ECE 277/477 Computer Audition
HW5: Singing voice separation with neural networks

Summary:
This homework will guide you through the process of applying neural networks to a traditional signal processing task: **separating singing voice from single-channel mixture**. The basic approach is as follows:

1. Transform the signal into its STFT domain.
2. For the magnitude spectrum $\mathbf{s}_t$ at the t-th frame, we predict a mask $\mathbf{m}_t$ and then calculate the spectrum $\hat{\mathbf{v}}_t$ of the singing voice as

$$\hat{\mathbf{v}}_t = \mathbf{m}_t \odot \mathbf{s}_t, \tag{1}$$

   where $\odot$ represents element-wise multiplication.
3. Reconstruct the estimated singing voice using our estimated magnitude spectrum and the original mixture's phase with inverse STFT and overlap add.

We will write in Python3 with Pytorch. Official Pytorch tutorials are good resources to learn: `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`. Pytorch is more user-friendly than Tensorflow and is well-known for building models with dynamic computation graph.

Deliverables:
1. Codes that can reproduce all your results. It is your responsibility to make all your codes readable. Detailed comments and meaningful variable names are highly recommended.
2. Your saved trained model and the script to run your model on a given audio file.
3. **A report (PDF) that describes your attempts. Your should include what you have done and discussions on cases when your model works and fails, along with possible explanations and potential improvements.** You are free to cite published papers in your report (But it's prohibited to cite their code in your code).
4. Any other files if necessary.

Submission Instruction:
1. Put all your files in one folder.
2. Compress this folder and name it <firstname>_<lastname>_HW5.zip. For example, *Zhiyao_Duan_HW5.zip*
3. Submit via Blackboard to the appropriate entry.

Grading Criteria:
You are allowed to make modifications to the architecture or parameters shown in the following sections for a reason. **Your submission will be graded according to:**
1. Correctness (60%): Is your implementation correct?
2. Performance (20%) : Does it have the expected performance?
3. Report (20%): How do you address the problems you identified?

Dataset:
We use DSD100 [1] in this assignment. The original dataset contains 100 stero tracks sampled at 44.1kHz. In order to reduce the training time, you are going to use a downsampled subset of the original dataset which containts 50 mono tracks sampled at 32kHz. We have provided a pytorch data module for this dataset (*Data.py*).
When we train a model on a dataset, we need to split it in three subsets; a) *Train* b) *Validation* c) *Test*. During training (on Train subset), we validate our model on the *Validation* subset to prevent our model from overfitting the training data. We can also tune the networks parameters, based on the performance of our model on the validation data. Finally, we evaluate our model on the Test subset. It is important the test data is never seen by the network during training.
The zip file only containing reduced downsampled dataset can be found here:
https://tinyurl.com/y5tdljkq

Required libraries
python3, pytorch, torchvision, matplotlib, numpy, scipy, tqdm (for progress bar)
We suggest you to install them all using a package management tool. For example, *anaconda* `https://www.anaconda.com/download` Using *anaconda*, you can easily create virtual environments to manage and isolate your projects.

Some supporting functions
In *util.py*, we provide several function wrappers for the convenience of Matlab users. You do not need to do STFT and ISTFT by hand this time since you have done that many times in your assignments.
In *Data.py*, we provide an implementation of *torchvision* data module for the DSD100 dataset. With this, you do not need to load the whole dataset into the RAM. Audio files will be loaded when needed. Additionally, we provide a *Transforms* class, which is used to perform all the necessary transformations (i.e., waveform to spectrum) and data augmentation. Detailed comments are also provided in the code.
Pytorch's *Dataloader* module uses multiple processes/workers to maintain a queue to prefetch data. You can modify the number of workers in the definition of dataloader (see the code) to much your CPU cores. If you get OOM error, keep decreasing the workers, until you don't

1. The single-frame model (60%)

[*Model.py*] The first model you need to build is a single-frame encoder-decoder architecture for estimating the vocal spectral mask from the mixture (Figure 1). Fill in blanks in *Model.py*. Comments in that file provide additional instructions. Save your best performing model (lowest validation loss) as "*savedModel_feedforward_best.pt*". Train your model. Fill in blanks in "*separate_feedforward.py*" to run your model as a command line tool on a specific audio file. The output of the encoder will be used as the input for the recurrent layer in the next section. You can play with audio files in the test portion of the dataset to see what's happening there.

