Automatic Speech Recognition Examples

Table of Contents

Connectionist Temporal Classification

The script `run_speech_recognition_ctc.py` can be used to fine-tune any pretrained Connectionist Temporal Classification Model for automatic speech recognition on one of the official speech recognition datasets or a custom dataset.

Speech recognition models that have been pretrained in unsupervised fashion on audio data alone, *e.g.* Wav2Vec2, HuBERT, XLSR-Wav2Vec2, have shown to require only very little annotated data to yield good performance on automatic speech recognition datasets.

In the script [`run_speech_recognition_ctc`], we first create a vocabulary from all unique characters of both the training data and evaluation data. Then, we preprocesses the speech recognition dataset, which includes correct resampling, normalization and padding. Finally, the pretrained speech recognition model is fine-tuned on the annotated speech recognition datasets using CTC loss.

NOTE

If you encounter problems with data preprocessing by setting `--preprocessing_num_workers` > 1, you might want to set the environment variable `OMP_NUM_THREADS` to 1 as follows:

```
OMP_NUM_THREADS=1 python run_speech_recognition_ctc ...
```

If the environment variable is not set, the training script might freeze, *i.e.* see: https://github.com/pytorch/audio/issues/1021#issuecomment-726915239

Single GPU CTC

The following command shows how to fine-tune XLSR-Wav2Vec2 on Common Voice using a single GPU in half-precision.

```
python run_speech_recognition_ctc.py \
    --dataset_name="common_voice" \
    --model_name_or_path="facebook/wav2vec2-large-xlsr-53" \
    --dataset_config_name="tr" \
    --output_dir="./wav2vec2-common_voice-tr-demo" \
    --overwrite_output_dir \
    --num_train_epochs="15" \
    --per_device_train_batch_size="16" \
    --gradient_accumulation_steps="2" \
    --learning_rate="3e-4" \
    --warmup_steps="500" \
    --evaluation_strategy="steps" \
    --text_column_name="sentence" \
    --length_column_name="input_length" \
    --save_steps="400" \
    --eval_steps="100" \
    --layerdrop="0.0" \
    --save_total_limit="3" \
    --freeze_feature_encoder \
    --gradient_checkpointing \
    --chars_to_ignore , ? . ! - \; \: \" “ % ‘ ” � \
    --fp16 \
    --group_by_length \
    --push_to_hub \
    --do_train --do_eval
```

On a single V100 GPU, this script should run in *ca.* 1 hour 20 minutes and yield a CTC loss of **0.39** and word error rate of **0.35**.

Multi GPU CTC

The following command shows how to fine-tune XLSR-Wav2Vec2 on Common Voice using 8 GPUs in half-precision.

```
python -m torch.distributed.launch \
    --nproc_per_node 8 run_speech_recognition_ctc.py \
    --dataset_name="common_voice" \
    --model_name_or_path="facebook/wav2vec2-large-xlsr-53" \
    --dataset_config_name="tr" \
    --output_dir="./wav2vec2-common_voice-tr-demo-dist" \
    --overwrite_output_dir \
    --num_train_epochs="15" \
    --per_device_train_batch_size="4" \
    --learning_rate="3e-4" \
    --warmup_steps="500" \
    --evaluation_strategy="steps" \
    --text_column_name="sentence" \
    --length_column_name="input_length" \
    --save_steps="400" \
    --eval_steps="100" \
    --logging_steps="1" \
    --layerdrop="0.0" \
    --save_total_limit="3" \
    --freeze_feature_encoder \
    --gradient_checkpointing \
    --chars_to_ignore , ? . ! - \; \: \" “ % ‘ ” � \
    --fp16 \
    --group_by_length \
    --push_to_hub \
    --do_train --do_eval
```

On 8 V100 GPUs, this script should run in *ca.* 18 minutes and yield a CTC loss of **0.39** and word error rate of **0.36**.

Multi GPU CTC with Dataset Streaming

The following command shows how to use Dataset Streaming mode to fine-tune XLS-R on Common Voice using 4 GPUs in half-precision.

