Background Subtraction in Video Streams

Author: Seoyoung Park

Abstract

These days, videos are commonly used to store information so that events of interest can be captured and analyzed with those. Videos have two main components for every frame: foreground and background. Dynamic Mode Decomposition(DMD) is a dimensional reduction algorithm that is capable of background modeling in video streams. DMD integrates Fourier transforms and singular value decomposition. Innovations in compressed sensing allow fast decomposition of video streams that scales with the intrinsic rank of the data matrix.

Introduction and Overview

In this paper, the Dynamic Mode Decomposition method is going to be used to take a video clip containing both foreground and background object and separate the video streams to both the foreground video and the background video. Matrix decomposition algorithms are allowed for the construction of low-rank features that dominate the data. DMD capitalizes on the low-rank feature extraction of SVD with an eigendecomposition in the time variable. Therefore, DMD results in a decomposition of data into spatio-temporal modes that represents the data across spatial features and assigns the correlated data to unique temporal Fourier modes.

Theoretical Background

The DMD method generates a spatio-temporal decomposition of data into a set of dynamic modes which are resulted from snapshots or measurements of a given system in time such as frames of a video sequence. The DMD is related to the Koopman spectral analysis of nonlinear dynamic systems so that the DMD builds to approximate its low-rank components.
There are two parameters in the data: n is the number of spatial points saved per time snapshot, and m is the number of snapshots taken. DMD is defined for pairs of data $\{(x_1, y_1), ..., (x_m, y_m)\}$ which makes $y_j = Ax_j$ possible, and the pairs are expected to be equispaced snapshots of some dynamical system $z(t)$, so $x_j = z((j-1)\Delta t)$ and $y_j = z(j\Delta t)$. However, since matrix A is not fully given by the snapshots, we assume the matrix A from the data in a least-squares sense.

$$A - YX^{\dagger}$$

where $X^\dagger$ is the pseudo-inverse of X. A is the minimizer of $\|AX - Y\|_F$, which can be expressed as the best fit linear system mapping X to Y, advancing $z(t)$ to $z(t + \Delta t)$.
The low-rank DMD approximation of both the eigenvalues and eigenvectors let us to construct the past, current, and future state of the system.

$$\omega_k = ln(\lambda_k)/\Delta t$$

Then the approximate solution at all future times, $\tilde{x}(t)$, is the following

$$\tilde{x}(t) = \Sigma b_k(0)\psi_k(\xi)exp(\omega_k t) = \Psi diag(exp(\omega t))b$$

where ξ are the spatial coordinates, $b_k(0)$ is the initial amplitue of each mode, Ψ is the matrix whose columns are the DMD modes ξ, $exp(\omega t)$ are the eigenvalues, and b is a vector of the coefficients of b_k. Assume that ω_p, where $p \in \{1, 2, ..., l\}$, is true for $\|\omega_p\| \approx 0$, and $\|\omega_j\|$ where $j \neq p$ is bounded away from zero.

$$X_{DMD} = b_p\varphi_p e^{\omega_p t} + \sum b_j\varphi_j e^{\omega_j t} \text{ where } j \neq p$$

$$X_{DMD}^{Low-Rank} = b_p\varphi_p e^{\omega_p t} : \text{ background video}$$

$$X_{DMD}^{Sparse} = \sum b_j\varphi_j e^{\omega_j t} \text{ where } j \neq p : \text{ foreground video}$$

