SpeechBrain

SpeechBrain Tutorial

Table of Contents

SpeechBrain

1. Introduction

- SpeechBrain is an easy to use all-in-one speech processing toolkit
- Based on **Pytorch**
- Supports multiple speech-processing tasks like Speech Recognition, Diarization, Verification, Enhancement and Text-to-Speech
- Contains pre-trained models, datasets and easy to customise
- Well-documented and supports multiple functionalities

2. Installation

It can be done in the following ways:

a) PyPI : Use this when you want to directly use any functionality of speechbrain. It can be simply installed and imported using the following commands

```
pip install speechbrain
import speechbrain as sb
```

b) Local : If you want to train your own speech processing system from scratch or make modifications to Speechbrain, you can install it locally after creating the python env

```
git clone https://github.com/speechbrain/speechbrain.git
cd speechbrain
pip install -r requirements.txt
pip install --editable .
```

Note : It is suggested to use different environment for different toolkits, hence it is better to use anaconda to create a new environment *speechbrain* and then perform the local installation. It can be done using :

```
conda create --name speechbrain python=3.9
conda activate speechbrain
```

SpeechBrain

3. Tasks Available

- It can used for a wide-variety of speech tasks such as :
    - Speech Recognition (Multi-Lingual)
    - Speech Separation
    - Speech Enhancement
    - Speaker Verification
    - Speaker Diarization
    - Text-to-Speech
- You can directly load the available pre-trained models and perform inference for the above mentioned tasks
- Fine-Tune the models available on huggingface/speechbrain on your own dataset.

Using pre-trained models

- SpeechBrain contains pre-trained models which can be imported from ***speechbrain.pretrained*** and used directly for inference. You can find the list of pre-trained models here.
- The following is an example of an Automatic Speech Recognition(ASR) task. It also supports multiple languages.

```
from speechbrain.pretrained  import EncoderDecoderASR

# Load the pre-trained model
asr_model = EncoderDecoderASR.from_hparams(source= "speechbrain/asr-crdnn-rnnlm-librispeech" ,
savedir="pretrained_models/asr-crdnn-rnnlm-librispeech" )

asr_model.transcribe_file( 'speechbrain/asr-crdnn-rnnlm-librispeech/example.wav' )

Output : THE BIRCH CANOE SLID ON THE SMOOTH PLANKS
```

SpeechBrain

4. Data Loader

- SpeechBrain Data Loading Pipeline follows the Pytorch Data Loading Pipeline.
- The Pytorch Data Loading Pipeline consists of the following argument :
  - Dataset : It loads one data point at a time
  - Collation Function : This converts the dataset into Pytorch Tensor batches
  - Sampler : decides how the dataset should be iterated
  - Data Loader : This takes the above mentioned and other arguments like batch_size and creates instances of data which are iterated during training
- You can also directly load the data to the brain.fit ( ) function and specify the data loader options (such as batch size) in train_loader_kwargs argument taken from the yaml file. For example :

```
brain.fit(range(hparams["N_epochs"]), data, train_loader_kwargs=hparams["dataloader_options"])
```

5. HyperPyYaml

- HyperPyYaml file is an extension of the Yaml file
- It is used to specify all the hyperparameters (like batch_size,optimizer,learning rate etc) involved in training the model
- Creates a more readable format separating hyperparameters from the model architecture
- Saved as *file_name.yaml* file
- The load_hyperyaml function is used to load the yaml file, and can be used to access the hyperparameters. For example :

```
from hyperpyyaml import load_hyperpyyaml
With open('hyp.yaml') as file:
    hparams = load_hyperpyyaml(file)          # Load the yaml file
brain.fit(range(hparams['n_epochs'],...)      # access no of epochs
```

6. Brain Class

- The Brain class is the most important part of SpeechBrain.
- It is used to perform the training loop (iterate through the dataset and update the model parameters) by using the **fit()** method. It abstracts away the details of the data loops.
- To use the **fit( )** method, the following two methods *need* to be defined :
  - *def compute_forward (self, batch, stage)* : computes the forward pass and generate the model predictions
  - *def compute_objectives (self, predictions, batch, stage)* : contains the loss function used to find the gradient

