NOTE: This example is outdated and is not longer actively maintained. Please follow the new instructions of fine-tuning Wav2Vec2 here

Fine-tuning Wav2Vec2

The `run_asr.py` script allows one to fine-tune pretrained Wav2Vec2 models that can be found here.

This finetuning script can also be run as a google colab TODO: here.

Fine-Tuning with TIMIT

Let's take a look at the script used to fine-tune wav2vec2-base with the TIMIT dataset:

```
#!/usr/bin/env bash
python run_asr.py \
--output_dir="./wav2vec2-base-timit-asr" \
--num_train_epochs="30" \
--per_device_train_batch_size="20" \
--per_device_eval_batch_size="20" \
--evaluation_strategy="steps" \
--save_steps="500" \
--eval_steps="100" \
--logging_steps="50" \
--learning_rate="5e-4" \
--warmup_steps="3000" \
--model_name_or_path="facebook/wav2vec2-base" \
--fp16 \
--dataset_name="timit_asr" \
--train_split_name="train" \
--validation_split_name="test" \
--orthography="timit" \
--preprocessing_num_workers="$(nproc)" \
--group_by_length \
--freeze_feature_extractor \
--verbose_logging \
```

The resulting model and inference examples can be found here. Some of the arguments above may look unfamiliar, let's break down what's going on:

`--orthography="timit"` applies certain text preprocessing rules, for tokenization and normalization, to clean up the dataset. In this case, we use the following instance of `Orthography`:

```
Orthography(
    do_lower_case=True,
    # break compounds like "quarter-century-old" and replace pauses "--"
    translation_table=str.maketrans({"-": " "}),
)
```

The instance above is used as follows:

- creates a tokenizer with `do_lower_case=True` (ignores casing for input and lowercases output when decoding)
- replaces `"-"` with `" "` to break compounds like `"quarter-century-old"` and to clean up suspended hyphens
- cleans up consecutive whitespaces (replaces them with a single space: `" "`)
- removes characters not in vocabulary (lacking respective sound units)

`--verbose_logging` logs text preprocessing updates and when evaluating, using the validation split every `eval_steps`, logs references and predictions.

Fine-Tuning with Arabic Speech Corpus

Other datasets, like the Arabic Speech Corpus dataset, require more work! Let's take a look at the script used to fine-tune wav2vec2-large-xlsr-53:

```
#!/usr/bin/env bash
python run_asr.py \
--output_dir="./wav2vec2-large-xlsr-53-arabic-speech-corpus" \
--num_train_epochs="50" \
--per_device_train_batch_size="1" \
--per_device_eval_batch_size="1" \
--gradient_accumulation_steps="8" \
--evaluation_strategy="steps" \
--save_steps="500" \
--eval_steps="100" \
--logging_steps="50" \
--learning_rate="5e-4" \
--warmup_steps="3000" \
--model_name_or_path="elgeish/wav2vec2-large-xlsr-53-arabic" \
--fp16 \
--dataset_name="arabic_speech_corpus" \
--train_split_name="train" \
--validation_split_name="test" \
--max_duration_in_seconds="15" \
--orthography="buckwalter" \
--preprocessing_num_workers="$(nproc)" \
--group_by_length \
--freeze_feature_extractor \
--target_feature_extractor_sampling_rate \
--verbose_logging \
```

First, let's understand how this dataset represents Arabic text; it uses a format called Buckwalter transliteration. We use the lang-trans package to convert back to Arabic when logging. The Buckwalter format only includes ASCII characters, some of which are non-alpha (e.g., `">"` maps to `"أ"`).

`--orthography="buckwalter"` applies certain text preprocessing rules, for tokenization and normalization, to clean up the dataset. In this case, we use the following instance of `Orthography`:

```
Orthography(
    vocab_file=pathlib.Path(__file__).parent.joinpath("vocab/buckwalter.json"),
    word_delimiter_token="/",  # "|" is Arabic letter alef with madda above
```

```
    words_to_remove={"sil"},  # fixing "sil" in arabic_speech_corpus dataset
    untransliterator=arabic.buckwalter.untransliterate,
    translation_table=str.maketrans(translation_table = {
        "-": " ",  # sometimes used to represent pauses
        "^": "v",  # fixing "tha" in arabic_speech_corpus dataset
    }),
)
```

The instance above is used as follows:

- creates a tokenizer with Buckwalter vocabulary and `word_delimiter_token="/"`
- replaces `"-"` with `" "` to clean up hyphens and fixes the orthography for `"ث"`
- removes words used as indicators (in this case, `"sil"` is used for silence)
- cleans up consecutive whitespaces (replaces them with a single space: `" "`)
- removes characters not in vocabulary (lacking respective sound units)

`--verbose_logging` logs text preprocessing updates and when evaluating, using the validation split every `eval_steps`, logs references and predictions. Using the Buckwalter format, text is also logged in Arabic abjad.

