Machine Learning for Signals (CS 545)
Homework 4

Due date: Dec. 1, 23:59 PM (Central)

Instructions

- Please don't submit your homework.
- The solution will be distributed on the due date.
- Please still do your homework, as the quizzes and exams will be based on the homework assignments (especially the exams).
- Unless mentioned otherwise, you are expected to implement all the functions instead of using existing toolboxes, e.g., from `sklearn`.
- Avoid using toolboxes.

P1: Neural Networks for the Concentric Dataset

1. Build a neural network that has a single hidden layer for the concentric circles problem. Instead of taking the kernel PCA-processed version of the data, your neural network will take the raw data matrix $\boldsymbol{X}$ as the input. Therefore, instead of a perceptron with 3 input units (plus bias), now your neural network will have 2 input units (plus bias), each of which will take one of the coordinates of a sample.

2. As we won't do any kernel PCA-like feature processing, we will convert these 2D data points into 3D feature vectors, which will correspond to the first layer of your neural network:

$$\boldsymbol{X}_{:,i}^{(2)} = \tanh\left(\boldsymbol{W}^{(1)}\boldsymbol{X}_{:,i}^{(1)} + \boldsymbol{b}^{(1)}\right), \tag{1}$$

   where $\boldsymbol{X}_{:,i}^{(1)} \in \mathbb{R}^{2\times 1}$ is the input to the network (i-th column vector of $\boldsymbol{X}$), while $\boldsymbol{X}_{:,i}^{(2)} \in \mathbb{R}^{3\times 1}$ is the output of the hidden units. What that means is that the weight matrix $\boldsymbol{W}^{(1)} \in \mathbb{R}^{3\times 2}$ and the bias vector $\boldsymbol{b}^{(1)} \in \mathbb{R}^{3\times 1}$. As for the activation functions, you can use other variations, of course, but let's just stick to tanh.

3. In the second, i.e., final, layer, you'll do the usual linear classification on $\boldsymbol{X}_{:,i}^{(2)}$, which will be similar to the perceptron you developed in the Kernel PCA version, except that this time the input to the perceptron is $\boldsymbol{X}_{:,i}^{(2)}$, not the kernel PCA results:

$$y_i = \sigma\left(\left(\boldsymbol{w}^{(2)}\right)^\top \boldsymbol{X}_{:,i}^{(2)} + b^{(2)}\right), \tag{2}$$

   where the activation function σ has to be a logistic sigmoid function as our labels are 0 and 1.

4. Note that you train these two layers altogether. Your backpropagation should work from the final layer back to the first layer. Eventually, you want to estimate these parameters with your backpropagation: $\boldsymbol{W}^{(1)} \in \mathbb{R}^{3\times 2}, \boldsymbol{w}^{(2)} \in \mathbb{R}^{3\times 1}, \boldsymbol{b}^{(1)} \in \mathbb{R}^{3\times 1}, b^{(2)} \in \mathbb{R}^{1}$.

5. Do not use any of the deep learning frameworks, such as Tensorflow or PyTorch, which gets around the manual differentiation. Write up your own backpropagation routine and update algorithms.

6. If you are curious, go ahead and scatter-plot the hidden unit outputs, i.e., the nonlinearly transformed features, to see if they are indeed linearly separable, like I did in L10-S37.

P2: Neural Network for Source Separation

1. When you were attending UIUC, 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 classes. 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 and 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 have 513 Fourier coefficients (after discarding the usual complex conjugate). $|\boldsymbol{X}|$ 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 100 hidden units. It takes one of the $|\boldsymbol{X}|$ matrix's column vector $|\boldsymbol{X}_{:,t}|$ as input and predicts its corresponding $\boldsymbol{M}_{:,t}$. 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 other options, such as early stopping, minibatching, etc, but you don't really have to. Don't worry about GPU computing, either. Your baseline shallow tanh network should work well.