$$X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}$$

Algorithm Implementation and Development

1. Get a data matrix from the video
Load the video, obtain the frame rate, convert it to the gray scale and double form and reshape the matrix to 2D in the matrix X.
2. Define matrices X1 and X2 from the data matrix
From the matrix X, set X1 = $[x_1, x_2, ...x_{m-1}]$ and X2 = $[x_2, x_3, ...x_m]$
3. Perform SVD on the matrix X1 to obtain the matrix A
After performing SVD on the X1, we get U, Σ, and V so that $X1 = U\Sigma V^*$, and we get obtain $A = U * X2 * V * \Sigma^{-1}$
4. Get the eigenvalues and eigenvectors of A to compute Φ and a vector of the coefficients b
$Aw = \lambda w$, w is an eigenvector and λ is an eigenvalue
. With the eigenvector w, we can get $\Phi = X2V\Sigma^{-1}w$. Also, from the vector of the coefficients b_k, we need to compute the initial coefficient value $b_k(0)$. If we consider the initial snapshot (x_1) at time 0, $x_1 = \Phi b$, so $b = \Phi^\dagger x_1$
5. Construct X_{DMD}

Now since we know Φ, b, and eigenvalues, we are able to reconstruct the original video in terms of DMD modes by multiplying those. X_{DMD} is n by m matrix where n is the number of pixels, m is the total number of frames.

6. Find the low rank and the index that corresponds to the low rank.

To divide the X_{DMD} to $X_{DMD}^{Low-Rank}$ and X_{DMD}^{Sparse}, we need to figure out what value of p should we use to construct $X_{DMD}^{Low-Rank}$. $\omega_k = ln(\lambda_k)/\Delta t$, so with the eigenvalues we got, take log of it and find the min so that $\|\omega_p\| \approx 0$.

7. Formulate L, background video, $b_p \varphi_p e^{\omega_p t}$

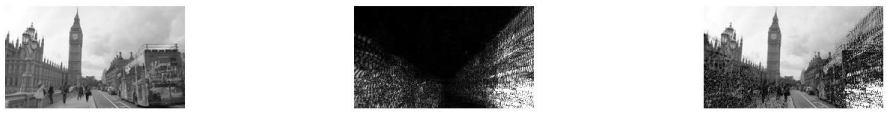
8. Formulate S, foreground video, $\sum b_j \varphi_j e^{\omega_j t}$ where $j \neq p$

Subtract absolute value of L from X_{DMD}, but this can result in S having negative values and pixel intensities can't be negative values, so residual negative values from S can be put into matrix R, then

$$X_{DMD}^{Low-Rank} = R + |X_{DMD}^{Low-Rank}|$$
$$X_{DMD}^{Sparse} = X_{DMD}^{Sparse} - R$$

9. Reshape L and S then convert them into uint8 format

Computational Results



(a) DMD (b) Foreground (c) Background

Figure 1: London - The people on the left and the bus on the right are moving in the video.

(a) DMD (b) Foreground (c) Background

Figure 2: NYC street - People on the street are walking across the road in the video.

(a) DMD

(b) Foreground

(c) Background

Figure 3: Suzzallo Library at University of Washington - A student is passing by the library in the video.

(a) DMD

(b) Foreground

(c) Background

Figure 4: 1st floor of CSE building at University of Washington - A person is going to the stairs in the video.

Four different videos were tested under dynamic mode decomposition. The videos for figure 1 and 2 are taken by other people, and those for figure 3 and 4 are taken by my phone. (a) from each figure shows 10th frame of DMD, (b) from each figure shows 10th frame of foreground video, and (c) from each figure shows 10th frame of background video. As you see, the objects moving in the video are shown in the foreground video and the other things that stay on the same spot are shown in the background video.

Summary and Conclusions

Dynamic Mode Decomposition was applied to four different videos that have objects in motion to separate those into foreground videos and background videos. $X_{DMD} = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}$; $X_{DMD}^{Low-Rank}$ represents background video ,and X_{DMD}^{Sparse} represents foreground video. As illustrated above, the objects in motion are captured in the foreground videos and others are captured in the background videos.

Reference

Pendergrass, S., S. L. Brunton, J. N. Kutz, N. B. Erichson, and T. Askham. "Dynamic Mode Decomposition for Background Modeling." 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), 2017. doi:10.1109/iccvw.2017.220.