(**Optional**) Alternatively, you may want to replace the biggest layer with a one-dimensional convolution layer or make both encoder and decoder fully convolutional networks. You can

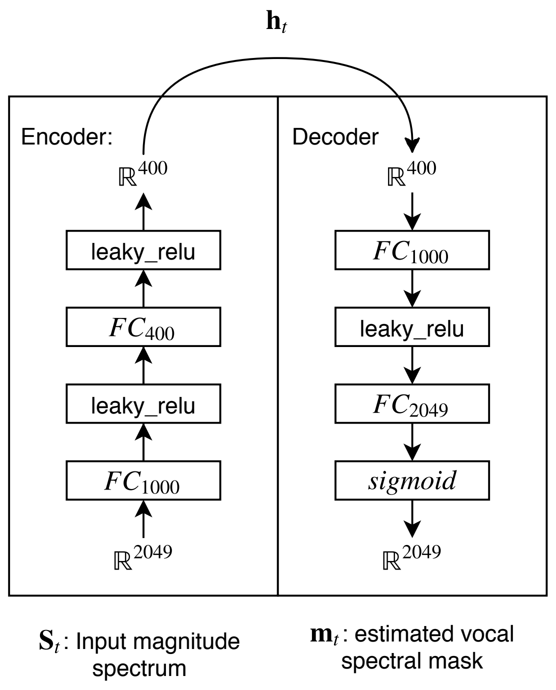

Figure 1: The encoder-decoder architecture for the single frame model. *FC* means the fully-connected layer (*nn.Linear* in Pytorch). Here we use FFT size 4096 and hop size 2048.

do it if you want.

The default loss function for this assignment is the generalized Kullback-Leibler divergence. You have seen this divergence in your previous assignment:

$$\min \mathbb{E}[D(\mathbf{v}_t \| \hat{\mathbf{v}}_t)], \tag{2}$$

where $\hat{\mathbf{v}}_t$ is the estimated spectrum for the frame $\mathbf{s}_t$ and $\mathbf{v}_t$ is the target spectrum of singing voice:

$$\hat{\mathbf{v}}_t = \mathbf{m}_t \odot \mathbf{s}_t.$$

Generalized Kullback-Leibler divergence is given by

$$D(\mathbf{X}\|\mathbf{Y}) = \sum_{ij} x_{ij}(\log(x_{ij} + \epsilon) - \log(y_{ij} + \epsilon)) - x_{ij} + y_{ij}, \tag{3}$$

where ϵ is a small number added for numerical stability. Matrix Notation is used here because you will deal with multiple instances batched together. The final value of the divergence should be averaged over batch size and the number of time steps in order to make the loss for different batch size and different songs comparable.
Note that swapping X and Y in the KL divergence gives a different objective. You can try this if you are interested.
Be patient. You can monitor the training progress by tracking the moving average on the noisy loss value:

$$\text{loss}_{\text{smoothed}} = \eta \text{loss}_{\text{smoothed}} + (1 - \eta)\text{loss}_{current}, \tag{4}$$

where $\eta \in (0, 1)$, usually very close to 1 (e.g., 0.99) The loss curve helps you decide when to stop training. You can use *matplotlib* (pyplot.plot) to do some basic plotting and save the figure to the disk periodically.
A simple way to measure the performance of your model is to calculate the Signal-to-distortion ratio (**SDR**, in decibels) using the ground truth and your estimated output on the **test** portion of the dataset.
You can calculate SDR with *BSS_EVAL 3.0* toolkit you used in the previous assignment (in this single source case, SDR still works).
Please include evaluation results by reporting the average SDR value on the whole test set, to compare all your models.

2. Adding temporal information (40%)

(a) You may have observed that the information contained in a single frame is not enough for many cases. This section will guide to incorporate temporal information.

