Deep Learning Systems
(ENGR-E 533)
Homework 2

Instructions

Due date: Oct. 17, 2021, 23:59 PM (Eastern)

- Start early if you're not familiar with the subject, TF or PT 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) < seven days (168 hours): no penalty
  - If your total late hours is larger than 168 hours, you'll get only 80% of all the late-submitted homework.
- I ask you to use either PyTorch or Tensorflow 2.x running on Python 3.
- Submit a `.ipynb` as a consolidated version of your report and code snippets. But, the math should be clear with LaTeX symbols and the explanations should be full by using text cells. In addition, submit an `.html` version of your notebook as well, where you embed your sound clips and images. For example, if you have a graph as a result of your code cell, it should be visible in this `.html` version before we run your code. Ditto for the sound examples.

Problem 2: Speech Denoising Using Deep Learning [3 points]

1. *If you took my MLSP class, you may think that you've seen this problem. But, it's actually somewhat different from what you did before, so read carefully. And, this time you SHOULD implement a DNN with at least two hidden layers. So, don't reuse your legacy MATLAB code for this problem.*
2. When you attended 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.
3. Although you meant to review the lecture every time, it turned out that you never listened to it. After graduation, 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.
4. You should have reviewed your recordings earlier. It turned out that 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.

5. But, you vaguly recall that you learned some things about speech denoising and source separation from Prof. K's class. So, you decided to build a simple deep learning-based speech denoiser that takes a noisy speech spectrum (speech plus chip eating noise) and then produces a cleaned-up speech spectrum.

6. Since you don't have Prof. K's clean speech signal, I prepared this male speech data recorded by other people. `train_dirty_male.wav` and

   `train_clean_male.wav` are the noisy speech and its corresponding clean speech you are going to use for training the network. Take a listen to them. Load them and covert them into spectrograms, which are the matrix representation of signals. To do so, you'll need to install `librosa` and use it by using the following codes:

   ```
   !pip install librosa # in colab, you'll need to install this
   import librosa
   s, sr=librosa.load('train_clean_male.wav', sr=None)
   S=librosa.stft(s, n_fft=1024, hop_length=512)
   sn, sr=librosa.load('train_dirty_male.wav', sr=None)
   X=librosa.stft(sn, n_fft=1024, hop_length=512)
   ```

   which is going to give you two matrices $\boldsymbol{S}$ and $\boldsymbol{X}$ of size 513×2459. This procedure is something called Short-Time Fourier Transform.

7. Take their magnitudes by using `np.abs()` or whatever suitable method, because $\boldsymbol{S}$ and $\boldsymbol{X}$ are complex valued. Let's call them $|\boldsymbol{S}|$ and $|\boldsymbol{X}|$.

8. Train a fully-connected deep neural network. A couple of hidden layers would work, but feel free to try out whatever structure, activation function, initialization scheme you'd like. The input to the network is a column vector of $|\boldsymbol{X}|$ (a 513-dim vector) and the target is its corresponding one in $|\boldsymbol{S}|$. You may want to do some mini-batching for this. Make use of whatever functions in Tensorflow or Pytorch.

9. But, remember that your network should predict nonnegative magnitudes as output. Try to use a proper activation function in the last layer to make sure of that. I don't care which activation function you use in the middle layers.

10. `test_01_x.wav` is the test noisy signal. Load them and apply STFT as before. Feed the magnitude spectra of this test mixture $|\boldsymbol{X}_{test}|$ to your network and predict their clean magnitude spectra $|\hat{\boldsymbol{S}}_{test}|$. Then, you can recover the (complex-valued) speech spectrogram of the test signal in this way:

$$\hat{\boldsymbol{S}} = \frac{\boldsymbol{X}_{test}}{|\boldsymbol{X}_{test}|} \odot |\hat{\boldsymbol{S}}_{test}|, \tag{1}$$

    which means you take the phase information of the input noisy signal $\frac{\boldsymbol{X}_{test}}{|\boldsymbol{X}_{test}|}$ and use that to recover the clean speech. $\odot$ stands for the Hadamard product and the division is element-wise, too.

11. Recover the time domain speech signal by applying an inverse-STFT on $\hat{\boldsymbol{S}}_{test}$, which will give you a vector. Let's call this cleaned-up test speech signal $\hat{\boldsymbol{s}}_{test}$. I'll calculate something called

Signal-to-Noise Ratio (SNR) by comparing it with the ground truth speech I didn't share with you. It should be reasonably good. You can actually write it out by using the following code:

```
librosa.output.write_wav('test_s_01_recons.wav', sh_test, sr)
```

or

```
import soundfile as sf
sf.write('test_s_01_recons.wav', sh_test, sr)
```

12. Do the same testing procedure for `test_02_x.wav`, which actually contains Prof. K's voice along with the chip eating noise. Enjoy his enhanced voice using your DNN.

13. Note: You cannot compute SNR without knowing the ground-truth source. So, you may wonder if your result is good enough or not. I deliberately don't share the ground-truth clean source with you, because if so, your model will overfit the test example's performance. There are a couple of ways to evaluate the performance of your system. First, you can just go ahead and listen to the input noisy signal and compare it with the processed version. You know, you can actually go ahead and listen to the result generated from the training examples, too, although they tend to be too good compared to the test examples. Second, you can also set aside a small number of training examples for validation, i.e., you don't include them during training. Once the training process is done, you apply your model to this validation set, and compute the SNR values. That way, you can *simulate* the testing environment, although it doesn't guarantee that the model will work well on the test example, because the validation set can be different from the test set. The second approach is related to the early stopping technique explained in M03 S37 (which you can actually implement if you want). For those who want to try out the second method, SNR is defined as follows:

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

where $s(t)$ and $\hat{s}(t)$ are the ground-truth clean speech and the recovered one in the time domain, respectively. Be careful with the division and logarithm: you don't want your denominator to be zero or anything inside the log function zero. Adding a very small number, e.g., $1e^{-20}$, is a good idea to prevent it.

