Homework 1
Linear Algebra

CMU 11-755/18-797: Machine Learning for Signal Processing (Fall 2022)

OUT: September 2nd, 2022
DUE: **September 20th, 2022, 11:59 PM**

START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 3.4"). Second, write your solution independently: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only.
- **Submitting your work:** Assignments should be submitted as PDFs using Canvas unless explicitly stated otherwise. Please submit all results as `report_{YourAndrewID}.pdf` in you submission. **Each derivation/proof should be completed on a separate page**. Submissions can be handwritten, but should be labeled and clearly legible. Else, submissions can be written in LaTeX. Upon submission, label each question using the template provided by Canvas. Please refer to Piazza for detailed instruction for joining Canvas and submitting your homework.
- **Programming**: All programming portions of the assignments should be submitted to Canvas as well. Please `zip` all the code and output files together, and submit the compressed file together with the pdf report. We will not be using this for autograding, meaning you may use any programming language you desire, though Python and MATLAB are common choices.

1 Linear Algebra

1.1 Rotational Matrices

1. A rotation in 3-D space (whose cartesian coordinates we will call x, y and z as usual) is characterized by three angles. We will characterize them as a rotation around the x-axis, a rotation around the y-axis, and a rotation around the z-axis.
   Derive the rotation matrix R_1 that transforms a vector $[x, y, z]^\top$ to a new vector $[\hat{x}, \hat{y}, \hat{z}]^\top$ by rotating it counterclockwise by angle θ around the x-axis, then an angle δ around the y-axis, and finally an angle ϕ around the z-axis.
   Derive the rotation matrix R_2 that transforms a vector $[x, y, z]^\top$ to a new vector $[\hat{x}, \hat{y}, \hat{z}]^\top$ by rotating it counterclockwise by an angle δ around the y-axis, then an angle θ around the x-axis, and finally an angle ϕ around the z-axis.
2. Confirm that $R_1 R_1^\top = R_2 R_2^\top = I$

2 Moore-Penrose Inverse

The pseudoinverse we covered in lecture is more formally known as the Moore-Penrose inverse. The Moore-Penrose inverse was independently discovered throughout the 20th century. Its name is due to E.H. Moore, an influential mathematician and first head of the mathematics department at the University of Chicago, and Roger Penrose, a mathematician and physicist with too many awards to count.

In lecture, we covered two pseudoinverses for two different cases. The one which we will explore here is the pseudoinverse for the under-determined case:

$$T^+ = T^\top (T T^\top)^{-1}.$$

2.1 Moore-Penrose Conditions

Wikipedia defines the pseudoinverse of $A \in \mathbb{K}^{m \times n}$ as $A^+ \in \mathbb{K}^{n \times m}$ which satisfies:

1. $AA^+A = A$
2. $A^+AA^+ = A^+$
3. $(AA^+)^* = AA^+$
4. $(A^+A)^* = A^+A$

Here, A^* refers to the conjugate transpose of A. Wikipedia assumes that A is defined over a given field $\mathbb{K}$; however, we will restrict our discussion and this problem to $\mathbb{R}$, the field of real numbers. In this case, the conjugate transpose becomes regular matrix transposition. The four conditions above are referred to as the Moore-Penrose conditions.

Using our definition of the pseudoinverse for the under-determined case, verify that each of the Moore-Penrose conditions holds.

2.2 Pinv and SVD

The pseudo-inverse of a matrix can be calculated via singular value decomposition. If we represent a matrix A as $A = U\Sigma V^*$, then you can show that $A^+ = V\Sigma^+ U^*$. The popular Python package Numpy uses this fact to calculate the pseudoinverse.

Show that the pseudoinverse of A can be computed by $A^+ = V\Sigma^+ U^*$. That is, show that each of the Moore-Penrose conditions in Section 2.1 hold when we define the pseudoinverse of A as above. Here, U, Σ, and V

are all the usual components from singular value decomposition. Please provide at least one reason why this may be a good implementation.