Streaming mode imposes several constraints on training:

1. We need to construct a tokenizer beforehand and define it via `--tokenizer_name_or_path`.
2. `--num_train_epochs` has to be replaced by `--max_steps`. Similarly, all other epoch-based arguments have to be replaced by step-based ones.
3. Full dataset shuffling on each epoch is not possible, since we don't have the whole dataset available at once. However, the `--shuffle_buffer_size` argument controls how many examples we can pre-download before shuffling them.

```
**python -m torch.distributed.launch \
    --nproc_per_node 4 run_speech_recognition_ctc_streaming.py \
    --dataset_name="common_voice" \
    --model_name_or_path="facebook/wav2vec2-xls-r-300m" \
    --tokenizer_name_or_path="anton-l/wav2vec2-tokenizer-turkish" \
    --dataset_config_name="tr" \
    --train_split_name="train+validation" \
    --eval_split_name="test" \
    --output_dir="wav2vec2-xls-r-common_voice-tr-ft" \
    --overwrite_output_dir \
    --max_steps="5000" \
    --per_device_train_batch_size="8" \
    --gradient_accumulation_steps="2" \
    --learning_rate="5e-4" \
    --warmup_steps="500" \
    --evaluation_strategy="steps" \
    --text_column_name="sentence" \
    --save_steps="500" \
    --eval_steps="500" \
    --logging_steps="1" \
    --layerdrop="0.0" \
    --eval_metrics wer cer \
    --save_total_limit="1" \
    --mask_time_prob="0.3" \
    --mask_time_length="10" \
    --mask_feature_prob="0.1" \
    --mask_feature_length="64" \
    --freeze_feature_encoder \
    --chars_to_ignore , ? . ! - \; \: \" “ % ‘ ” � \
    --max_duration_in_seconds="20" \
    --shuffle_buffer_size="500" \
    --fp16 \
    --push_to_hub \
    --do_train --do_eval \
    --gradient_checkpointing**
```

On 4 V100 GPUs, this script should run in *ca.* 3h 31min and yield a CTC loss of **0.35** and word error rate of **0.29**.

Examples CTC

The following tables present a couple of example runs on the most popular speech-recognition datasets. The presented performances are by no means optimal as no hyper-parameter tuning was done. Nevertheless, they can serve as a baseline to improve upon.

TIMIT CTC

- TIMIT

Dataset	Dataset Config	Pretrained Model	Word error rate on eval	Phoneme error rate on eval	GPU setup	Training time	Fine-tuned Model & Logs	Comm to reprod
TIMIT	-	wav2vec2-base	0.21	-	1 GPU TITAN RTX	32min	here	run.sh
TIMIT	-	wav2vec2-base	0.21	-	1 GPU TITAN RTX	32min	here	run.sh
TIMIT	-	unispeech-large-1500h-cv	0.22	-	1 GPU TITAN RTX	35min	here	run.sh
TIMIT	-	asapp/sew-mid-100k	0.30	-	1 GPU TITAN RTX	28min	here	run.sh
TIMIT	-	ntu-spml/distilhubert	0.68	-	1 GPU TITAN RTX	26min	here	run.sh

Librispeech CTC

- Librispeech

Dataset	Dataset Config	Pretrained Model	Word error rate on eval	Phoneme error rate on eval	GPU setup	Training time	Fine-tuned Model & Logs
Librispeech	"clean" - "train.100"	microsoft/wavlm-large	0.049	-	8 GPU V100	1h30min	here
Librispeech	"clean" - "train.100"	microsoft/wavlm-base-plus	0.068	-	8 GPU V100	1h30min	here
Librispeech	"clean" - "train.100"	facebook/wav2vec2-large-lv60	0.042	-	8 GPU V100	1h30min	here
Librispeech	"clean" - "train.100"	facebook/wav2vec2-large-lv60	0.042	-	8 GPU V100	1h30min	here
Librispeech	"clean" - "train.100"	facebook/hubert-large-ll60k	0.088	-	8 GPU V100	1h30min	here
Librispeech	"clean" - "train.100"	asapp/sew-mid-100k	0.167		8 GPU V100	54min	here

Common Voice CTC

- Common Voice

Dataset	Dataset Config	Pretrained Model	Word error rate on eval	Phoneme error rate on eval	GPU setup	Training time	Fine-tuned Model & Logs
Common Voice	"tr"	facebook/wav2vec2-large-xls-r-300m	-	0.099	8 GPU V100	23min	here
Common Voice	"it"	facebook/wav2vec2-large-xls-r-300m	-	0.077	8 GPU V100	23min	here
Common Voice	"sv-SE"	facebook/wav2vec2-large-xls-r-300m	-	0.099	8 GPU V100	23min	here
Common Voice	"tr"	facebook/wav2vec2-large-xlsr-53	0.36	-	8 GPU V100	18min	here