Appendix A

OBJ = VideoReader(FILENAME) constructs a multimedia reader object, OBJ, that can read in video data from a multimedia file.
DATA, COUNTREAD= read(OBJ, COUNTREQUESTED) reads the requested number of items from the input stream.
rgb2gray converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.
V,D= eig(A) produces a diagonal matrix D of eigenvalues and a full matrix V whose columns are the corresponding eigenvectors so that A*V = V*D.
Y,I= min(X) returns the indices of the minimum values in vector I.
reshape(X,M,N) or reshape(X,[M,N]) returns the M-by-N matrix whose elements are taken columnwise from X.

Appendix B

```
clear all; close all; clc;

obj=VideoReader('London2.mp4');
dt = 1/obj.FrameRate;
vidFrames = read(obj);
numFrames = get(obj,'numberOfFrames');

for k = 1:numFrames
    frame(:, :, k) = rgb2gray(vidFrames(:, :, :, k));
end

X = double(reshape(frame,[136*240,344]));
% define matrices from the data
X1 = X(:,1:end-1); % X
X2 = X(:,2:end); % Y

[u,s,v] = svd(X1,'econ'); % take the svd of X

A = u.' * X2 * v * s^(-1); % get matrix A
[eigvec, eigval] = eig(A); % compute the eigendecomposition of A
psi = X2 * v / s * eigvec; %/ eigval ;
eigvals = diag(eigval);
```

```
b = psi\X1(:,1); % X1(:,1) = initial snapshot(x_1) at time t1 = 0

[lowRank,index] = min(abs((log(eigvals)/dt)));

t = (1:numFrames-1) .* dt;

for j = 1:343
    DMD(:,j) = psi * (eigval.^t(j)) * b;
end

L = zeros(32640,343);
for j = 1:343
    L(:,j) = psi(:,index)*(lowRank^t(j))*b(index,1);
end

S = DMD - abs(L);
vtak e
S = abs(S);
L = abs(L);
foreG = uint8(reshape(S,[136,240,343]));
backG = uint8(reshape(L,[136,240,343]));

for t = 1:160
    t
    imshow(backG(:,:,t));
end

figure(1);imshow(frame(:,:,5));
figure(2);imshow(foreG(:,:,5)); drawnow
figure(3);imshow(backG(:,:,5)); drawnow

clear all; close all; clc;

obj=VideoReader('cse_cut.mp4');
dt = 1/obj.FrameRate;
vidFrames = read(obj);
numFrames = get(obj,'numberOfFrames');

for k = 1:numFrames
```

```
    frame(:, :, k) = rgb2gray(vidFrames(:, :, :, k));
end

X = double(reshape(frame,[136*241,161]));
% define matrices from the data
X1 = X(:,1:end-1); % X
X2 = X(:,2:end); % Y

[u,s,v] = svd(X1,'econ'); % take the svd of X

A = u.' * X2 * v * inv(s); % get matrix A
[eigvec, eigval] = eig(A); % compute the eigendecomposition of A
psi = X2 * v / s * eigvec;
eigvals = diag(eigval);
b = psi\X1(:,1); % X1(:,1) = initial snapshot(x_1) at time t1 = 0

[lowRank,index] = min(abs((log(eigvals)/dt)));

DMD = zeros(32776,160);
t = (1:numFrames-1) .* dt;
for j = 1:160
    DMD(:,j) = psi * (eigval.^t(j)) * b;
end

L = zeros(32776,160);
for j = 1:160
    L(:,j) = psi(:,index)*(lowRank.^t(j))*b(index,1);
end

S = DMD - abs(L);
R = zeros(length(S), 160);
for i = 1:length(S(:,1))
    for j = 1:length(S(1,:))
        if S(i,j) < 0
            R(i,j) = S(i,j);
        end
    end
end
```

```
S = S - R;
L = R + abs(L);

S = abs(S);
L = abs(L);