N.B.: In this problem, you will see that a lot of quantities either cancel out or become the identity matrix. In your homework submission, you need to clearly explain why a given quantity would reduce to the identity matrix or cancel out. For example, saying "$BB^T = I$" will not get you any points. Instead, saying "$BB^T = I$ since B is an orthogonal matrix, and the inverse of an orthogonal matrix is its transpose" will not only get you points but it will also make the course staff smile.

2.3 Bonus problem: SVD

Singular Value Decomposition decomposes any matrix X as

$$X = USV^\top,$$

where V is the matrix of right singular vectors, U is the matrix of left singular vectors, and S is a diagonal matrix of singular values. We learned how to interpret these in class.

Let V_i represent the columns of V, U_i be the columns of U, and S_i be the i^{th} diagonal entry of S. By the definition of SVD, $\|V_i\|^2 = 1$, and $V_i^\top V_j = 0 \ \forall \ i \neq j$, i.e. the right singular vectors are orthonormal. Similarly the left singular vectors too are orthonormal, i.e. $\|U_i\|^2 = 1$, and $U_i^\top U_j = 0 \ \forall \ i \neq j$.

When X is viewed as a data-container matrix the "energy" in the data is given by $\mathcal{E}(X) = \sum_{i,j} X_{ij}^2$, where X_{ij} is the $(i,j)^{\text{th}}$ entry of X.

Show that

$$\mathcal{E}(X) = \sum_i S_i^2.$$

Hint: $\mathcal{E}(X) = \text{trace}(X^\top X)$.

3 Music Transcription

3.1 Projection

The song "Misirlou" is played on the guitar in the file `Misirlou.wav` which can be found in the folder `hw1materials`. You may recognize this tune from the "Pulp Fiction" movie or the song "Pump It" by The Black Eyed Peas.

A set of notes from a guitar can be found in the folder `hw1materials/notes_scale`. These are notes from what western music calls the double harmonic major scale, also known as the Byzantine scale. You are required to transcribe the music. For transcription you must determine how each of these notes is played to compose the music, i.e., what we called the score in the lecture.

You need to compute the spectrogram of the music file using your language/toolbox of choice, such as Python (Scipy or Librosa) or MATLAB. First, read and load the audio file at 16000 Hz sample rate.

If you are using Python, you can use Librosa to load the wav file as follows (we also recommend using the `numpy` package for matrix operations below if you use python):

```
import librosa
audio, sr = librosa.load(filename, sr = 16000)
```

Next, we can compute the complex Short-Time Fourier Transform (STFT) of the signal and its magnitude spectrogram. Use 2048 sample windows, which correspond to 64 ms analysis windows; overlap/hop length of 256 samples to 64 frames by second of signal. Different toolboxes should provide similar spectrograms. If you are using the Python Librosa library, you can use the following command:

```
spectrogram = librosa.stft(audio, n_fft=2048, hop_length=256, center=False, win_length=2048)
M = abs(spectrogram)
phase = spectrogram/(M + 2.2204e-16)
```

In this case, **M** represents the music file and should be a matrix, where the rows correspond to the frequencies and the columns to time. A visualization (see the documentation online for `librosa.display.specshow`) of this matrix (spectrogram) should look like in Figure 1.

To represent notes, you also need to compute the spectrogram of each note file. However, unlike the music file, we need to represent the matrix just as a one column vector. Hence, we can choose only one vector, or compute the mean of the matrix across time, etc. In this example, we select the middle column:

```
# n is the spectrogram of the note
import math
middle = n[:, int(math.ceil(n.shape[1]/2))]
```

To focus on the most relevant frequencies, we can clean up and normalize the note as follows:

```
middle[middle < (max(middle)/100)] = 0
```

