Speculative Decoding

Speculative Decoding was proposed in Fast Inference from Transformers via Speculative Decoding by Yaniv Leviathan et. al. from Google. It works on the premise that a faster, assistant model very often generates the same tokens as a larger main model.

This project aims to test this using Open AI's Whisper speech transcription model.

Benchmarking Whisper large-v2

```
!pip install torch
!pip install transformers
!pip install accelerate
!pip install datasets
!pip install librosa
!pip install soundfile
!pip install langchain
!pip install sentence-transformers
!pip install numpy==1.26.4
!pip install evaluate
!pip install jiwer
```

```
Downloading rapidfuzz-3.11.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 38.5 MB/s eta 0:00:00
Installing collected packages: rapidfuzz, jiwer
Successfully installed jiwer-3.0.5 rapidfuzz-3.11.0
```

```
import torch
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
torch_dtype = torch.float16 if torch.cuda.is_available() else torch.float32

model_id = "openai/whisper-large-v2"

model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id,
    torch_dtype=torch_dtype,
    # low_cpu_mem_usage=True, # fast loading
    use_safetensors=True, # secure (over pickle)
    attn_implementation="sdpa", # Flash Attention speed-up
)

model.to(device)

processor = AutoProcessor.from_pretrained(model_id)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100%                                                 1.99k/1.99k [00:00<00:00, 119kB/s]
model.safetensors: 100%                                           6.17G/6.17G [00:48<00:00, 226MB/s]
generation_config.json: 100%                                      4.29k/4.29k [00:00<00:00, 253kB/s]
preprocessor_config.json: 100%                                    185k/185k [00:00<00:00, 2.99MB/s]
tokenizer_config.json: 100%                                       283k/283k [00:00<00:00, 4.01MB/s]
vocab.json: 100%                                                  836k/836k [00:00<00:00, 12.4MB/s]
tokenizer.json: 100%                                              2.48M/2.48M [00:00<00:00, 10.1MB/s]
merges.txt: 100%                                                  494k/494k [00:00<00:00, 7.99MB/s]
normalizer.json: 100%                                             52.7k/52.7k [00:00<00:00, 4.03MB/s]
added_tokens.json: 100%                                           34.6k/34.6k [00:00<00:00, 569kB/s]
special_tokens_map.json: 100%                                     2.19k/2.19k [00:00<00:00, 130kB/s]
```

```
from datasets import load_dataset

dataset = load_dataset("hf-internal-testing/librispeech_asr_dummy", "clean", split="validation")
```

```
README.md: 100%                                                   520/520 [00:00<00:00, 11.1kB/s]
validation-00000-of-00001.parquet: 100%                           9.19M/9.19M [00:00<00:00, 31.9MB/s]
Generating validation split: 100%                                 73/73 [00:00<00:00, 257.20 examples/s]
```

```
# time taken test

import time

def time_gen(model, inputs, **kwargs):
    start = time.time()
    outputs = model.generate(**inputs, **kwargs)
    gen_time = time.time() - start
    return outputs, gen_time


from tqdm import tqdm

all_time = 0
pred = []
ref = []
```

```
print("Generating predictions...")
for sample in tqdm(dataset):
    audio = sample["audio"]
    inputs = processor(audio["array"], sampling_rate=audio["sampling_rate"], return_tensors="pt")
    inputs = inputs.to(device=device, dtype=torch.float16)

    output, gen_time = time_gen(model, inputs)
    all_time += gen_time
    pred.append(processor.batch_decode(output, skip_special_tokens=True, normalize=True)[0])
    ref.append(processor.tokenizer._normalize(sample["text"]))

print(f"Total time taken: {all_time:.2f}s")
```

```
Generating predictions...
  0%|          | 0/73 [00:00<?, ?it/s]Passing a tuple of `past_key_values` is deprecated and will be removed in Transformers
The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequence, yo
/usr/local/lib/python3.10/dist-packages/transformers/models/whisper/tokenization_whisper.py:501: UserWarning: The private me
  warnings.warn(
100%|██████████| 73/73 [01:22<00:00,  1.14s/it]Total time taken: 79.89s
```

```
from evaluate import load

wer = load("wer")
print(wer.compute(predictions=pred, references=ref))
```

```
Downloading builder script: 100%  4.49k/4.49k [00:00<00:00, 315kB/s]
0.035072711719418 31
```

Using Speculative Decoding

```
from transformers import AutoModelForCausalLM

assistant_model_id = "distil-whisper/distil-large-v2"

assistant_model = AutoModelForCausalLM.from_pretrained(
    assistant_model_id,
    torch_dtype=torch_dtype,
    low_cpu_mem_usage=True, # fast loading
    use_safetensors=True, # secure (over pickle)
    attn_implementation="sdpa", # Flash Attention speed-up
)

assistant_model.to(device)
```

```
config.json: 100%                                                  2.29k/2.29k [00:00<00:00, 144kB/s]
model.safetensors: 100%                                            1.51G/1.51G [00:35<00:00, 42.7MB/s]
generation_config.json: 100%                                       3.62k/3.62k [00:00<00:00, 207kB/s]
WhisperForCausalLM(
  (model): WhisperDecoderWrapper(
    (decoder): WhisperDecoder(
      (embed_tokens): Embedding(51865, 1280, padding_idx=50257)
      (embed_positions): WhisperPositionalEmbedding(448, 1280)
```

```
def assisted_generate_with_time(model, inputs, **kwargs):
    start_time = time.time()
    outputs = model.generate(**inputs, assistant_model=assistant_model, **kwargs)
    generation_time = time.time() - start_time
    return outputs, generation_time
```

```
            (out_proj): Linear(in_features=1280, out_features=1280, bias=True)
```

```
all_time = 0
pred = []
ref = []

for sample in tqdm(dataset):
    audio = sample["audio"]
    inputs = processor(audio["array"], sampling_rate=audio["sampling_rate"], return_tensors="pt")
    inputs = inputs.to(device=device, dtype=torch.float16)

    out, gen_time = assisted_generate_with_time(model, inputs)
    all_time += gen_time
    pred.append(processor.batch_decode(out, skip_special_tokens=True, normalize=True)[0])
    ref.append(processor.tokenizer._normalize(sample["text"]))

print(f"Total time taken: {all_time:.2f}s")
```

```
  0%|          | 0/73 [00:00<?, ?it/s]From v4.47 onwards, when a model cache is to be returned, `generate` will return a `Ca
/usr/local/lib/python3.10/dist-packages/transformers/models/whisper/tokenization_whisper.py:501: UserWarning: The private me
  warnings.warn(
100%|██████████| 73/73 [00:55<00:00,  1.32it/s]Total time taken: 52.54s
```

```
print(wer.compute(predictions=pred, references=ref))
```

```
0.03507271171941831
```

Conclusion

We end with a slight speed-up in inference time, while maintaining the same WER score.

Start coding or generate with AI.