`--target_feature_extractor_sampling_rate` resamples audio to target feature extractor's sampling rate (16kHz).

`--max_duration_in_seconds="15"` filters out examples whose audio is longer than the specified limit, which helps with capping GPU memory usage.

DeepSpeed Integration

To learn how to deploy Deepspeed Integration please refer to this guide.

But to get started quickly all you need is to install:

```
pip install deepspeed
```

and then use the default configuration files in this directory:

- `ds_config_wav2vec2_zero2.json`
- `ds_config_wav2vec2_zero3.json`

Here are examples of how you can use DeepSpeed:

(edit the value for `--num_gpus` to match the number of GPUs you have)

ZeRO-2:

```
PYTHONPATH=../../../src deepspeed --num_gpus 2 \
run_asr.py \
--output_dir=output_dir --num_train_epochs=2 --per_device_train_batch_size=2 \
--per_device_eval_batch_size=2 --evaluation_strategy=steps --save_steps=500 --
eval_steps=100 \
--logging_steps=5 --learning_rate=5e-4 --warmup_steps=3000 \
--model_name_or_path=patrickvonplaten/wav2vec2_tiny_random_robust \
--dataset_name=hf-internal-testing/librispeech_asr_dummy --dataset_config_name=clean \
--train_split_name=validation --validation_split_name=validation --orthography=timit \
--preprocessing_num_workers=1 --group_by_length --freeze_feature_extractor --
```

```
verbose_logging \
--deepspeed ds_config_wav2vec2_zero2.json
```

For ZeRO-2 with more than 1 gpu you need to use (which is already in the example configuration file):

```
    "zero_optimization": {
        ...
        "find_unused_parameters": true,
        ...
    }
```

ZeRO-3:

```
PYTHONPATH=../../../src deepspeed --num_gpus 2 \
run_asr.py \
--output_dir=output_dir --num_train_epochs=2 --per_device_train_batch_size=2 \
--per_device_eval_batch_size=2 --evaluation_strategy=steps --save_steps=500 --
eval_steps=100 \
--logging_steps=5 --learning_rate=5e-4 --warmup_steps=3000 \
--model_name_or_path=patrickvonplaten/wav2vec2_tiny_random_robust \
--dataset_name=hf-internal-testing/librispeech_asr_dummy --dataset_config_name=clean \
--train_split_name=validation --validation_split_name=validation --orthography=timit \
--preprocessing_num_workers=1 --group_by_length --freeze_feature_extractor --
verbose_logging \
--deepspeed ds_config_wav2vec2_zero3.json
```

Pretraining Wav2Vec2

The `run_pretrain.py` script allows one to pretrain a Wav2Vec2 model from scratch using Wav2Vec2's contrastive loss objective (see official paper for more information). It is recommended to pre-train Wav2Vec2 with Trainer + Deepspeed (please refer to this guide for more information).

Here is an example of how you can use DeepSpeed ZeRO-2 to pretrain a small Wav2Vec2 model:

```
PYTHONPATH=../../../src deepspeed --num_gpus 4 run_pretrain.py \
--output_dir="./wav2vec2-base-libri-100h" \
--num_train_epochs="3" \
--per_device_train_batch_size="32" \
--per_device_eval_batch_size="32" \
--gradient_accumulation_steps="2" \
--save_total_limit="3" \
--save_steps="500" \
--logging_steps="10" \
--learning_rate="5e-4" \
--weight_decay="0.01" \
--warmup_steps="3000" \
--model_name_or_path="patrickvonplaten/wav2vec2-base-libri-100h" \
--dataset_name="librispeech_asr" \
--dataset_config_name="clean" \
--train_split_name="train.100" \
--preprocessing_num_workers="4" \
--max_duration_in_seconds="10.0" \
```

```
--group_by_length \
--verbose_logging \
--fp16 \
--deepspeed ds_config_wav2vec2_zero2.json \
```