AMATH 482 Homework 5
Brad Hong
March 17, 2021

Abstract

This report will take a video, in our case a monte carlo race and a skier going down a mountain and try to separate the foreground and background using Dynamic Mode Decomposition. Utilizing SVD to find the energies of different modes through singular value analysis, we minimized the number of modes needed to clearly see the background and the foreground.

Section I. Introduction and Overview

The objective of this experiment is to use the Dynamic Mode Decomposition to separate the foreground from the background of two different videos. To do this, we first take a video and we turn it into usable data. Since each video is basically a picture taken over a period of time, we can disperse these “images” into equal intervals to apply the DMD. Once this is done, we use the SVD to find how much energy is in each mode, truncating unnecessary modes if viable. Once we have truncated values, we are able to perform DMD to produce the background of the video. Subtracting this background from the original data, we are able to get the foreground.

Section II. Theoretical Background

The SVD is a method of deconstructing a matrix into different parts much like diagonalizing a matrix. It deconstructs the matrix A below in the following way

$$A = U\Sigma V^{*} \quad (1)$$

This equation, while seems unimpressive at first, is actually quite powerful. The V* matrix is used as a change of basis matrix to align the vectors of x to the correct axis. Then multiplying by sigma stretches/shrinks the vector to size and finally the U vector properly orients the vectors. These three matrices explain the process that is undergone when Ax is performed.

One of the major uses of the SVD is analyzing the diagonal values of sigma, also known as singular values, to determine how much energy is in each mode.

Dynamic Mode Decomposition or DMD is generally used with the purpose of creating high-dimensional models that both model and predict values. It utilizes the low-dimensionality of observed data to not only create amazing models of high-dimensional data seen in the world but also forecast that data to see how it moves in the future.

It does this through the use of the Koopman Operator.

$$\mathbf{x}_{j+1} = \mathbf{A}\mathbf{x}_j, \quad (1)$$

Which is a linear and time-independent operator that moves a model in time by simply multiplying a vector by a matrix. In other words, we go from time j to j+1 by multiplying the space vector x_j with the matrix A.

$$\mathbf{X}_1^{M-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_2 & \cdots & \mathbf{x}_{M-1} \end{bmatrix}, \quad (2)$$

Observe how the matrix above can be rewritten with the Koopman operator below:

$$\mathbf{X}_1^{M-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{A}\mathbf{x}_1 & \mathbf{A}^2\mathbf{x}_1 & \cdots & \mathbf{A}^{M-2}\mathbf{x}_1 \end{bmatrix}. \quad (3)$$

Each column can be represented by simply multiplying the previous column by a matrix A. From this matrix, we can derive this equation below:

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{X}_1^{M-1} + \mathbf{r}e_{M-1}^T, \quad (4)$$

Where we are multiplying each column of X_1^{M-1} with A once again to get X_2^M. However, since the M-1 value is not a part of our basis there will be an error which is represented by the last term in equation 4.

By combining the SVD and the Koopman operator above, we are able to diagonalize X_1^{M-1} to U, Σ,and V values.

$$\mathbf{X}_2^M = \mathbf{A}\mathbf{U}\Sigma\mathbf{V}^* + \mathbf{r}e_{M-1}^T. \quad (5)$$

With some simplifications, we are able to get the equation below:

$$\mathbf{U}^*\mathbf{A}\mathbf{U} = \underbrace{\mathbf{U}^*\mathbf{X}_2^M\mathbf{V}\Sigma^{-1}}_{=:\tilde{\mathbf{S}}}. \quad (6)$$

This equation provides us with S-tilde, which is a similar matrix to A. This means that both S-tilde and A have the same eigenvalues and eigenvectors. This information can be used to create our low-dimensional approximation of the linear mapping of the data.

The beauty of DMD is the way it does not require a governing equation to work, it only needs data to effectively create a model.

Section III. Algorithm Implementation and Development

The implementation of the algorithm was done in the following steps:

(1) First we loaded in the video using VideoReader() to get data on height, width, number of frames, and frame rate of the video.
(2) We first reshaped the data into a column vector and to truncate our data, converted the data into black and white. This was done frame by frame, where each frame was added onto a matrix X.
(3) We used the matrix X to create the X_1^{M-1} and X_2^M matrices.

(4) SVD was performed on X_1^{M-1} to find the singular values. We used these values to determine how many modes were necessary for the DMD.
(5) The USV values were truncated to only accommodate the necessary modes.
(6) DMD was applied to produce the model of the background.
(7) We subtracted the DMD value we got from step 6 to the original video to get the foreground of the video.

Section IV. Computational Results

The first video we analyzed was the ski_drop_low.mp4 file. The singular values of Sigma were around $1.4*10^6$ for the first mode, $5.063*10^4$ for the second mode, and $3.50*10^4$ for the third mode. The monte_carlo_low.mp4 file had value $1.87*10^6$ for the first mode, $2.1*10^5$ for the second mode, and $9.7*10^4$ for the third mode. As you can see, compared to the third mode and onwards, the first two modes held the most information.

For both cases, it was decided that 3 modes would be enough to encapsulate the background and foreground of the data.