Common Voice	"tr"	facebook/wav2vec2-large-xlsr-53	0.31	-	8 GPU V100	1h05	here
Common Voice	"tr"	facebook/wav2vec2-large-xlsr-53	0.35	-	1 GPU V100	1h20min	here
Common Voice	"tr"	facebook/wav2vec2-xls-r-300m	0.31	-	8 GPU V100	1h05	here
Common Voice	"tr"	facebook/wav2vec2-xls-r-1b	0.21	-	2 GPU Titan 24 GB RAM	15h10	here
Common Voice	"tr" in streaming mode	facebook/wav2vec2-xls-r-300m	0.29	-	4 GPU V100	3h31	here

Multilingual Librispeech CTC

- Multilingual Librispeech

Dataset	Dataset Config	Pretrained Model	Word error rate on eval	Phoneme error rate on eval	GPU setup	Training time	Fine-tuned Model & Logs
Multilingual Librispeech	"german"	facebook/wav2vec2-large-xlsr-53	0.13	-	1 GPU Titan 24 GB RAM	15h04	here
Multilingual Librispeech	"german"	facebook/wav2vec2-xls-r-300m	0.15	-	1 GPU Titan 24 GB RAM	15h04	here

Sequence to Sequence

The script `run_speech_recognition_seq2seq.py` can be used to fine-tune any Speech Sequence-to-Sequence Model for automatic speech recognition on one of the official speech recognition datasets or a custom dataset.

A very common use case is to leverage a pretrained speech encoding model, *e.g.* Wav2Vec2, HuBERT, XLSR-Wav2Vec2 with a pretrained text decoding model, *e.g.* Bart to create a SpeechEnocderDecoderModel. Consequently, the warm-started Speech-Encoder-Decoder model can be fine-tuned in this script.

As an example, let's instantiate a *Wav2Vec2-2-Bart* model with the `SpeechEnocderDecoderModel` framework:

First create an empty repo on `hf.co` :

```
huggingface-cli repo create wav2vec2-2-bart-base
git clone https://huggingface.co/<your-user-name>/wav2vec2-2-bart-base
cd wav2vec2-2-bart-base
```

Next, run the following script **inside** the just cloned repo:

```
from transformers import SpeechEncoderDecoderModel, AutoFeatureExtractor,
AutoTokenizer, Wav2Vec2Processor

# checkpoints to leverage
encoder_id = "facebook/wav2vec2-base"
decoder_id = "facebook/bart-base"

# load and save speech-encoder-decoder model
# set some hyper-parameters for training and evaluation
model = SpeechEncoderDecoderModel.from_encoder_decoder_pretrained(encoder_id,
decoder_id, encoder_add_adapter=True, encoder_feat_proj_dropout=0.0,
encoder_layerdrop=0.0, max_length=200, num_beams=5)
model.config.decoder_start_token_id = model.decoder.config.bos_token_id
model.config.pad_token_id = model.decoder.config.pad_token_id
model.config.eos_token_id = model.decoder.config.eos_token_id
model.save_pretrained("./")

# load and save processor
feature_extractor = AutoFeatureExtractor.from_pretrained(encoder_id)
tokenizer = AutoTokenizer.from_pretrained(decoder_id)
processor = Wav2Vec2Processor(feature_extractor, tokenizer)
processor.save_pretrained("./")
```

Finally, we can upload all files:

```
git lfs install
git add . && git commit -m "upload model files" && git push
```

and link the official `run_speech_recognition_seq2seq.py` script to the folder:

```
ln -s $(realpath <path/to/transformers>/examples/pytorch/speech-
recognition/run_speech_recognition_seq2seq.py) ./
```

Note that we have added a randomly initialized adapter to `wav2vec2-base` with `encoder_add_adapter=True` which further samples the output sequence of `wav2vec2-base` along the time dimension. The reason is that by default a single output vector of `wav2vec2-base` has a receptive field of *ca.*

25ms (*cf.* with section *4.2* of the official Wav2Vec2 paper), which represents a little less a single character. BART on the other hand makes use of a sentence-piece tokenizer as an input processor so that a single hidden vector of `bart-base` represents *ca.* 4 characters. To better align the output of *Wav2Vec2* and *BART*'s hidden vectors for the cross-attention mechanism, we further subsample *Wav2Vec2*'s output by a factor of 8 by adding a convolution-based adapter.

Having warm-started the speech-encoder-decoder model `<your-user-name>/wav2vec2-2-bart`, we can now fine-tune it on speech recognition.

