Deep Learning Systems (ENGR-E 533) Homework 2

Instructions

Due date: Oct. 20, 2019, 23:59 PM (Eastern)

- Submit your pdf report and a zip file of source codes to Canvas
- No handwritten solutions
  - Go to `http://overleaf.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) $<$ 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 running on Python 3.
- You can submit a `.ipynb` as a consolidated version of report and code if you want. But the math should be clear with LaTeX symbols and the explanations should be full by using text cells. Your sound clips embedded in there should be playable even before running all the codes (otherwise, submit the wavefile separately). Your images in there should be visible without running the code. Don't convert your `.ipynb` into PDF or HTML. We know how to read `.ipynb`, so PDF or HTML export will slow down the grading process and make the AIs grumpy.

Problem 1: Speech Denoising Using 1D CNN [5 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 homework 1 Q2. If you can't believe in your subjective listening test, you could go ahead and calculate the SNR thing by using the following equation, but you might only be able to do it on your training signals:

$$\text{SNR} = 10 \log_{10} \frac{\sum_t \big(s(t)\big)^2}{\sum_t \big(s(t) - \hat{s}(t)\big)^2}. \tag{1}$$

Note that the equation is based on the time-domain signals, the clean speech $s(t)$ and the reconstruction $\hat{s}(t)$. If this SNR value is zero, it means that the recovered signal contains some noise that's as loud as the clean source. Therefore, this number should be larger than 0. Another thing to note is that a too large SNR value calculated from the training signals doesn't always mean a good result on the test signal due to the potential overfitting issue. So, you still need to check the quality of the test signal by listening to it.

Problem 2: Speech Denoising Using 2D CNN [5 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:

$$|\boldsymbol{S}^{\top}_{:,t+19}| \approx \mathcal{F}_{\text{CNN}}\big(|\boldsymbol{X}^{\top}_{:,t:t+19}|\big). \tag{2}$$

   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:

$$|\boldsymbol{S}^{\top}_{:,t+20}| \approx \mathcal{F}_{\text{CNN}}\big(|\boldsymbol{X}^{\top}_{:,t+1:t+20}|\big), \tag{3}$$

   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.