Trial 2

```
import os
import re
import json
import random
import string
from dataclasses import dataclass
from typing import Dict, List, Union, Optional

import torch
import torchaudio
import librosa
import evaluate
from datasets import load_dataset, Audio, DatasetDict
from transformers import (
    Wav2Vec2BertForCTC,
    Wav2Vec2CTCTokenizer,
    Wav2Vec2FeatureExtractor,
    Wav2Vec2BertProcessor,
    TrainingArguments,
    Trainer,
    set_seed,
)

print("Torch:", torch.__version__)
print("CUDA available:", torch.cuda.is_available())
device = "cuda" if torch.cuda.is_available() else "cpu"
set_seed(42)
```

```
Torch: 2.8.0+cu126
CUDA available: True
```

```
print(device)
```

```
cuda
```

```
import random

def display10(dataset):
    for i in range(10):
        r = random.randint(0,len(dataset))
        print(i+1 , dataset[r]['sentence'])
```

```
# Dataset and language
CV_VERSION = "mozilla-foundation/common_voice_16_0"
LANG_ID = "hi"  # Hindi

# Base SSL model (wav2vec2-bert encoder)
BASE_MODEL = "facebook/w2v-bert-2.0"

# Audio parameters
TARGET_SAMPLING_RATE = 16000

# Training output dir
OUTPUT_DIR = "w2vbert-hi-ctc-cv16"

# Training hyperparameters (tune for your budget)
BATCH_SIZE          = 1      # 1 is the safest, 2 may work if audio lengths are short
GRAD_ACCUM          = 16     # Accumulate gradients for effective batch of 16
LEARNING_RATE       = 2e-4   # Good starting LR for ASR, tune lower if model is unstable
NUM_TRAIN_EPOCHS    = 10
EVAL_STRATEGY       = "steps"
EVAL_STEPS          = 1000   # Evaluate less frequently to save memory
SAVE_STEPS          = 1000   # Save less frequently to reduce disk I/O
LOGGING_STEPS       = 50
WARMUP_RATIO        = 0.05
FP16                = torch.cuda.is_available()      # Enable mixed precision
```

```
# If you want to push to the Hub, set these:
PUSH_TO_HUB = True
HF_REPO_ID = "Ed-168/Fine-tuned-wav2vec2-BERT-indian-languages"  # e.g. "username/w2vbert-hi-ctc-cv17"
```

```
# This will download and prepare the dataset (first run may take a while)
from datasets import load_dataset

common_voice_train = load_dataset(
    "mozilla-foundation/common_voice_16_0",
    "hi",
    split="train+validation",
    trust_remote_code=True
)
common_voice_test = load_dataset(
    "mozilla-foundation/common_voice_16_0",
    "hi",
    split="test",
    trust_remote_code=True
)
```

```
NUM_TRAIN_SAMPLES = 1000
NUM_TEST_SAMPLES = 500

common_voice_train = common_voice_train.select(range(NUM_TRAIN_SAMPLES))
common_voice_test = common_voice_test.select(range(NUM_TEST_SAMPLES))
```

```
print(len(common_voice_train))
print(len(common_voice_test))
```

```
1000
500
```

```
common_voice_train = common_voice_train.remove_columns(["accent", "age", "client_id", "down_votes", "gender", "locale", "segment", "up_v
common_voice_test = common_voice_test.remove_columns(["accent", "age", "client_id", "down_votes", "gender", "locale", "segment", "up_vot
```

```
from datasets import Audio

common_voice_train = common_voice_train.cast_column("audio" , Audio(sampling_rate = 16000))
common_voice_test = common_voice_test.cast_column("audio" , Audio(sampling_rate = 16000))
```

```
display10(common_voice_train)
```

```
1 उस राजा से उसकी ताक़त छीन ली गई।
2 प्रियंका चतुर्वेदी और अमृता फडणवीस में ट्विटर वॉर, आरे पर 'भिड़ंत'
3 मैं चाय भी पीती हूँ।
4 तुम्हारी तलवार कहाँ है?
5 तुम्हें संगीत तो पसंद है, है ना?