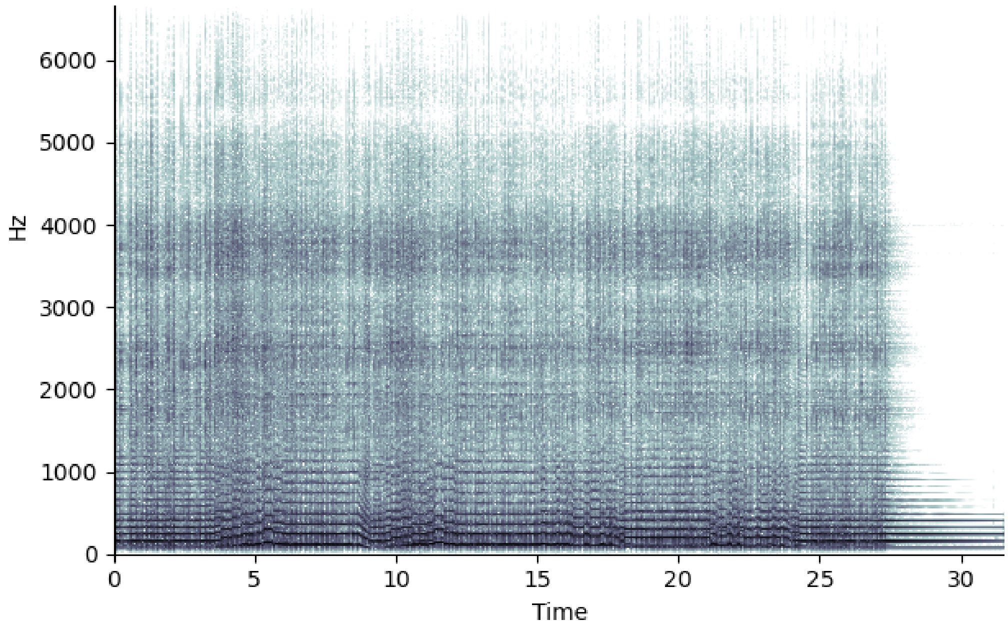

Figure 1: Spectrogram of `Misirlou.wav`

Finally, you need to normalize this vector as follows,

```
middle = middle/np.linalg.norm(middle) # import numpy as np
```

1. Compute the joint contribution of all notes to the entire music.
   Mathematically, if $\mathbf{N} = [N_1, N_2, ...]$ is the note matrix where the individual columns are the notes, find the matrix $\mathbf{W}$ such that $\mathbf{NW} \approx \mathbf{M}$, or that produce a small error $||\mathbf{M} - \mathbf{NW}||_F^2$. The i_{th} row of $\mathbf{W}$ is the transcription of the i_{th} note. Submit the matrix $\mathbf{W}$ as `problem1.csv` together with your code.

2. Recompose the music by "playing" each note according to the transcription you found in last question. Set all negative elements in W to zero and compute $\hat{M} = \mathbf{N}W$. Report the value of $||\mathbf{M} - \hat{\mathbf{M}}||_F^2 = \sum_{i,j}(\mathbf{M}_{i,j} - \hat{\mathbf{M}}_{i,j})^2$ and submit the recomposed music named as `resythensized_proj.wav` file.

   To recover the signal from the reconstructed spectrogram $\hat{\mathbf{M}}$ we need to use the phase matrix we computed earlier from the original signal. Combine both and compute the Inverse-STFT to obtain a vector and then write them into a wav file. To compute the STFT and then write the wav file you can use the following python command:

   ```
   # Latest Librosa doesn't have an audio write function. Use PySoundFile instead.
   import soundfile as sf
   signal_hat = librosa.istft(M_hat*phase, hop_length=256, center=False, win_length=2048)
   sf.write("resynthensized_proj.wav", signal_hat, 16000)
   ```

3.2 Optimization and non-negative decomposition

A projection of the music magnitude spectrogram (which are non-negative) onto a set of notes will result in negative weights for some notes. To explain this, let $\mathbf{M}$ be the (magnitude) spectrogram of the music, which is a matrix of size $D \times T$, where D is the size of the Fourier Transform and T is the number of spectral vectors in the signal. Let $\mathbf{N}$ be a matrix of notes of size $D \times K$, where K is the number of notes and each column D is the magnitude spectral vector of one note.

