Machine Learning for Signal Processing
(ENGR-E 511)
Homework 3

Due date: March 15, 2020, 23:59 PM (Eastern)

Instructions

- No hand-written report
- Option 1: PDF + ZIP
  - A.pdf file generated from LaTeX
  - A.zip file that contains source codes and media files
- Option 2: IPython Notebook + HTML
  - Your notebook should be a comprehensive report, not just a code snippet. Mark-ups are mandatory to answer the homework questions. You need to use LaTeX equations in the markup if you're asked.
  - Download your notebook as an .html version and submit it as well, so that the AIs can check out the plots and audio.
  - Meaning you need to embed an audio player in there if you're asked to submit an audio file
- Avoid using toolboxes.

P1: Instantaneous Source Separation [4 points]

1. As you might have noticed from my long hair, I've got a rock spirit. However, for this homework I dabbled to compose jazz music. The title of the song is boring: `Homework 3`.

2. From `x_ica_1.wav` to `x_ica_20.wav` are 20 recordings of my song, `Homework 3`. Each recording has N time domain samples. In this music there are K unknown number of musical sources played at the same time. In other words, it could simulate the situation that 20 of my students come to my gig and record my band's play from 20 different locations (sounds unethical, so I wouldn't invite you guys, no worries). This can be seen as a situation where the source was mixed up with a $20 \times K$ mixing matrix $\boldsymbol{A}$ to the K sources to create the 20 channel mixture:

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_{20}(t) \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} s_1(t) \\ s_2(t) \\ \vdots \\ s_K(t) \end{bmatrix} \tag{1}$$

3. As you've learned how to do source separation using ICA, you should be able to separate them out into K clean speech sources.

4. First, you don't like the fact that there are too many recordings for this separation problem, because you have a feeling that the number of sources is a lot smaller than 20. So, you decided to do a dimension redcution first, before you actually go ahead and do ICA. For this, you choose to perform PCA with the whitening option. Apply your PCA algorithm on your data matrix $\boldsymbol{X}$, a $20 \times N$ matrix. Don't forget to whiten the data. Make a decision as to how many dimensions to keep, which will correspond to your K. Hint: take a very close look at your eigenvalues.

5. On your whitened/dimension reduced data matrix $\boldsymbol{Z}$ ($K \times N$), apply ICA. At every iteration of the ICA algorithm, use these as your update rules:

$$\begin{aligned} \Delta \boldsymbol{W} &\leftarrow \left(N\boldsymbol{I} - g(\boldsymbol{Y})f(\boldsymbol{Y})^\top\right)\boldsymbol{W} \\ \boldsymbol{W} &\leftarrow \boldsymbol{W} + \rho \Delta \boldsymbol{W} \\ \boldsymbol{Y} &\leftarrow \boldsymbol{W}\boldsymbol{Z} \end{aligned}$$

where

$\boldsymbol{W}$: The ICA unmixing matrix you're estimating
$\boldsymbol{Y}$: The $K \times N$ source matrix you're estimating
$\boldsymbol{Z}$: Whitened/dim reduced version of your input (using PCA)
$g(x)$: $\tanh(x)$
$f(x)$: x^3
ρ : learning rate
N : number of samples

6. Enjoy your separated music. Submit your separated .wav files, source code, and the convergence graph.

7. Implementation notes: Depending on the choice of the learning rate the convergence of the ICA algorithm varies. But I always see the convergence in from 5 sec to 90 sec in my iMac.

P2: Ideal Masks [3 points]

1. `piano.wav` and `ocean.wav` are two sources you're interested in. Load them separately and apply STFT with 1024 point frames and 50% overlap. Use Hann windows. Let's call these two spectrograms $\boldsymbol{S}$ and $\boldsymbol{N}$, respectively. Discard the complex conjugate part, so eventually they will be an 513×158 matrix. Later on in this problem when you recover the time domain signal out of this, you can easily recover the discarded half from the existing half so that you can do inverse-DFT on the column vector of full 1024 points. Hint: Why 513, not 512? Create a very short random signal with 16 samples, and do a DFT transform to convert it into a spectrum of 16 complex values. Check out their complex coefficients to see why you need $N/2 + 1$, not $N/2$[1].