6 चाहे जो भी हो, मैं कभी भी अपने दोस्त को धोखा नहीं दूँगा।
7 आप क्या पहनने वाले हैं?
8 खेल खतम नहीं हुआ है।
9 क्या तुम उस आदमी को जानते हो?
10 टॉम काफी शक्तिशाली है।
```

```
# Define the regex at the top level so that subprocesses can access it
chars_to_ignore_regex = r"[\"\'\(\)\[\]\{\}\<\>\—\-\-\—\—\-\—\.\,\?\!\:\;\|\d\@\#\$\%\^\&\*\+\=\_\\\/\|~`]+"

def normalize_text(batch):
    text = batch["sentence"]
    text = text.lower()
    text = re.sub(chars_to_ignore_regex, " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    batch["sentence"] = text
    return batch

common_voice_train = common_voice_train.map(normalize_text)
common_voice_test = common_voice_test.map(normalize_text)
display10(common_voice_train)
```

```
Map:   0%|          | 0/1000 [00:00<?, ? examples/s]
Map:   0%|          | 0/500 [00:00<?, ? examples/s]
1 वह अंग्रेज़ी बोलता है क्या
2 उनको दूर से बंदूक की आवाज़ सुनाई दी
3 सूरत में राहुल ने लिया गुजराती चाय का स्वाद नए अंदाज से जीता दिल
4 वही तो
5 मुझे मालूम नहीं कल बारिश होगी या नहीं
6 उल्टा कोतवाल कांग्रेस राज में हुए दंगों पर रिपोर्ट जारी कर 'खुलासा' करेगी बीजेपी
7 बोलो तुम्हे कौनसा चाहिए
8 सूरज मेरे ऊपर है
9 साउथ दिल्ली में इस नई तकनीक से उठाया जाएगा कूड़ा
10 तुमने अपनी कमीज़ उलटी पहनी हुई है
```

```
# Build a set of characters present in the training transcripts
def extract_all_chars(batch):
    all_text = " ".join(batch["sentence"])
    return {"all_text": [all_text]}

vocabs = common_voice_train.map(extract_all_chars, batched=True, batch_size=-1, remove_columns=common_voice_train.column_names)
all_text = " ".join(vocabs["all_text"])
vocab_list = sorted(list(set(list(all_text))))

# Remove the space from the set; we'll add a dedicated word_delimiter_token later.
if " " in vocab_list:
    vocab_list.remove(" ")

# Build vocab dict
vocab_dict = {v: k for k, v in enumerate(vocab_list)}
vocab_dict["|"] = len(vocab_dict)  # word delimiter
vocab_dict["[UNK]"] = len(vocab_dict)
vocab_dict["[PAD]"] = len(vocab_dict)

print("Vocab size:", len(vocab_dict))
print("Sample of vocab keys:", list(vocab_dict.keys())[:60])

# Save vocab to disk
os.makedirs(OUTPUT_DIR, exist_ok=True)
vocab_path = os.path.join(OUTPUT_DIR, "vocab-v2.json")
with open(vocab_path, "w", encoding="utf-8") as f:
    json.dump(vocab_dict, f, ensure_ascii=False, indent=2)
print("Saved vocab to:", vocab_path)
```

```
Map:   0%|          | 0/1000 [00:00<?, ? examples/s]
Vocab size: 92
Sample of vocab keys: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'v', 'w', 'x',
Saved vocab to: w2vbert-hi-ctc-cv16\vocab-v2.json
```

```
# Tokenizer for CTC

from transformers import SeamlessM4TFeatureExtractor

tokenizer = Wav2Vec2CTCTokenizer(
    vocab_path,
    unk_token="[UNK]",
    pad_token="[PAD]",
    word_delimiter_token="|",
)

# Feature extractor (handles audio to input features)
feature_extractor = SeamlessM4TFeatureExtractor.from_pretrained(BASE_MODEL)

# Combined processor (specific to wav2vec2-bert)
from transformers import Wav2Vec2BertProcessor

processor = Wav2Vec2BertProcessor(feature_extractor=feature_extractor , tokenizer=tokenizer)

# # Save processor for later use/inference
# processor.save_pretrained(OUTPUT_DIR)
# print("Processor saved to:", OUTPUT_DIR)
```

```
rand_clip = random.randint(0 , len(common_voice_train) -1 )
print("Target text:", common_voice_train[rand_clip]["sentence"])
print("Input array shape:", common_voice_train[rand_clip]["audio"]["array"].shape)
print("Sampling rate:", common_voice_train[rand_clip]["audio"]["sampling_rate"])
```

```
Target text: पहले खाएंगे फिर जाएंगे
Input array shape: (42624,)
Sampling rate: 16000
```

```
def prepare_dataset(batch):
    audio = batch['audio']
    batch['input_features'] = processor(audio['array'] , sampling_rate=audio['sampling_rate'])
    batch["input_length"] = len(batch['input_features'])