Conventional projection of $\mathbf{M}$ onto the notes $\mathbf{N}$ computes the following approximation:

$$\hat{\mathbf{M}} = \mathbf{N}\mathbf{W}$$

where $||\mathbf{M} - \hat{\mathbf{M}}||_F^2 = \sum_{i,j}(\mathbf{M}_{i,j} - \hat{\mathbf{M}_{i,j}})^2$ is minimized. Here, $||\mathbf{M} - \hat{\mathbf{M}}||_F$ is known as the Frobenius norm of $\mathbf{M} - \hat{\mathbf{M}}$, where $\mathbf{M}_{i,j}$ is the $(i,j)^{th}$ entry of $\mathbf{M}$ and $\hat{\mathbf{M}}_{i,j}$ is similarly the $(i,j)^{th}$ entry of $\hat{\mathbf{M}}$. We will use later the definition of the Frobenius norm.

$\hat{\mathbf{M}}$ is the projection of $\mathbf{M}$ onto $\mathbf{N}$. Moreover, $\mathbf{W}$ is given by $\mathbf{W} = pinv(\mathbf{N})\mathbf{M}$ and $\mathbf{W}$ can be viewed as the transcription of $\mathbf{M}$ in terms of the notes in $\mathbf{N}$. So, the j^{th} column of $\mathbf{M}$, which we represent as M_j is the spectrum in the j^{th} frame of the music, which are approximated by the notes in $\mathbf{N}$ as follows:

$$\mathbf{M_j} = \sum_i \mathbf{N}_i \mathbf{W_{i,j}}$$

where $\mathbf{N_i}$, the i^{th} column of $\mathbf{N}$ represents the i^{th} note and $\mathbf{W_{i,j}}$ is the (contribution) weight assigned to the i^{th} note in composing the j^{th} frame of the music.

The problem is that in this computation, we will frequently find $\mathbf{W}_{i,j}$ values to be negative. In other words, this model requires you to subtract some notes, since $\mathbf{W}_{i,j}\mathbf{N}_i$ will have negative entries. Clearly, this is an unreasonable operation intuitively; when we actually play music, we never unplay a note (which is what playing a negative note would be).

Also, $\hat{\mathbf{M}}$ may have negative entries due to the values in $\mathbf{W}$. In other words, our projection of $\mathbf{M}$ onto the notes in $\mathbf{N}$ can result in negative spectral magnitudes in some frequencies at certain times. Again, this is meaningless physically – spectral magnitudes cannot, by definition, be negative.

Hence, we will compute the approximation $\hat{\mathbf{M}} = \mathbf{N}\mathbf{W}$ with the constraint that the entities of $\mathbf{W}$ must always be greater than or equal to 0, *i.e.* they must be non-negative. To do so we will use a simple gradient descent algorithm which minimizes the error $||\mathbf{M} - \mathbf{N}\mathbf{W}||_F^2$, subject to the constraint that all entries in $\mathbf{W}$ are non-negative.

1. **Computing a Derivative**

   We define the following error function:

   $$E = \frac{1}{DT}||\mathbf{M} - \mathbf{N}\mathbf{W}||_F^2,$$

   where D is the number of dimensions (rows) in $\mathbf{M}$, and T is the number of vectors (frames) in $\mathbf{M}$.