8. `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 $\boldsymbol{M}_{test}$ (ranging between 0 and 1). Then, you can recover the (complex-valued) speech spectrogram of the test signal in this way: $\boldsymbol{X}_{test} \odot \boldsymbol{M}_{test}$.

9. 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):

$$\text{SNR} = 10 \log_{10} \frac{\boldsymbol{s}^\top \boldsymbol{s}}{(\boldsymbol{s} - \hat{\boldsymbol{s}})^\top (\boldsymbol{s} - \hat{\boldsymbol{s}})}. \tag{3}$$

10. Note: My shallow network implementation converges in fewer than 1000 epochs, which never takes more than 5 minutes using Google Colab (with a CPU session). But, no matter how many layers you use, your network should give you at least 13 dB SNR or higher.

11. 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 in the previous homework.

P3: Rock or Metal

1. `trX.npy` 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.npy` 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:

$$\mathcal{E}(y_t || \hat{y}_t) = w_t (y_t - \hat{y}_t)^2, \tag{4}$$

   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 L14 S26.

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

P4: Optimal K for PLSI

1. It is known that there are different ways to express one's feelings using emoticons depending on the culture. For example, in Korea, where I'm from, we use double circumflexes to represent a smiley face, e.g., (^^). On the other hand, an angry face can be represented by (` ´). Note that lips are not necessary.

2. I found, though in the western culture people recognize someone's feeling via the lip shapes, e.g., :) or :(. I have no problem with these emoticons except that they are rotated by 90 degrees (kinda weird to me). After realizing this difference, in my real life, I'm trying to smile by moving my lips instead of using eyes to better communicate my friends from the western culture.

3. `faces.npy` contains eight different human faces, each of which is a vectorized 2D array. If you reshape one of the 441 dimensional vectors into a 21×21 2D array, and then display it (e.g., using the `imshow` function), you will see a picture. Draw all eight faces in this way and include to your solution.

4. While there are eight faces in this dataset, you will see that there are a smaller number of latent variables that are combined to "make up" one's face. For example, I wouldn't say the eyes and nose are one of the effective latent variables, because all have the same eyes and nose in this dataset. To identify those latent variables instead, you may want to figure out the combination of different parts of faces to reconstruct a face and examine all faces. What are the unique components that make up all the human faces effectively? This will define the number of latent variables K.

5. Based on your guess on the number of latent variables K, train a topic model. Don't worry about using LDA, just a PLSI should work well. I recommend the first set of EM equations in L13-S18. PLSI will give you two matrices $\boldsymbol{B} \in \mathbb{R}^{441 \times K}$ and $\boldsymbol{\Theta} \in \mathbb{R}^{K \times 8}$. Since they are from "probabilistic" topic modeling, $\sum_{f=1}^{441} \boldsymbol{B}_{f,k} = 1$ for any choice of k and $\sum_{k=1}^{K} \boldsymbol{\Theta}_{k,t} = 1$ for any choice of t.

6. Draw your K basis images. Reshape each of $\boldsymbol{B}_{:,k}$ back into a 21×21 2D array and show it as an image. Repeat it K times. That K images will show what the underlying face components are to make up the database, if your K is correct.

7. Draw $\boldsymbol{\Theta}$ as an image. Its column vector $\boldsymbol{\Theta}_{:,t}$ will tell you the probability of K basis images as to how much they contribute to reconstruct the t-th face.

8. Draw your reconstructed facial images, i.e., $\boldsymbol{X} \approx \hat{\boldsymbol{X}} = \boldsymbol{B}\boldsymbol{\Theta}$. Again, reshape each of $\hat{\boldsymbol{X}}_{:,t}$ back into a 21×21 2D array and show it as an image. Repeat it 8 times. These eight reconstructed images should be near-perfectly similar to those from $\boldsymbol{X}$.

9. Note that the order of basis images can be shuffled, although it won't affect your solution. The resulting images should be correct to receive full points.