    batch['labels'] = processor(text = batch['sentence']).input_ids

    return batch

common_voice_train = common_voice_train.map(prepare_dataset , remove_columns=common_voice_train.column_names)
common_voice_test = common_voice_test.map(prepare_dataset , remove_columns = common_voice_test.column_names)

print("Example lengths:", len(common_voice_train[0]["input_features"]), len(common_voice_train[0]["labels"]))
```

```
Map:   0%|          | 0/1000 [00:00<?, ? examples/s]
Map:   0%|          | 0/500 [00:00<?, ? examples/s]
Example lengths: 2 23
```

```
print(common_voice_train[0]['input_features'])
```

```
{'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
import numpy as np
```

```
data = [{'attention_mask': [[1, 1, 1, ...]],
         'input_features': [[[-2.3008129596710205, -2.249695301055908, ...]]]}]

input_features = [item['input_features'] for item in data]

input_features
```

```
[[[[-2.3008129596710205, -2.249695301055908, Ellipsis]]]]
```

```
from dataclasses import dataclass
from typing import List, Dict, Any
import torch

@dataclass
class DataCollatorCTCWithPadding:
    processor: Any
    padding: str = 'longest'

    def __call__(self, features: List[Dict[str, Any]]) -> Dict[str, torch.Tensor]:
        # Extract feature arrays and masks from your nested structure
        input_feature_arrays = []
        attention_masks = []
        for item in features:
            nested = item["input_features"]
            # 'input_features': [[...]], 'attention_mask': [[...]]
            feats = nested["input_features"]
            mask = nested["attention_mask"]

            # Remove the extra outer list if present
            if isinstance(feats, list) and isinstance(feats[0], list):
                feats = feats[0]
            if isinstance(mask, list) and isinstance(mask[0], list):
                mask = mask[0]

            # Convert to tensors
            feats = torch.tensor(feats, dtype=torch.float32)
            mask = torch.tensor(mask, dtype=torch.long)

            input_feature_arrays.append(feats)
            attention_masks.append(mask)

        # If feature_extractor expects a list of dicts per sample, pass both feature+mask together
        batch = self.processor.feature_extractor.pad(
            [{"input_features": f, "attention_mask": m} for f, m in zip(input_feature_arrays, attention_masks)],
            padding=self.padding,
            return_tensors="pt"
        )

        # Labels: extract directly
        labels = [item["labels"] for item in features]
        # Pad labels with tokenizer
        label_features = [{"input_ids": l} for l in labels]
        labels_batch = self.processor.tokenizer.pad(
            label_features,
```

```
            padding=self.padding,
            return_tensors="pt"
        )

        # Replace tokenizer padding ID with -100 for CTC loss
        labels = labels_batch["input_ids"].masked_fill(labels_batch.attention_mask.ne(1), -100)
        batch["labels"] = labels

        return batch
```

```
data_collator = DataCollatorCTCWithPadding(processor=processor, padding='longest')
```

```
wer_metric = evaluate.load("wer")

def compute_metrics(pred):
    # pred.predictions is float logits of shape (batch, time, vocab_size)
    pred_logits = pred.predictions
    pred_ids = torch.from_numpy(pred_logits).argmax(-1)

    # Decode predictions and references
    pred_str = processor.batch_decode(pred_ids, skip_special_tokens=True)
    # Replace -100 with pad_token_id for decoding refs
    label_ids = pred.label_ids
    label_ids[label_ids == -100] = processor.tokenizer.pad_token_id
    label_str = processor.batch_decode(label_ids, group_tokens=False)

    wer = wer_metric.compute(predictions=pred_str, references=label_str)
    return {"wer": wer}
```

