Assignment: Homework 4

What to Hand In

1. Your code with appropriate comments and a **readme file**. You are responsible for making us understand and know how to run your code.
2. A report with answers, figures, and explanations.
3. Any sound files asked for in the problems.

How to Hand It In

1. Put all your solutions **including a report** showing your answers and figures in one folder. Compress this folder and name it <firstname>_<lastname>_HW4.zip. For example, "Zhiyao_Duan_HW4.zip".
2. Submit to the appropriate entry on Blackboard.

When to Hand It In

It is due at 11:59 PM on the date specified on the course calendar. **Late assignments will receive a 20% deduction of the full grade each day.**

Problems (10 points in total)

1. (7 points) Implement the NMF using K-L divergence and the multiplicative rule. Test your NMF implementation on a source separation task.
    a. (1 point) Write a function named "myNMF" to decompose a nonnegative matrix into the multiplication of two nonnegative matrices, using the multiplicative update rule for the K-L divergence cost function. Your function should allow users to choose one matrix (or both) to update. The input/output specification of the function is written in the provided "myNMF.py". If you use MATLAB, please use a similar input/output specification.
    Hint 1: Remember to normalize the W matrix and scale the H matrix accordingly. This is to deal with the non-uniqueness of NMF solutions.

    b. (1 point) Run your function on the **linear-amplitude** magnitude spectrogram of the provided piano recording "piano.wav", with a random initialization. Set r to 4 and the number of iterations to 50. Use hamming window (length=2048 points and hop=512 points) when calculating the spectrogram. Plot the columns of the calculated dictionary matrix W and the rows of the calculated activation matrix H. Annotate the axis of your plots appropriately. Also, plot the original spectrogram and the reconstructed spectrogram. Are they close? Convert the reconstructed spectrogram back to the time domain, using the overlap-add technique, and save it as "piano_recon_r4.wav". Listen to your reconstructed piano sound. Does it sound close to the original?
    Hint 2: When plotting the magnitude spectrogram, always use the log-amplitude. The linear-amplitude is not easy to visualize. But when performing NMF, always use the linear-amplitude, because we are using its (approximately) additive property.

c. (1 point) Re-run your function on the provided piano recording "piano.wav" twice, with r=3 and r=5, respectively. The other settings remain the same. Plot the dictionary, the activation matrix, and the reconstructed spectrogram. Explain the meaning of W and H, i.e., what spectral pattern in V is each column of W describing? Convert the reconstructed spectrogram back to the time domain and save it as "piano_recon_r3.wav" and "piano_recon_r5.wav", respectively. Listen to your reconstructed signals. Do they sound close to the original?

d. (1 point) Run your function on the provided "speech_train.wav" to learn a speech spectral dictionary. Use hamming window (length=1024 points and hop=512 points) when calculating the spectrogram. Choose appropriate parameters such as r and #iterations. Plot several basis spectra in the dictionary.
Hint 3: To choose an appropriate r, you can try different values (e.g., 10, 20, 50, 100, and 200.) to decompose and reconstruct the training signal. Choose the smallest r that you feel the reconstruction is good enough.

e. (1 point) Run your function on the provided "noise_train.wav" to learn a noise spectral dictionary. Use hamming window (length=1024 points and hop=512 points) when calculating the spectrogram. Choose appropriate parameters such as r and #iterations. Plot several basis spectra in the dictionary.

f. (1 point) Use your learned speech and noise dictionaries to separate the provided "noisyspeech.wav". Run your function on the linear-amplitude magnitude spectrogram with **fixed** W, where W = [W_speech, W_noise]. W_speech and W_noise are the speech and noise dictionaries you have learned from the training data. Do not update W during the NMF iterations. Only update the activation matrix H. Reconstruct the speech magnitude spectrogram using W_speech and the corresponding rows of H. Reconstruct the noise magnitude spectrogram using W_noise and the corresponding rows of H. Plot these spectrograms. Convert them back to the time domain and save them as "speech_sep.wav" and "noise_sep.wav", respectively. These are the separated speech and noise signals.
Hint 4: You can use the mixture signal's phase spectrogram when transforming the magnitude spectrogram of separated sources back to the time domain.

g. (1 point) Install BSS_EVAL toolbox with Python/MATLAB, which is a commonly used toolbox to evaluate source separation results. The Python version in the mir_eval package is a reimplementation of the MATLAB version. Use the function "bss_eval_sources" to evaluate your separated speech signal. The original sources are "speech_test.wav" and "noise_test.wav". Report the signal-to-distortion ratio (SDR), signal-to-interference ratio (SIR) and signal-to-artifacts ratio (SAR) values of the separated speech signal. You may want to read Section I-D and Section II-B of [1] to know what this toolbox is trying to evaluate.

2. (3 points) Implement the Viterbi algorithm. Run it to smooth the single pitch detection results. (For MATLAB, all the matrix stored in npy files can be found in "pitchdata.mat" with the same name)
    a. (1 point) Write a function named "myViterbi" to implement the Viterbi algorithm to decode a finite-state hidden Markov model. The specifications of the input and output is described in the provided "myViterbi.py".

    b. (1 point) The provided "female_factory.wav" is a female speech file corrupted by factory noise with SNR=0dB. "loglikeMat.npy" stores the log-likelihood of each pitch hypothesis (state) in each time frame (observation). There are in total 65 pitch hypotheses in each frame (which is why loglikeMat has 65 rows). The first pitch hypothesis is "no pitch". The 2nd to the 65th hypotheses correspond the pitch values between 65Hz and 370Hz, equally spaced in log-frequency. The formula to convert from the index to Hz is **Hz = midi2hz((index-2)*0.5 + 35.25),** i.e., the 2nd pitch hypothesis is actually midi2hz(35.25) Hz. You already know how to convert MIDI number to Hz in HW1.

    Now, given "loglikeMat", one way to estimate the pitch in each frame is to choose the pitch hypothesis that achieves the maximum log-likelihood in that frame. Plot the estimated pitches on top of the magnitude spectrogram in one figure. The pitches should be plotted as dots. You should see many of the estimated pitches are at the first harmonic of the female voice and some are at higher harmonics or at other places. In calculating the spectrogram, use frame length of 1024 points and hop size of 160 points (10ms), hamming window, and 4 times zero padding, because "loglikeMat" was calculated using these parameters. Do you think the pitches estimated in this way are good?

    c. (1 point) Suppose we know how human pitch transitions between time frames, we can use an HMM to get better (smoother) pitch estimates. The "transMat.npy" is the transition probability matrix learned from a lot of files of human speech. The "initProb.npy" is the initial probability of the pitch hypotheses learned from a lot of files of human speech. Both of them have 65 states, corresponding to the 65 pitch hypotheses described above. You can visualize "transMat" and see that it has a strong diagonal, which means human pitch contour tends to be smooth. Now you can use your Viterbi algorithm to find the best path of states (pitch hypotheses across frames) that gives you the highest posterior probability. Run "myViterbi" on this data. Plot the pitch estimates on top of the magnitude spectrogram in one figure. Do you think the pitches estimated after using the HMM model are better than the pitches estimated in b?

References

[1] E. Vincent, R. Gribonval and C. Févotte, **Performance measurement in blind audio source separation**, *IEEE Trans. Audio, Speech and Language Processing*, 14(4), pp 1462-1469, 2006.