2. Now you build a mixture spectrogram by simply adding the two source spectrograms: $\boldsymbol{X} = \boldsymbol{S} + \boldsymbol{N}$.

[1] I'll allow you to use a toolbox for STFT, but I encourage you to use your own implementation.

3. Since you know the sources, the source separation job is trivial. One way is to calculate the ideal masks $\boldsymbol{M} = \frac{\boldsymbol{S}}{\boldsymbol{S}+\boldsymbol{N}}$ (note that they are all complex valued and the division is element-wise). By the definition of the mixture spectrogram, $\boldsymbol{S} = \boldsymbol{M} \odot \boldsymbol{X}$, where $\odot$ stands for a Hadamard product.

4. Sometimes we can only estimate a nonnegative real-valued masking matrix $\bar{\boldsymbol{M}}$ especially if we don't have access to the phase of the sources. For example, $\bar{\boldsymbol{M}} = \frac{|\boldsymbol{S}|^2}{|\boldsymbol{S}|^2+|\boldsymbol{N}|^2}$. Go ahead and calculate $\bar{\boldsymbol{M}}$ from your sources, and multiply it to your mixture spectrogram, i.e. $\boldsymbol{S} \approx \bar{\boldsymbol{M}} \odot \boldsymbol{X}$. Convert your estimated piano spectrogram back to the time domain. Submit the .wav file of your recovered piano source or embed it in your notebook.

5. Listen to the recovered source. Is it too different from the original? One way to objectively measure the quality of the recovered signal is to compare it to the original signal by using a metric called Signal-to-Noise Ratio (SNR):

$$SNR = 10 \log_{10} \left(\frac{\sum_t \{s(t)\}^2}{\sum_t \{s(t) - \hat{s}(t)\}^2} \right), \tag{2}$$

   where $s(t)$ is the t-th sample of the original source and $\hat{s}(t)$ is that of the recovered one. Evaluate the SNR between `piano.wav` and your reconstruction for it. Note: their lengths could be slightly different. Just ignore the small difference in the end.

6. Yet another masking scheme is something called Ideal Binary Masks (IBM). This time, we use a binary (0 or 1) masking matrix $\boldsymbol{B}$, which is definded by

$$B_{ft} = \begin{cases} 1 & \text{if } S_{ft} > N_{ft} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

7. Create your IBM from the sources, and apply it to your mixture spectrogram, $\boldsymbol{S} \approx \boldsymbol{B} \odot \boldsymbol{X}$. Do the inverse STFT. How does it sound? What's its SNR value?

P3: Single-channel Source Separation [4 points]

1. `trs.wav` is a speech signal of a speaker. Convert this signal into a spectrogram and take its magnitudes. Let's call this magnitude spectrogram $\boldsymbol{S}$. Learn an NMF model out of this such as $\boldsymbol{S} \approx \boldsymbol{W_S H_S}$. You know, $\boldsymbol{W_S}$ is a set of basis vectors. If you discard the complex conjugates, your $\boldsymbol{S}$ is a 513×990 matrix (the number of columns could be slightly different depending on how you implemented STFT). If you choose to learn this NMF model with 30 basis vectors, then $\boldsymbol{W_S} \in \mathbb{R}_+^{513 \times 30}$, where $\mathbb{R}_+$ is a set of nonnegative real numbers. You're going to use $\boldsymbol{W_S}$ for your separation.[2]

2. Learn another NMF model from `trn.wav`, which is another training signal for your noise. From this get $\boldsymbol{W_N}$.

3. `x_nmf.wav` is a noisy speech signal made of the same speaker's different speech and the same type of noise you saw. By using our third NMF model, we're going to denoise this one.