```
# Initialize the CTC head on top of wav2vec2-bert encoder
model = Wav2Vec2BertForCTC.from_pretrained(
    BASE_MODEL,
    vocab_size=len(processor.tokenizer),
    pad_token_id=processor.tokenizer.pad_token_id,
    ctc_loss_reduction="mean",
    # You can set this to True for long-form training stability with LayerDrop models
    # but w2v-bert-2.0 doesn't use LayerDrop by default.
)

# Make sure the model knows the correct special tokens
model.config.pad_token_id = processor.tokenizer.pad_token_id
model.config.vocab_size = len(processor.tokenizer)
model.to(device)

# Optionally freeze the feature encoder for a few epochs if you have small compute
# (uncomment to try). Often helps stabilize early training.
# if hasattr(model, "freeze_feature_encoder"):
#     model.freeze_feature_encoder()

# Print parameter count
total_params = sum(p.numel() for p in model.parameters())
trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f"Total params: {total_params:,} | Trainable: {trainable_params:,}")
```

```
Some weights of Wav2Vec2BertForCTC were not initialized from the model checkpoint at facebook/w2v-bert-2.0 and are newly initialized
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Total params: 580,589,470 | Trainable: 580,589,470
```

```
# TrainingArguments
training_args = TrainingArguments(
    output_dir=OUTPUT_DIR,
    group_by_length=True,
    per_device_train_batch_size=BATCH_SIZE,
    per_device_eval_batch_size=BATCH_SIZE,
    gradient_accumulation_steps=GRAD_ACCUM,
    save_steps=SAVE_STEPS,
    eval_steps=EVAL_STEPS,
    logging_steps=LOGGING_STEPS,
    learning_rate=LEARNING_RATE,
    num_train_epochs=NUM_TRAIN_EPOCHS,
    warmup_ratio=WARMUP_RATIO,
    fp16=FP16,
    save_total_limit=2,
    metric_for_best_model="wer",
```

```
    greater_is_better=False,
    push_to_hub=PUSH_TO_HUB,
    report_to=["none"],
)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=common_voice_train,
    eval_dataset=common_voice_test,
    tokenizer=processor.feature_extractor,  # ensures padding works
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

print("Trainer is ready.")
```

```
C:\Users\EDWIN\AppData\Local\Temp\ipykernel_21668\1816509248.py:21: FutureWarning: `tokenizer` is deprecated and will be removed in
  trainer = Trainer(
Trainer is ready.
```

```
# common_voice_train = common_voice_train.rename_column("input_features", "input_values")
# common_voice_train
```

```
common_voice_train
```

```
Dataset({
    features: ['input_features', 'input_length', 'labels'],
    num_rows: 1000
})
```

```
train_result = trainer.train()
trainer.save_model(OUTPUT_DIR)
processor.save_pretrained(OUTPUT_DIR)
```

```
print("Training complete. Model and processor saved to:", OUTPUT_DIR)
```

[630/630 8:46:53, Epoch 10/10]

Step	Training Loss
50	15.074300
100	5.441500
150	3.609600
200	3.505900
250	3.473100
300	3.472900
350	3.506500
400	3.469300
450	3.469100
500	3.458500
550	3.495200
600	3.446700

```
# After training
metrics = train_result.metrics
metrics["train_samples"] = len(common_voice_train)
metrics["num_epochs"] = training_args.num_train_epochs

trainer.log_metrics("train", metrics)
trainer.save_metrics("train", metrics)
trainer.save_state()

print(f"Training complete! Model and processor saved to: {OUTPUT_DIR}")
print(f"Epochs: {training_args.num_train_epochs}")
print(f"Metrics: {metrics}")
```

```
Start coding or generate with AI.
```