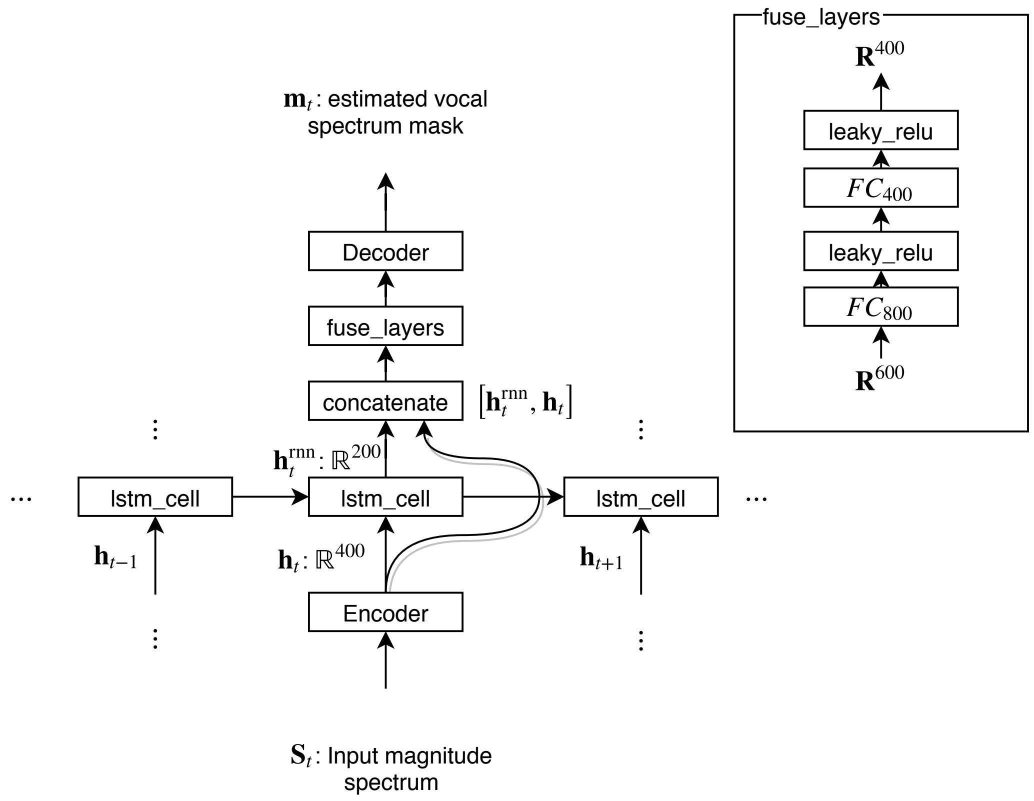
$\mathbf{m}_t$: estimated vocal spectrum mask
Decoder
fuse_layers
concatenate
$[\mathbf{h}_t^{\text{rnn}}, \mathbf{h}_t]$
$\mathbf{h}_t^{\text{rnn}}: \mathbb{R}^{200}$
lstm_cell
lstm_cell
lstm_cell
$\mathbf{h}_{t-1}$
$\mathbf{h}_t: \mathbb{R}^{400}$
$\mathbf{h}_{t+1}$
Encoder
$\mathbf{S}_t$: Input magnitude spectrum
fuse_layers
$\mathbf{R}^{400}$
leaky_relu
FC_{400}
leaky_relu
FC_{800}
$\mathbf{R}^{600}$

Figure 2: An LSTM layer takes the output $\mathbf{h}_t$ of the encoder and outputs $\mathbf{h}_t^{rnn}$ in order to summarize information across frames. $\mathbf{h}_t$ and $\mathbf{h}_t^{rnn}$ are then concatenated and fed into the fuse_layers. The output of the fuse_layers are then fed into the decoder for obtaining the estimated mask.

The architecture is shown in figure 2. We allow $\mathbf{h}_t$ to be able to skip the recurrent layer for two reasons: 1. it can at least degenerate to the previous single-frame model; 2. it allows gradient to take a shorter path.
Note that it is recommended to directly reuse your *Model.py* (import Model) and initialize

weights with the model you have trained to speed up training. This time, do not shuffle frames within an audio file (see comments in *Model.py*) because the ordering of frames does matter.
An audio file can have up to 10k frames. You can either split a whole instance into smaller chunks or do truncated backpropagation through time (TBPTT).
Please name your files *Model_RNN.py*, *savedModel_RNN_best.pt* and *separate_RNN.py* for this modification. **Please also report your evaluation results of this model, with the same way as in part 1**

References

[1] Antoine Liutkus, Fabian-Robert Stöter, Zafar Rafii, Daichi Kitamura, Bertrand Rivet, Nobutaka Ito, Nobutaka Ono, and Julie Fontecave. The 2016 signal separation evaluation campaign. In Petr Tichavský, Massoud Babaie-Zadeh, Olivier J.J. Michel, and Nadège Thirion-Moreau, editors, *Latent Variable Analysis and Signal Separation - 12th International Conference, LVA/ICA 2015, Liberec, Czech Republic, August 25-28, 2015, Proceedings*, pages 323–332, Cham, 2017. Springer International Publishing.