   <u>Derive and write down the formula for $\frac{dE}{d\mathbf{W}}$.</u>

2. **A Non-Negative Projection**

   We define the following gradient descent rule to estimate $\mathbf{W}$. It is an iterative estimate. Let $\mathbf{W}^0$ be the initial estimate of $\mathbf{W}$ and $\mathbf{W}^n$ the estimate after n iterations.
   We use the following project gradient update rule

   $$\hat{\mathbf{W}}^{n+1} = \mathbf{W}^n - \eta \left.\frac{dE}{d\mathbf{W}}\right|_{\mathbf{W}^n}$$

   $$\mathbf{W}^{n+1} = \max(\hat{\mathbf{W}}^{n+1}, 0)$$

   where $\frac{dE}{d\mathbf{W}}|_{\mathbf{W}^n}$ is the derivative of E with respect to $\mathbf{W}$ computed at $\mathbf{W} = \mathbf{W}^n$, and $\max(\hat{\mathbf{W}}^{n+1}, 0)$ is a *component-wise* flooring operation that sets all negative entries in $\hat{\mathbf{W}}^{n+1}$ to 0.

In effect, our *feasible set* for values of $\mathbf{W}$ are $\mathbf{W} \geq 0$, where the symbol $\geq$ indicates that *every* element of $\mathbf{W}$ must be greater than or equal to 0. The algorithm performs a conventional gradient descent update, and projects any solutions that fall outside the feasible set back onto the feasible set, through the max operation.

Implement the above algorithm. Initialize $\mathbf{W}$ to a matrix of all 0s. Run the algorithm for η values $(100, 1000, 10000, 100000)$. Run 1000 iterations in each case. Plot E as a function of iteration number n for all ηs in a figure. Show this plot with some analysis in the separate page, and submit the best final matrix $\mathbf{W}$ (which resulted in the lowest error) named as `problem2W.csv` with the code.

3. **Recreating the music**

   For the best η (which resulted in the lowest error) recreate the music using this transcription as $\hat{\mathbf{M}} = \mathbf{NW}$. Resynthesize the music from $\hat{M}$. What does it sound like? Submit the resynthesized music named as `resynthesized_nnproj.wav` with the code.

4 Style transfer using a Linear Transformation

Here we have three pieces of audio. The first two are `Synth.wav` (audio A) and `Piano.wav` (audio B), which are recordings of a chromatic scale in a single octave played by a synthesizer and a piano respectively. The third piece of audio is the intro melody of "Blinding Lights" (audio C) by The Weeknd, played with the same synth tone used to generate `Synth.wav`.

All audio files are in the `hw1materials/audio` folder.

From these files, you can obtain the spectrogram $\mathbf{M}_A$, $\mathbf{M}_B$ and $\mathbf{M}_C$. Your objective is to find the spectrogram of the piano version of the song "Blinding Lights" ($\mathbf{M}_D$).

To compute the spectrogram from the given files, use the same instructions of the previous problem, but using 1024 as the window length instead of 2048. Keep all other parameters the same.

In this problem, we assume that style can be transferred using a linear transformation. Formally, we need to find the matrix $\mathbf{T}$ such that

$$\mathbf{T}\mathbf{M}_A \approx \mathbf{M}_B$$

1. Write code to determine matrix $\mathbf{T}$ and report the value of $\|\mathbf{T}\mathbf{M}_A - \mathbf{M}_B\|_F^2$.
   Submit the matrix $\mathbf{T}$ as `problem3t.csv` and your code
2. Our model assumes that $\mathbf{T}$ can transfer style from synthesizer music to piano music. Applying $\mathbf{T}$ on $\mathbf{M}_C$ should give us a estimation of "Blinding Lights" played by Piano, getting an estimation of $\mathbf{M}_D$. Using this matrix and phase matrix of C, synthesize an audio signal.
   Submit your code, your estimation of the matrix $\mathbf{M}_D$ as `problem3md.csv` and the sythensized audio named as `problem3.wav`

5 Music Decomposition

5.1 Introduction

One of the most important problems in machine learning, particularly as applied to signal processing, is that of finding the "building blocks" (or "bases") of a given data or data type. The ideal building blocks for any specific form of data are those that are *specific* (i.e. most characteristic) to that data. Knowing these building blocks enables various kinds of analyses and inferences as we will see during the course.

However the notion of a "specific" building block can be vague, and it is generally hard for us to determine if we have succeeded in finding them.

