Machine Learning for Signal Processing
(ENGR-E 511)
Homework 4

Due date: May 3, 2020, 23:59 PM (Eastern)
NO GRACE PERIOD.
THE SUBMISSION SYSTEM WILL BE CLOSED AT THIS HARD DEADLINE.

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: Neural Network for Source Separation [4 points]

1. When you were attending IUB, you took a course taught by Prof. K. Since you really liked his lectures, you decided to record them without the professor's permission. You felt awkward, but you did it anyway because you really wanted to review his lectures later.

2. Although you meant to review the lecture every time, it turned out that you never listened to it. After you graduated, you realized that a lot of concepts you face at work were actually covered by Prof. K's class. So, you decided to revisit the lectures and study the materials once again using the recordings.

3. You should have reviewed your recordings earlier. It turned out that there was a fellow student who used to sit next to you always ate chips in the middle of the class right beside your microphone. So, Prof. K's beautiful deep voice was contaminated by the annoying chip eating noise. So, you decided to build a simple NN-based speech denoiser that takes a noisy speech spectrum (speech plus chip eating noise) and then produces a cleaned-up speech spectrum.

4. `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 403,255 dimensional column vector.

5. 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 800 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).

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

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

   whose column vectors are the target samples.

7. Train 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 training options, such as weight decaying and early stopping (by folding out 10% of spectra for validation). But you don't have to.

8. Your baseline shallow tanh network should work to some degree, and once the performance is above my criterion, you'll get a full score.

9. `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: $\boldsymbol{X}_{test} \odot \boldsymbol{M}_{test}$.

10. 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 Signal-to-Noise Ratio (SNR):

$$\text{SNR} = 10 \log_{10} \frac{\boldsymbol{s}^\top \boldsymbol{s}}{(\boldsymbol{s} - \hat{\boldsymbol{s}})^\top (\boldsymbol{s} - \hat{\boldsymbol{s}})}. \tag{1}$$

11. 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. Your network should give at least 6 dB SNR.

12. **Note: DO NOT use Tensorflow, PyTorch, or any other package that calculates gradients for you. You need to come up with your own backpropagation algorithm. It's okay to use the one you wrote up in the previous homework.**

P2: Stereo Matching (revisited) [4 points]

1. `im0.ppm` (left) and `im8.ppm` (right) are the pictures taken by two different camera positions[1]. If you load the images, they will be a three dimensional array of $381 \times 430 \times 3$, whose third dimension is for the three color channels (RGB). Let's call them X^L and X^R. For the (i,j)-th pixel in the right image, $X^R_{(i,j,:)}$, which is a 3-d vector of RGB intensities, we can scan and find the most similar pixel in the left image at i-th row (using a metric of your choice). For example, I did the search from $X^L_{(i,j,:)}$ to $X^L_{(i,j+39,:)}$, to see which pixel among the 40 are the closest. I record the index-distance of the closest pixel. Let's say that $X^L_{(i,j+19,:)}$ is the most similar one to $X^R_{(i,j,:)}$. Then, the index-distance is 19. I record this index-distance (to the closest pixel in the left image) for all pixels in my right image to create a matrix called "disparity map", $\boldsymbol{D}$, whose (i,j)-th element says the index-distance between the (i,j)-th pixel of the right image and its closest pixel in the left image. For an object in the right image, if its pixels are associated with an object in the left image, but are shifted far away, that means the object is close to the cameras, and vice versa.

2. Calculate the disparity map $\boldsymbol{D}$ from `im0.ppm` and `im8.ppm`, which will be a matrix of 381×390 (since we search within only 40 pixels). Vectorize the disparity matrix and draw a histogram. How many clusters do you see?

3. Write up your own GMM clustering code, and cluster the disparity values in $\boldsymbol{D}$ (you can of course use your own code from the previous homework, but not the ones from the toolboxes). Each value will belong to (only) one of the clusters. The number of clusters says the number of depth levels. If you replace the disparity values with the cluster means, you can recover the depth map with k levels. Plot your depth map (the disparity map replaced by the mean disparities as in the image quantization examples) in gray scale–pixels of the frontal objects should be bright, while the ones in the back get darker.

4. 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 discussed in class (M10 S8).

5. Submit your estimated depth maps from both naïve GMM and MRF.

P3: Rock or Metal [4 points]

1. `trX.mat` contains a matrix of size 2×160. Each of the column vectors holds "loudness" and "noisiness" features that describe a song. If the song is louder and noisier, it belongs to the "metal" class, and vice versa. `trY.mat` holds the labeling information of the songs: -1 for "rock", +1 for "metal".

2. Implement your own AdaBoost training algorithm. Train your model by adding weak learners. For your m-th weak learner, train a perceptron (no hidden layer) with the weighted error function:

$$\mathcal{E}(y_t||\hat{y}_t) = w_t(y_t - \hat{y}_t)^2, \tag{2}$$

[1] http://vision.middlebury.edu/stereo/data/

where w_t is the weight applied to the t-th example after $m-1$-th step. Note that $\hat{y}_t$ is the output of your perceptron, whose activation function is tanh.

3. Implementation note: make sure that the m-th weak learner $\phi_m(\boldsymbol{x})$ is the sign of the perceptron output, i.e. $\text{sgn}(\hat{y}_t)$. What that means is, during training the m-th perceptron, you use $\hat{y}_t$ as the output to calculate backpropagation error, but once the perceptron training is done, $\phi_m(\boldsymbol{x}_t) = \text{sgn}(\hat{y}_t)$, not $\phi_m(\boldsymbol{x}_t) = \hat{y}_t$.

4. Don't worry about testing the model on the test set. Instead, report a figure that shows the final weights over the examples (by changing the size of the markers), as well as the prediction of the models (giving different colors to the area). I'm expecting something similar to the ones in M12 S26.

5. Report your classification accuracy on the training samples, too.

P4: PLSI for Analyzing Twitter Stream [3 points]

1. `twitter.mat` holds two Term-Frequency (TF) matrices $\boldsymbol{X}_{tr}$ and $\boldsymbol{X}_{te}$. 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 M6 S37-39 to review what softmax is.

6. Report your classification accuracy.