Machine Learning for Signal Processing
(ENGR E599)
Homework 5

Due date: Apr. 21, 2017, 23:59 PM

Instructions

- Place the pdf report along with code and any other files in a single zip file
- Name the zip file:

  `<iu_username>_HW<#>_<language>.zip`

- It will greatly help the grading process if you use python 3 instead of python 2. MATLAB and R are fine.
- No handwritten solutions
  - Go to `http://sharelatex.com` ASAP and learn how to use LaTeX
- Start early if you're not familiar with the subject, programming, and LaTeX.
- Do it yourself. Discussion is fine, but code up on your own
- Late policy
  - If the sum of the late hours (throughout the semester) ¡ five days (120 hours): no penalty
  - If your total late hours is between 120:00:01 and 144:00:00 (from 5 to 6 days): you'll lose 20% of your total late homework score you earned
  - Between 144:00:01 and 168:00:00 (from 6 to 7 days): you'll lose 40% of your total late homework score you earned
  - Between 168:00:01 and 192:00:00 (from 7 to 8 days): you'll lose 60% of your total late homework score you earned
  - Between 192:00:01 and 216:00:00 (from 8 to 9 days): you'll lose 80% of your total late homework score you earned
  - If you're late by more than 9 days for any of the assignment or for some of the assignment in total, you get no points for the late submissions
- It's okay to use whatever programming language you prefer, but avoid using toolboxes.

Problem 1: Speech Denoising Using Neural Networks [2 points]

1. You connected to a conference call you don't want to attend. So, instead of paying attention, you decided to keep working on your final project by writing up a lot of code.

2. You became worried about your keyboard typing noise if it could be heard by the other meeting participants. They'll be offended if they know that you're not paying attention. So, you decided to build a simple NN-based speech denoiser that takes a noisy speech spectrum (speech plus typing noise) and then produces a cleaned-up speech spectrum.

3. `trs.wav` and `trn.wav` are the speech and noise signals you are going to use for training the network. Load them. Let's call the variables $\boldsymbol{s}$ and $\boldsymbol{n}$. Add them up. Let's call this noisy signal $\boldsymbol{x}$. They all must be a 456707 dimensional column vector.

4. Transform the three vectors using STFT (frame size 1024, hop size 512, Hann windowing). Then, you can come up with three complex-valued matrices, $\boldsymbol{S}, \boldsymbol{N}, \boldsymbol{X}$, each of which has about 900 spectra. A spectrum should be with 513 Fourier coefficients (after discarding complex conjugate as usual). $|\boldsymbol{X}|$ is your input matrix (its column vector is one input sample).

5. Define an Ideal Binary Mask (IBM) $\boldsymbol{M}$ by comparing $\boldsymbol{S}$ and $\boldsymbol{N}$:

$$M_{f,t} = \begin{cases} 1 & \text{if } |S_{f,t}| > |N_{f,t}| \\ 0 & \text{otherwise} \end{cases}$$

   This is your target variable.

6. Train a neural network with whatever structure you'd like. A baseline could be a shallow neural network with a single hidden layer, which has 50 hidden units. For the hidden layer, you can use tanh (or whatever activation function you prefer, e.g. rectified linear units). But, for the output layer, you have to apply a logistic function to each of your 513 output units rather than any other activation functions, because you want your network output to be ranged between 0 and 1 (remember, you're predicting a binary mask!). Feel free to investigate the other options if you're up to such as deeper structure, dropout, early stopping, or whatever (but you don't have to). Don't worry about GPU computing and everything like that. Your baseline shallow tanh network should work to some degree, and once the performance is above my criterion, you'll get a full score.

7. `tex.wav` and `tes.wav` are the test noisy signal and its corresponding ground truth clean speech. Load them and apply STFT as before. Feed the magnitude spectra of the test mixture $|\boldsymbol{X}_{test}|$ to your network and predict their masks $\boldsymbol{M}_{test}$ (ranged between 0 and 1). Then, you can recover the (complex-valued) speech spectrogram of the test signal in this way: $\boldsymbol{X}_{test} \odot \boldsymbol{M}_{test}$.