There *is* one class of data, however, where the definition is reasonably clear – music, particularly when it is from a single instrument. Clearly, the building blocks here are the notes. We can evaluate our building-block-extraction algorithms by determining if they have actually extracted the notes.

This homework is the first of a series where we will evaluate various decomposition algorithms for their ability to successfully extract notes as the building blocks of music. Our representation of the audio data is spectrographic – as a spectrogram, which, as you may recall, is essentially just a matrix of numbers. Thus, the decomposition algorithms all take the form of matrix decomposition algorithms of various kinds.

In this homework we test singular value decomposition as a mechanism for deriving meaningful building blocks from a music recording.

5.2 Problem

In the folder `hw1materials/music_decomposition` you'll find the file called `FoxTitle.wav`. Everyone should be familiar with this tune. This is deliberately rearranged much simpler to be played by oboes. The audio is already resampled at 16000 Hz for your convenience.

We know that music is composed by individual notes, and these notes are the natural building blocks that constitute the whole signal. Finding these building blocks is a fundamental problem in signal processing study. In this piece of music, you can hear 11 oboe notes in different pitches. We consider these notes as the basic building blocks of the signal, as the spectrums are steady in the time domain and we can view the signal as the weighted sum of these spectra over time. Different from Question 3, here you'll need to do music decomposition without knowing the spectrums. In other words, you need to determine the spectrums themselves! This might sound pretty tricky now, but don't worry. You'll encounter this problem again in the later homeworks as we learn more techniques. For now let's see how well we can make use of SVD on this task. Let $\mathbf{M}$ be the (magnitude) spectrogram of the music, which is of size $D \times T$, where D is the FFT size and T is the total number of windows in Fourier analysis, which represents the length of music. We can perform SVD on $\mathbf{M}$ that:

$$\mathbf{M} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

Here $\mathbf{U}$ has the size of $D \times D$, $\mathbf{V}$ has the size of $T \times T$, and $\mathbf{S}$ has the size of $D \times T$. We can also rewrite $\mathbf{M}$ as the weighted sum of the product of singular vectors:

$$\mathbf{M} = \sum_i \mathbf{s}_i \mathbf{U}_i \mathbf{V}_i^T$$

Each left singular vector $\mathbf{U}_i$ and the corresponding right singular vector ${\mathbf{V}^\mathbf{T}}_i$ contribute on a basic component of the data. As $\mathbf{U}_i$ is of size $D \times 1$ and ${\mathbf{V}^\mathbf{T}}_i$ is of size $1 \times T$, it's intuitive that $\mathbf{U}_i$ describes the spectrum of one building block of music, and ${\mathbf{V}^\mathbf{T}}_i$ is its modulation curve over time. And the singular value $\mathbf{s}_i$ serves as the scaling factor that measures the magnitude of contribution of this building block.

The goal of this task is to analyze the music and tune your algorithm to restore as many individual notes as possible.

1. Use STFT to analyze the music signal. Please use a window size of 1024 samples and a hop size of 160 samples. You can analyze the signal either in linear scale or in logarithmic scale, and with any other STFT parameters you find that perform the best. Then, decompose the signal with SVD and fetch the 11 most significant bases in **U**.

   Submit the 11 most significant components of matrix **U** as `musicbases.csv`.

2. In the folder `hw1materials/music_decomposition` we also have the ground truths of the individual notes for you to validate your results. Using a windows size of 1024 samples and a hop size of 160 samples, every individual note recording will be segmented to 62 frames. Here we view the part starting from the 21st frame to the 40th frame as the steady part with a consistent spectrum. The average spectrum of these frames is the spectrum of this building block note. Compare the 11 building blocks you've got using SVD to each of the ground truth notes, and calculate the inner products of your bases and the ground truth bases to find the best match. How many unique notes can you actually restore and how close are your answers? Please report the note names of the best match of your 11 bases, and their respective inner product values. Write your thoughts about why SVD is / isn't proper for this task.