Note : *stage* is used to track the stages of the experiment. The stage can be defined by the following

`TRAIN= 1 , VALID= 2, TEST= 3`

Brain Class

- In order to define the Brain Class, we require five arguments :
  - *modules* : It takes the model and converts to Torch ModelDict. It makes sure to convert all parameters to same device, for calling train( ) and eval ( )
  - *opt_class* : This argument takes the pytorch optimizer that is to be used. It can be defined in the HyperPyYaml file and can be passed as an argument.
  - *hparams* : This argument takes the set of hyperparameters that need to be defined separately.
  - *run_opts* : This argument handles the execution details of training such as the training device, distributed execution etc.
  - *checkpointer* : This is used to save various details relevant to saving the model like parameters, training progress etc

Example of brain class : *brain = SimpleBrain ({"model": model}, hparams['opt_class'], hparams, run_opts={'device':device},)*

Brain Class

- The ***fit ( )*** method performs the training by taking in arguments such as number of epochs, train data, validation data and parameters related to dataloader such as batch_size. The following is as example of how to call the fit ( ) method

```
brain.fit ( range(hparams["N_epochs"]), data, train_loader_kwargs=hparams["dataloader_options"]  )
```

- The evaluate ( ) method is used to iterate over the testing dataset.
- To **run an experiment** you have to run the following command after activating the speechbrain environment, the train file contains the Brain class, compute_forward, compute_ objectives and the data loader, while the yaml file contains all the hyperparameters required for training the model :

```
python train.py hyp.yaml
```

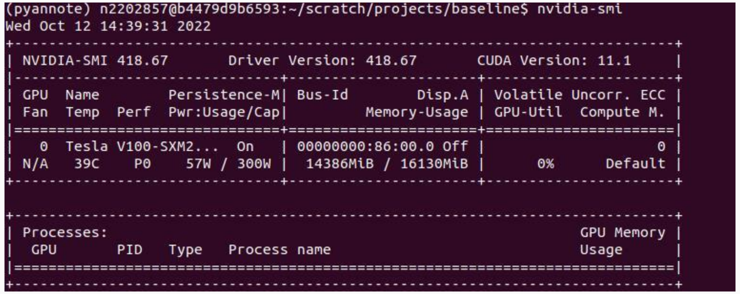
7. Other Useful Links

1) HyperPyYaml Tutorial
2) Checkpointing in SpeechBrain
3) Multi-GPUs
4) Pre-trained Models and Fine-Tuning
5) HyperParameter Optimization

SpeechBrain

8. Some NSCC Tips

- While using the interactive environment on NSCC, you can sometimes get the “CUDA: Out of Memory error”, and the following shows up when you type *nvidia-smi* command



```
(pyannote) n2202857@b4479d9b6593:~/scratch/projects/baseline$ nvidia-smi
Wed Oct 12 14:39:31 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.67       Driver Version: 418.67       CUDA Version: 11.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:86:00.0 Off |                    0 |
| N/A   39C    P0    57W / 300W |  14386MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

- In such a case, just open another terminal and log in to nscc again. Then request for another job on the interactive environment while the previous one is still running. You can run your code here after the new job is scheduled without running into the Out of Memory Error

9. Conclusion

- SpeechBrain is an easy to use, flexible and well-documented speech processing toolkit that can be used for a wide-variety of speech related tasks
- This tutorial consists of the most important steps required to infer from/train models for speech tasks
- SpeechBrain is still in beta version (in progress) project with many upcoming features

10. References

1) https://speechbrain.github.io/
2) **Important Tutorials -** https://speechbrain.github.io/tutorial_basics.html
3) https://arxiv.org/pdf/2106.04624.pdf
4) https://github.com/speechbrain/speechbrain
5) https://speechbrain.readthedocs.io/en/latest/index.html

11. Contribution

- Arabelly Abhinav
- Thanks to Lim Zhi Hao, Liu Chenyu for useful suggestions

END OF PRESENTATION
THANK YOU