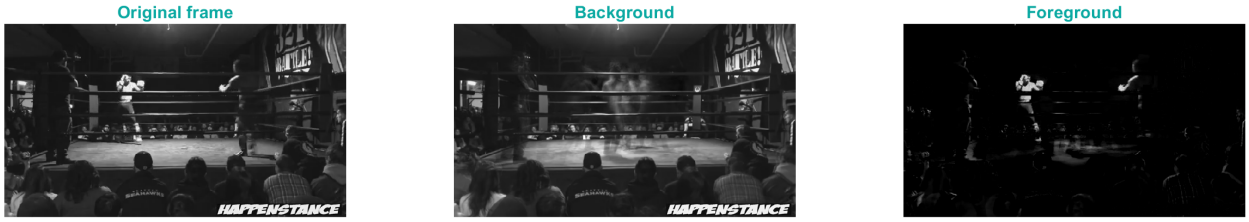

Figure 2: Video surveillance featuring a boxing fight between two people, frame 11. The algorithm nicely separates out the foreground (the two fighters fighting) from the background (the audience), even though in some frames, the audience in the background are moving.

Figure 3: Camera recording of a corgi twerking, frame 130. Due to camera shakes, both the background and the foreground (the corgi) are moving. This has create challenges to the algorithm, but overall DMD still manages to capture the overall background and foreground.

V. Summary and Conclusions

The effectiveness of the algorithm is affected by the quality of the movie clip: The boxing and corgi twerking video have better quality, e.g better light contrast, so the algorithm better separates the background and the foreground. On the other hand, the first corgi video has poorer quality where objects edges and contrasts are blurrier. The foreground seems "blend in" with the background, thus making it harder to extract a clear foreground.

We choose not to add the residual matrix back into the background. Doing so tends to put the details belonging to the foreground into the background, resulting in worse decomposition.

Appendix A: MATLAB functions

VideoReader(): Read into MATLAB the video.

[Phi, omega, lambda, b, Xdmd] = DMD(X1,X2,rank,dt): Perform DMD on the data matrix **X1** and its shifted frame *X2*. The outputs are Phi: the DMD modes, omega: the associated Fourier modes, lambda: the discrete-time DMD eigenvalues, b: The coefficients of the initial snapshot, and Xdmd is the matrix constructed by Phi, omega, and b.

Appendix B: MATLAB codes

DMD.m, from (1)

```
function [Phi,omega,lambda,b,Xdmd] = DMD(X1,X2,r,dt)
% function [Phi,omega,lambda,b,Xdmd] = DMD(X1,X2,r,dt)
% Computes the Dynamic Mode Decomposition of X1, X2
%
% INPUTS:
% X1 = X, data matrix
% X2 = X', shifted data matrix
% Columns of X1 and X2 are state snapshots
% r = target rank of SVD
% dt = time step advancing X1 to X2 (X to X')
%
% OUTPUTS:
% Phi, the DMD modes
% omega, the continuous-time DMD eigenvalues
% lambda, the discrete-time DMD eigenvalues
% b, a vector of magnitudes of modes Phi
% Xdmd, the data matrix reconstrcted by Phi, omega, b
%% DMD
[U, S, V] = svd(X1, 'econ');
r = min(r, size(U,2));
U_r = U(:, 1:r); % truncate to rank-r
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);
Atilde = U_r' * X2 * V_r / S_r; % low-rank dynamics
[W_r, D] = eig(Atilde);
Phi = X2 * V_r / S_r * W_r; % DMD modes
lambda = diag(D); % discrete-time eigenvalues
omega = log(lambda)/dt; % continuous-time eigenvalues. Convert eigvalues into
                        %Fourier mode for prediction

%% Compute DMD mode amplitudes b
x1 = X1(:, 1);
b = Phi\x1;
%% DMD reconstruction
mm1 = size(X1, 2); % mm1 = m - 1
time_dynamics = zeros(r, mm1);
t = (0:mm1-1)*dt; % time vector
for iter = 1:mm1,
time_dynamics(:,iter) = (b.*exp(omega*t(iter)));
end;
Xdmd = Phi * time_dynamics;
```

Analysis code

```
clear all; close all; clc

%  obj=VideoReader('Video.mp4')
% obj=VideoReader('corgi.mp4')
 obj=VideoReader('corgi_twerk.mp4')
vidFrames = read(obj);
numFrames = get(obj,'numberOfFrames');

for k = 1 : numFrames
    mov(k).cdata = vidFrames(:,:,:,k);
    mov(k).colormap = [];
end
X = [];
for j=1:numFrames
%     X=frame2im(mov(j));
%     imshow(X); drawnow
      x = rgb2gray(vidFrames(:,:,:,j));
%     imshow(x); drawnow
    x = x(:);
    X = [X x];
end

%%
% load('X.mat');
X = double(X);
rank = 10;
%plot(diag(s),'ko','Linewidth', [1.5])
X1 = X(:, 1:end - 1);
X2 = X(:, 2:end);
t2 = linspace(1, obj.Duration,numFrames); t = t2(1:end -1);
dt = t(2) - t(1);
[Phi, omega, lambda, b, Xdmd] = DMD(X1,X2,rank,dt); %omega(j): associated fourier mode
%Find X DMD low rank.
%find the index p with omega value closest to 0
% p = find(abs(omega) < 1);       %Boxing
p = find(abs(omega) < 0.5);       %Corgi
XdmdLowRank = zeros(size(X1));
for k = 1:length(p)
      XdmdLowRank = XdmdLowRank + b(p(k)) .* Phi(:, p(k)) * lambda(p(k));
end

%% Get the real value of Xdmd sparse
%Put all residual negative values in X sparse into R n-by-m
R = zeros(size(XSparseDmd));
for k = 1:numel(XSparseDmd)
     if XSparseDmd(k) < 0          && abs(XSparseDmd(k)) < 1
        R(k) = XSparseDmd(k);
    end
end
XdmdLowRank2 = abs(XdmdLowRank);
XSparseDmd = XSparseDmd - R;
```

```
%%
frameWidth = get(obj,'Width');
frameHeight = get(obj,'Height');
for j=130   %j = 11 for boxing & corgi vid
    x = rgb2gray(vidFrames(:,:,:,j));
    background2 = reshape(XdmdLowRank2(:,j), frameHeight, frameWidth, []);
    foreGround = reshape(XSparseDmd(:,j), frameHeight, frameWidth, []);
    subplot(1, 3, 1), imshow(x); drawnow;
    title('Original frame', 'color', [10, 169, 162]/255)
    subplot(1, 3, 2), imshow(uint8(double(background2))); drawnow;...
        title('Background','color', [10, 169, 162]/255)
     subplot(1, 3, 3), imshow(uint8(double(foreGround))); drawnow;...
         title('Foreground', 'color', [10, 169, 162]/255)
end
```

Reference

1. Kutz, Nathan J. "Computational Methods for Data Analysis." *In Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data.* Oxford University Press, 2013.
2. Grosek, Jacob, and J. Nathan Kutz. "Dynamic mode decomposition for real-time background/foreground separation in video." *arXiv preprint arXiv:1404.7592* (2014).