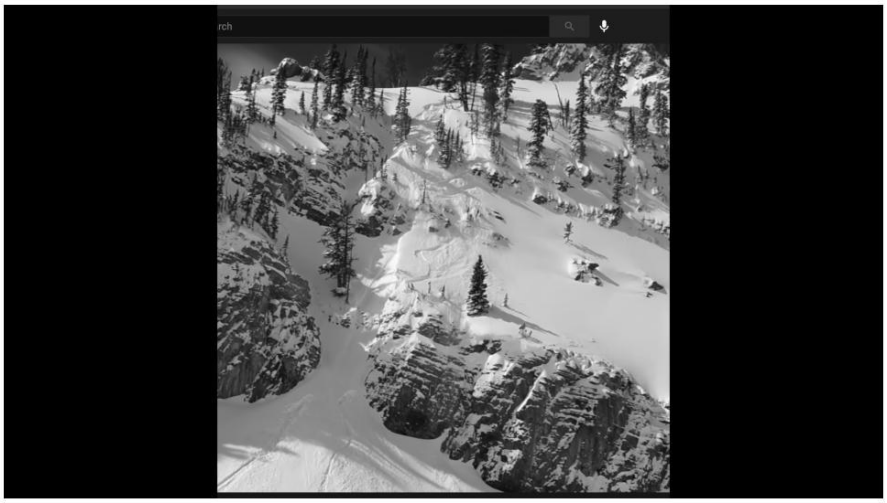

Image 1: The Background of the Ski Video from DMD

As you can see above, the DMD did an amazing job at recreating the background for the ski video. Below are the results of when this background is subtracted from the original image.

Image 2: The Foreground of the Ski Video

As you can see above, it is not the best reproduction of the skier but this is to be expected. The skier in the original video was extremely hard to see and the small details are probably encoded in some of the later modes rather than the first three. However, we are able to see where the difference is happening, which is where the skier was at the time of the snapshot.

Image 3: The Background of the Monte Carlo Race Video

As you can see, the background is once again reproduced in great detail, the only part missing is the flag and of course, the racers, which is what we wanted.

Image 4: The Foreground of the Monte Carlo Race Video

In contrast to the foreground of the skier video, the foreground of this video came out much nicer. There were less details to pick out so it did a much better job as such. The two cars are easily seen and the flag is slightly visible on the top left.

Section V. Summary and Conclusions

In general, I think the DMD did an amazing job at producing the backgrounds of both videos. I was genuinely amazed at how clear the pictures were and almost felt like something was wrong at first. On the other hand, the foreground of both could use a bit of work. I imagine if we were to use more modes, the foreground would turn out better since we would be able to see more of the details of the frames. This in turn would make the backgrounds a little bit worse. In conclusion, the DMD is an amazing tool that only requires data to be effective. The fact that it can create these stunning images out of only data is amazing.

Appendix A. MATLAB functions used and brief implementation explanation

VideoReader()

This function was used to load in the videos. It held values of the height, width, number of frames, and frame rate which all were used to perform the DMD

readFrame()
This function was used in conjunction with the object created by VideoReader() to get each frame of the video.

svd()
Used to get the Single Value Decomposition of a matrix. A second parameter ‘econ’ is used to use the simplified version of the SVD to save time.
This function return values U, S, and V each of which correspond to U, Sigma, and V from the SVD algorithm. Note that V should be transposed to get V*.

imshow()
Used to display an image to the user.

Appendix B. MATLAB codes

```
vid = VideoReader('ski_drop_low.mp4');
dt = 1/vid.FrameRate;
t = 0:dt:vid.Duration;
nFrames = vid.NumFrames;
height = vid.Height;
width = vid.Width;
X = zeros(height*width, nFrames);

i = 0;
while hasFrame(vid)
  i = i + 1;
  f = readFrame(vid);
  fg = rgb2gray(f);
  sz = size(fg,1) * size(fg,2);
  image = reshape(fg(:,:),sz,1);
  X(:,i) = image;
end
%%
X1 = X(:, 1:end-1);
X2 = X(:, 2:end);
%%
[U,S,V] = svd(X1, 'econ');
numModes = 3;
U_modes = U(:, 1:numModes);
S_modes = S(1:numModes, 1:numModes);
V_modes = V(:, 1:numModes);
%%
S_tilda = U_modes' * X2 * V_modes * diag(1./diag(S_modes));
```

```
[eV, D] = eig(S_tilda);
mu = diag(D);
omega = log(mu)/dt;
Phi = U_modes*eV;
%%
y0 = Phi\X1(:,1);
u_modes = zeros(length(y0), length(t));
for iter = 1:length(t)
    u_modes(:, iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi*u_modes;
%%
for i = 1:nFrames
    image = reshape(u_dmd(:,i),height,width);
    r_image = uint8(real(image));
    imshow(r_image);
    drawnow
end

%%
u_dmd_fg = X - abs(u_dmd);
for i = 1:nFrames
    image = reshape(u_dmd_fg(:,i),height,width);
    r_image = uint8(real(image));
    imshow(r_image);
    drawnow
end
```