Problem 2: Speech Denoising Using 1D CNN [3 points]

1. As an audio guy it's sad to admit, but a lot of audio signal processing problems can be solved in the time-frequency domain, or an *image version* of the audio signal. You've learned how to do it in the previous homework by using STFT and its inverse process.

2. What that means is nothing stops you from applying a CNN to the same speech denoising problem. In this question, I'm asking you to implement a 1D CNN that does the speech denoising job in the STFT magnitude domain. 1D CNN here means a variant of CNN which does the convolution operation along only one of the axes. In our case it's the frequency axis.

3. Like you did in homework 1 Q2, install/load `librosa`. Take the magnitude spectrograms of the dirty signal and the clean signal $|\boldsymbol{X}|$ and $|\boldsymbol{S}|$.

4. Both in Tensorflow and PyTorch, you'd better transpose this matrix, so that each row of the matrix is a spectrum. Your 1D CNN will take one of these row vectors as an example, i.e. $|\boldsymbol{X}|^{\top}_{:,i}$. Since this is not an RGB image with three channels, nor you'll use any other information than just the magnitude during training, your input image has only one channel (depth-wise). Coupled with your choice of the minibatch size, the dimensionality of your minibatch would be like this: [(batch size) $\times$ (number of channels) $\times$ (height) $\times$ (width)] = $[B \times 1 \times 1 \times 513]$. Note that depending on the implementation of the 1D CNN layers in TF or PT, it's okay to omit the height information. Carefully read the definition of the function you'll use.

5. You'll also need to define the size of the kernel, which will be $1 \times D$, or simply D depending on the implementation (because we know that there's no convolution along the height axis).

6. If you define K kernels in the first layer, the output feature map's dimension will be $[B \times K \times 1 \times (513 - D + 1)]$. You don't need too many kernels, but feel free to investigate. You don't need too many hidden layers, either.

7. In the end, you know, you have to produce an output matrix of $[B \times 513]$, which are the approximation of the clean magnitude spectra of the batch. It's a dimension hard to match using CNN only, unless you take care of the edges by padding zeros (let's not do zero-padding for this homework). Hence, you may want to flatten the last feature map as a vector, and add a regular linear layer to reduce that dimensionality down to 513.

8. Meanwhile, although this flattening-followed-by-linear-layer approach should work in theory, the dimensionality of your flattened CNN feature map might be too large. To handle this issue, we will used the concept we learned in class, *striding*: usually, a stride larger than 1 can reduce the dimensionality after each CNN layer. You could consider this option in all convolutional layers to reduce the size of the feature maps gradually, so that the input dimensionality of the last fully-connected (FC) layer is manageable. Maxpooling, coupled with the striding technique, would be something to consider.

9. Be very careful about this dimensionality, because you have to define the input and output dimensionality of the FC layer in advance. For example, a stride of 2 pixels will reduce the feature dimension down to roughly 50%, though not exactly if the original dimensionality is an odd number.

10. Don't forget to apply the activation function of your choice, at every layer, especially in the last layer.

11. Try whatever optimization techniques you've learned so far.

12. Check on the quality of the test signals you used in P1. Here, once again, you have no good way to judge the quality of the test results other than listening to them.

Problem 3: Speech Denoising Using 2D CNN [4 points]

1. Now that we know the audio source separation problem can be solved in the image representation, nothing stops us from using 2D CNN for this.

2. To this end, let's define our input "image" properly. You extract an image of 20×513 out of the entire STFT magnitude spectrogram (transposed). That's an input sample. Using this your 2D CNN estimates the cleaned-up spectrum that corresponds to the last (20th) input frame:

$$|\boldsymbol{S}^{\top}_{:,t+19}| \approx \mathcal{F}_{\text{CNN}}\big(|\boldsymbol{X}^{\top}_{:,t:t+19}|\big). \tag{3}$$

   What it means is, the network takes all 20 current and previous input frames $|\boldsymbol{S}^{\top}_{:,t:t+19}|$ into account to predict the clean spectrum of the current frame, $t + 19$.

3. Your next image will be another 20 frames shifted by one frame:

$$|\boldsymbol{S}^{\top}_{:,t+20}| \approx \mathcal{F}_{\text{CNN}}\big(|\boldsymbol{X}^{\top}_{:,t+1:t+20}|\big), \tag{4}$$

   and so on. Therefore, a pair of adjacent images (unless you shuffle the order) will be with 19 overlapped frames. Since your original STFT spectrogram has 2,459 frames, you can create 2,440 such images as your training dataset.

4. Therefore the input to the 2D CNN will be of [(batch size) $\times 1 \times 20 \times 513$].

5. Your 2D CNN should be of course with a kernel whose size along both the width (frequencies) and the height axes (frames) should be larger than 1. Feel free to investigate different sizes.

6. Otherwise, the basic idea must be similar with the 1D CNN case. You'll still need those techniques as well as the FC layer.

7. Report the denoising results in the same way. One thing to note is that your output will be with only 2,440 spectra, and it's lacking the first 19 frames. You can ignore those first few frames when you calculate the SNR of the training results. A better way is to augment your input $\boldsymbol{X}$ with 19 silent frames (some magnitude spectra with very small random numbers) in the beginning to match the dimension. I recommend the latter approach.