[2] I'll allow you to use a toolbox for STFT, but I encourage you to use your own implementation.

Load this signal and convert it into a spectrogram $\boldsymbol{X} \in \mathbb{C}^{513\times 131}$. Let's call its magnitude spectrogram $\boldsymbol{Y} = |\boldsymbol{X}| \in \mathbb{R}_+^{513\times 131}$. Your third NMF will learn this approximation:

$$\boldsymbol{Y} \approx [\boldsymbol{W_S W_N}]\boldsymbol{H}. \tag{4}$$

What this means is that for this third NMF model, instead of learning new basis vectors, you reuse the ones you trained from the previous two models as your basis vectors for testing: $\boldsymbol{W} = [\boldsymbol{W_S W_N}]$. As you are very sure that the basis vectors for your test signal should be the same with the ones you trained from each of the sources, you initialize your $\boldsymbol{W}$ matrix with the trained ones and don't even update it during this third NMF. Instead, you learn a whole new $\boldsymbol{H} \in \mathbb{R}_+^{60\times 131}$ that tells you the activation of the basis vectors for a given time frame. Implementation is simple. Skip the update for $\boldsymbol{W}$. Update $\boldsymbol{H}$ by using $\boldsymbol{W} = [\boldsymbol{W_S W_N}]$ and $\boldsymbol{Y}$. Repeat.

4. You can think of $\boldsymbol{W_S H}_{(1:30,:)}$ as an estimation of $\boldsymbol{S}$. But, you need its corresponding phase information to convert it back to the time domain. It might not be perfect, but the phase matrix of the input mixture, $\angle \boldsymbol{X} = \frac{\boldsymbol{X}}{\boldsymbol{Y}}$, can be used to recover the complex valued spectrogram of the speech source. Then, you can get the time domain signal. Submit your signal.

5. Because $\boldsymbol{W_S H}_{(1:30,:)}$ can be seen as the speech source estimate, you can also create a magnitude masking matrix out of it:

$$\bar{\boldsymbol{M}} = \frac{\boldsymbol{W_S H}_{(1:30,:)}}{\boldsymbol{W_S H}_{(1:30,:)} + \boldsymbol{W_N H}_{(31:60,:)}} = \frac{\boldsymbol{W_S H}_{(1:30,:)}}{[\boldsymbol{W_S W_N}]\boldsymbol{H}}. \tag{5}$$

Use this masking matrix to recover your speech source. Submit the wav file or embed it to your notebook. Compared to the IRM result, which one do you think will have a larger SNR?

P4: Motor Imagery [4 points]

1. `eeg.mat` has the training and testing samples and their labels. Use them to replicate my classification experiments in Module 5 (not the entire lecture, but from S3 to S8 and S37).

2. But, instead of PCA, use NMF to reduce the dimension. Differently from PCA, where you can keep learning a new eigenvector on top of the previously learned ones, NMF requires you to run its multiplicative update rules with a pre-defined number of basis vectors. Learn 7 different pairs of $\boldsymbol{W}$ and $\boldsymbol{H}$, by varying their rank as in S37: {2, 4, 6, 8, 10, 15, 20}, meaning you need to run the NMF algorithm 7 different times by differently initializing them, and by setting up different number of basis vectors. Note that you should use the magnitudes of the spectrogram as NMF only takes nonnegative input.

3. With eigenvectors, you transposed them and multiplied it to the data matrix from the left-hand side (S8). But, you can't do that anymore, because NMF basis vectors are not orthogonal. Instead, you can just use the $\boldsymbol{H}$ matrix as if it's the dimension-reduced version of your data matrix, $\boldsymbol{Z}$ in S8.

4. Instead of using naïve Bayes classification, do a kNN classification. Report your classification accuracies from various choices of the number of basis vectors and the number of neighbors (I think a table of accuracies will be great). You don't have to submit all the intermediate plots. What I need is the table of accuracies and your source code. Compare your results with mine in S37.