8. Recover the time domain speech signal by applying an inverse-STFT on $\boldsymbol{X}_{test} \odot \boldsymbol{M}_{test}$. Let's call this cleaned-up test speech signal $\hat{\boldsymbol{s}}$. From `tes.wav`, you can load the ground truth clean test speech signal $\boldsymbol{s}$. Report their SNR (see Homework #3 Problem 2 if you forgot what SNR is).

9. Note: My shallow network implementation converges in 5000 epoch, which never takes more than 5 minutes using my laptop CPU. Don't bother learning GPU computing for this problem. But, your shallow network should give you about 12 dB SNR.

Problem 2: Stereo Matching (revisited) [2 points]

1. You remember you did the GMM-based stereo matching, right? If you forgot, revisit Homework 2 Problem 2.

2. Extend your implementation with the MRF's smoothing priors using an eight neighborhood system (e.g. $\mathcal{N}_{i,j} = \big\{(i-1,j-1),(i-1,j),(i-1,j+1),(i,j-1),(i,j+1),(i+1,j-1),(i+1,j),(i+1,j+1)\big\}$. Feel free to choose either ICM or Gibbs sampling. Show me the smoothed results. You can use the Gaussian-kernel-looking prior probability equations.

Problem 3: Probabilistic Latent Semantic Indexing (PLSI) for Speech Denoising [2 points]

1. Convert the two training signals `trs.wav` and `trn.wav` using the same setup with P1-4. This time, you don't have to mix the two. You just need $\boldsymbol{S}$ and $\boldsymbol{N}$.

2. Like you did in Homework #3 Problem 3, build a speech denoising system by using the PLSI algorithm, rather than NMF.

3. Report the SNR value of the separation results for the test signal in Problem 1.

4. Note: you'll learn PLSI in the next lecture.

Problem 4: PLSI for Analyzing Twitter Stream [2 points]

1. `twitter.mat` holds two Term-Frequency (TF) matrices $\boldsymbol{X}_{tr}$ and $\boldsymbol{X}_{te}$ that I kindly extracted from a twitter stream for you. It also contains $Y_{tr}Mat$ and $Y_{te}Mat$, the target variables in the one-hot vector format.

2. Each column of the TF matrix $\boldsymbol{X}_{tr}$ can be either "positive", "negative", or "neutral", which are represented numerically as 1, 2, and 3 in the $Y_{tr}Mat$. They are sentimental classes of the original twits.

3. Learn 50 PLSI topics $\boldsymbol{B} \in \mathbb{R}^{891\times 50}$ and their weights $\boldsymbol{\Theta}_{tr} \in \mathbb{R}^{50\times 773}$ from the training data $\boldsymbol{X}_{tr}$, using the ordinary PLSI update rules.

4. Reduce the dimension of $\boldsymbol{X}_{te}$ down to 50, by learning the weight matrix $\boldsymbol{\Theta}_{te} \in \mathbb{R}^{50\times 193}$. This can be done by doing another PLSI on the test data $\boldsymbol{X}_{te}$, but this time by reusing the topic matrix $\boldsymbol{B}$ you learned from the training set. So, you skip the update rule for $\boldsymbol{B}$. You only update $\boldsymbol{\Theta}_{te} \in \mathbb{R}^{50\times 193}$.

5. Define a perceptron layer for the softmax classification. This part is similar to the case with kernel PCA with a perceptron as you did in Homework #4 Problem 3. Instead of the kernel PCA results as the input to the perceptron, you use $\boldsymbol{\Theta}_{tr}$ for training, and $\boldsymbol{\Theta}_{te}$ for testing. This time the number of output units is 3 as there are three classes, and that's why the target variable $Y_{tr}Mat$ is with three elements. Review L8S37-S39 to review what softmax is.

6. Report your classification accuracy.