foreG = uint8(reshape(S,[136,241,160]));
backG = uint8(reshape(L,[136,241,160]));

for t = 1:160
    imshow(foreG(:,:,t));
end

figure;imshow(frame(:,:,10));
figure;imshow(foreG(:,:,10)); drawnow
figure;imshow(backG(:,:,10)); drawnow

clear all; close all; clc;

obj=VideoReader('library.mp4');
dt = 1/obj.FrameRate;
vidFrames = read(obj);
numFrames = get(obj,'numberOfFrames');

for k = 1:numFrames
    frame(:, :, k) = rgb2gray(vidFrames(:, :, :, k));
end

X = double(reshape(frame,[136*241,143]));
% define matrices from the data
X1 = X(:,1:end-1); % X
X2 = X(:,2:end); % Y

[u,s,v] = svd(X1,'econ'); % take the svd of X

A = u.' * X2 * v * inv(s); % get matrix A
[eigvec, eigval] = eig(A); % compute the eigendecomposition of A
psi = X2 * v / s * eigvec;
eigvals = diag(eigval);
b = psi\X1(:,1); % X1(:,1) = initial snapshot(x_1) at time t1 = 0
```

```
[lowRank,index] = min(abs((log(eigvals)/dt)));

DMD = zeros(32776,142);
t = (1:numFrames-1) .* dt;
for j = 1:142
    DMD(:,j) = psi * (eigval.^t(j)) * b;
end

L = zeros(32776,142);
for j = 1:142
    L(:,j) = psi(:,index)*(lowRank.^t(j))*b(index,1);
end

S = DMD - abs(L);
R = zeros(length(S), 142);
for i = 1:length(S(:,1))
    for j = 1:length(S(1,:))
        if S(i,j) < 0
            R(i,j) = S(i,j);
        end
    end
end

S = S - R;
L = R + abs(L);

S = abs(S);
L = abs(L);
foreG = uint8(reshape(S,[136,241,142]));
backG = uint8(reshape(L,[136,241,142]));

for t = 1:142
    imshow(backG(:,:,t));
end

figure;imshow(frame(:,:,10));
figure;imshow(foreG(:,:,10)); drawnow
figure;imshow(backG(:,:,10)); drawnow

clear all; close all; clc;
```

```
obj=VideoReader('nyc.mp4');
dt = 1/obj.FrameRate;
vidFrames = read(obj);
numFrames = get(obj,'numberOfFrames');

for k = 1:numFrames
    frame(:, :, k) = rgb2gray(vidFrames(:, :, :, k));
end

X = double(reshape(frame,[136*240,150]));
% define matrices from the data
X1 = X(:,1:end-1); % X
X2 = X(:,2:end); % Y

[u,s,v] = svd(X1,'econ'); % take the svd of X

A = u.' * X2 * v * inv(s); % get matrix A
[eigvec, eigval] = eig(A); % compute the eigendecomposition of A
psi = X2 * v / s * eigvec;
eigvals = diag(eigval);
b = psi\X1(:,1); % X1(:,1) = initial snapshot(x_1) at time t1 = 0

[lowRank,index] = min(abs((log(eigvals)/dt)));

DMD = zeros(32640,149);
t = (1:numFrames-1) .* dt;
for j = 1:149
    DMD(:,j) = psi * (eigval.^t(j)) * b;
end

L = zeros(32640,149);
for j = 1:149
    L(:,j) = psi(:,index)*(lowRank.^t(j))*b(index,1);
end

S = DMD - abs(L);
R = zeros(length(S), 149);
for i = 1:length(S(:,1))
```

```
    for j = 1:length(S(1,:))
        if S(i,j) < 0
            R(i,j) = S(i,j);
        end
    end
end

S = S - R;
L = R + abs(L);

S = abs(S);
L = abs(L);
foreG = uint8(reshape(S,[136,240,149]));
backG = uint8(reshape(L,[136,240,149]));

for t = 1:149
    imshow(backG(:,:,t));
end

figure;imshow(frame(:,:,10));
figure;imshow(foreG(:,:,10)); drawnow
figure;imshow(backG(:,:,10)); drawnow
```