In the script [`run_speech_recognition_seq2seq`], we load the warm-started model, the feature extractor, and the tokenizer, process a speech recognition dataset, and then make use of the `Seq2SeqTrainer`. Note that it is important to also align the decoder's vocabulary with the speech transcriptions of the dataset. *E.g.* the `Librispeech` has only captilized letters in the transcriptions, whereas BART was pretrained mostly on normalized text. Thus it is recommended to add `--do_lower_case` to the fine-tuning script when using a warm-started `SpeechEncoderDecoderModel`. The model is fine-tuned on the standard cross-entropy language modeling loss for sequence-to-sequence (just like *T5* or *BART* in natural language processing).

NOTE

If you encounter problems with data preprocessing by setting `--preprocessing_num_workers` > 1, you might want to set the environment variable `OMP_NUM_THREADS` to 1 as follows:

```
OMP_NUM_THREADS=1 python run_speech_recognition_ctc ...
```

If the environment variable is not set, the training script might freeze, *i.e.* see: https://github.com/pytorch/audio/issues/1021#issuecomment-726915239

Single GPU Seq2Seq

The following command shows how to fine-tune XLSR-Wav2Vec2 on Common Voice using a single GPU in half-precision.

```
python run_speech_recognition_seq2seq.py \
     --nproc_per_node 8 run_speech_recognition_seq2seq.py \
    --dataset_name="librispeech_asr" \
    --model_name_or_path="./" \
    --dataset_config_name="clean" \
    --train_split_name="train.100" \
    --eval_split_name="validation" \
    --output_dir="./" \
    --preprocessing_num_workers="16" \
    --length_column_name="input_length" \
    --overwrite_output_dir \
    --num_train_epochs="5" \
    --per_device_train_batch_size="8" \
    --per_device_eval_batch_size="8" \
    --gradient_accumulation_steps="8" \
    --learning_rate="3e-4" \
    --warmup_steps="400" \
    --evaluation_strategy="steps" \
    --text_column_name="text" \
    --save_steps="400" \
    --eval_steps="400" \
    --logging_steps="10" \
    --save_total_limit="1" \
    --freeze_feature_encoder \
    --gradient_checkpointing \
    --fp16 \
    --group_by_length \
    --predict_with_generate \
    --generation_max_length="40" \
    --generation_num_beams="1" \
    --do_train --do_eval \
    --do_lower_case
```

On a single V100 GPU, this script should run in *ca.* 5 hours and yield a cross-entropy loss of **0.405** and word error rate of **0.0728**.

Multi GPU Seq2Seq

The following command shows how to fine-tune [XLSR-Wav2Vec2]() on [Common Voice]() using 8 GPUs in half-precision.

```
python -m torch.distributed.launch \
     --nproc_per_node 8 run_speech_recognition_seq2seq.py \
    --dataset_name="librispeech_asr" \
    --model_name_or_path="./" \
    --dataset_config_name="clean" \
    --train_split_name="train.100" \
    --eval_split_name="validation" \
    --output_dir="./" \
    --preprocessing_num_workers="16" \
    --length_column_name="input_length" \
    --overwrite_output_dir \
    --num_train_epochs="5" \
    --per_device_train_batch_size="8" \
    --per_device_eval_batch_size="8" \
    --gradient_accumulation_steps="1" \
    --learning_rate="3e-4" \
    --warmup_steps="400" \
    --evaluation_strategy="steps" \
    --text_column_name="text" \
    --save_steps="400" \
    --eval_steps="400" \
    --logging_steps="10" \
    --save_total_limit="1" \
    --freeze_feature_encoder \
    --gradient_checkpointing \
    --fp16 \
    --group_by_length \
    --predict_with_generate \
    --do_train --do_eval \
    --do_lower_case
```

On 8 V100 GPUs, this script should run in *ca.* 45 minutes and yield a cross-entropy loss of **0.405** and word error rate of **0.0728**

Examples Seq2Seq

Librispeech Seq2Seq

- Librispeech

Dataset	Dataset Config	Pretrained Model	Word error rate on eval	Phoneme error rate on eval	GPU setup	Training time	Fine-tunec Mode & Logs
Librispeech	`"clean"` - `"train.100"`	facebook/wav2vec2-base and facebook/bart-base	0.0728	-	8 GPU V100	45min	here
Librispeech	`"clean"` - `"train.100"`	facebook/wav2vec2-large-lv60 and facebook/bart-large	0.0486	-	8 GPU V100	1h20min	here