

OpenAI Whisper に追加学習をさせる試み

2022年12月08日 木曜日



【この記事を書いた人】
とみ（ともとも言う）

地方拠点の一つ、九州支社に所属しています。サーバ・ストレージを中心としたSI業務に携わってましたが、2018年に難病を患い、定期的に入退院を繰り返しながら、現在は技術探索・深堀業務を中心に対応しています。



CONTENTS

1. Whisperの追加学習に挑む2022年の冬
2. 今期はディープラーニングにもっと没頭してみることに
3. Whisperを使って議事録起こしとかできないだろうか？
4. Whisperについて
5. Whisperの学習について
6. 必要なこと
7. アノテーションって
8. いざお試し
9. フайнチューニングじゃなくて転移学習
10. 書き起こして比較じゃ！
11. Whisperの弱点
12. まとめ
13. 参考文献

Whisperの追加学習に挑む2022年の冬

2022年アドベントカレンダー企画だそうです。

いかがお過ごしでしょうか。

私はもう興味を引くものに没頭するしか楽しみがないもんで、PCに向かってぼんやり面白いネタはないかなーと探す日々です。

最近はすっかりディープラーニングにズブズブで、とうとう数式かくのが面倒なあまり手書き入力のためのペンタプレットを買いました。てへ。

今回は9月から10月にかけてStable-Diffusionの後に盛り上がったOpen AI Whisperのネタになります。

今期はディープラーニングにもっと没頭してみることに

先期（2022年上期）ではですね、ディープラーニングとは何かというのをインフラエンジニアに伝え、インフラストラクチャ構築において何を気を付けるべきかについて調査をしていました。ディープラーニングって言葉を聞いただけで「あ、その手の奴はあまりよく知らないんで」って腰が引けるのは少しどうか？と感じることがたまにあり、それで取り組んできました。今期はさらにどっぷりディープラーニングに埋没して、もう少し色々知ることができそうであれば知りたいなと思って取り組みを始めています。

今回取り掛かることになったのが、OpenAIがリリースしているWhisperという文字書き起こしをメインとして作られたディープラーニングモデルです。

Whisperを使って議事録起こしかできないだろうか？

ある日Whisperに触れた私の上司Syucchin氏が「とみー、このWhisper動く環境を検証PCで作ってみて。議事録作れるかどうかを見てみたい。」とおっしゃる。いつもだったらAzure上のNVIDIA A100にお頼み申すわけなんだけど、やっぱり機微情報の詰まったお話を解析させてるので、社内オンプレミスじゃないと難しそうだとのこと。



社内検証PC(過去案件検証用途で構築されたy-morimoto氏の小型のRyzen5マシン)に別途上司にお願いして買ってもらったGPU(本当はQuadro系なGPUが欲しかったけど、筐体サイズの問題で、GeForce RTX3060LHR/12GBに。)を装着し、出来上がった検証PCにWhisperを実装してみる。そしてその環境で動かしたところ、「うーむ、やっぱり固有名詞系が弱いな。とみー、これをなんか追加学習とかできんやろうか？」とのこと。

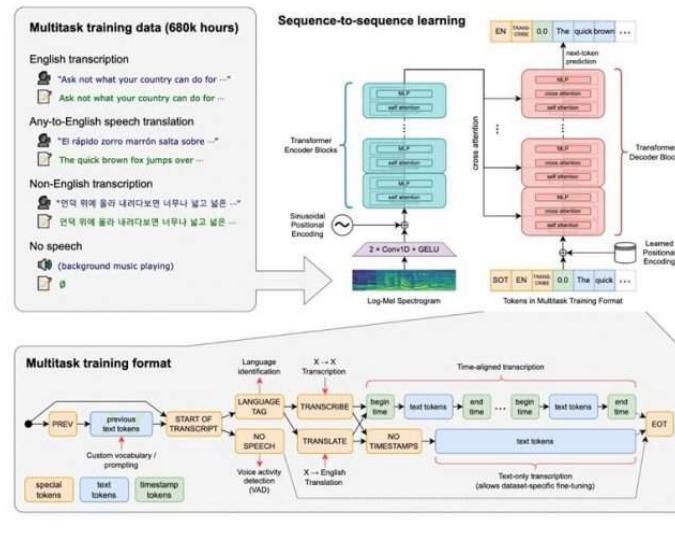
私はいつものノリで「あー、そっすねー。面白そうだしやってみましょうか」と答えてしましました。恐らく追加学習なんて参考例がごろごろしててだろうしそんなに難しくなからう。でもそもそもがディープラーニングどころかPythonの知識すら足りてない私。だいぶ苦労する羽目になったのであります。

Whisperについて

Whisperは先述したように、音声から文字の書き起こしをする事をメインとして作られたディープラーニングモデルの一つで、モデルの構造としては割とシンプルな部類に入ります。

Whisper

- Whisperは音声書き起こしソフトウェアの一種で、OpenAIによって開発されたものである。
- Transformerと構成はほぼ一緒、左半分は音声データを扱う箇所、右半分はメタデータ（言語データ）を扱う箇所である。
- エンコーダとデコーダ間は CrossAttentionLayerによってスペクトログラムと言語が関連付けられている。
- デコーダーに渡すトレーニングデータのフォーマットおよびその内部構造は右図下半分に記載されているとおりである。



事前定義モデル

モデルとして、以下のものが主に提供され (.enモデルは外しました) 、日本語でもそれなりに書き起こしにおいて高い品質を有しています。tinyやbaseはさすがに厳しいものがありますが、はきはきした発言であれば、わりかしsmallでもそれなりの書き起こしができます。mediumやlargeになると句読点や漢字の変換もしっかりした内容になっていきます。

事前定義モデルの一覧

モデル	層数	幅	ヘッド	パラメータ数	実行時 必要メモリ量	備考
tiny	4	384	6	39M	~1GB	
base	6	512	8	74M	~1GB	
small	12	768	12	244M	~2GB	話す内容によっては全く違う応答を返す
medium	24	1,024	16	769M	~5GB	会話が成り立つギリギリの精度
large	32	1,280	20	1,550M	~10GB	ほぼ会話が成り立つ精度

※1 基本的に使用されるテンソルの次元数である。値が多いほど計算量が増える
 ※2 上位モデルであればあるほど、どんなGPUであっても書き起こし速度は遅くなる。

なお、ヘッドというのが表中にあるんですが、ざっくりと処理の並列度と考えてもらえばいいかなと思います。Transformerの長所というのは、RNNではできなかった「処理の並列化」にあります。当然Whisperも並列処理できるように色々なニューラルネットワークの層がくみ上げられています。

モデル自体は単純な構成を取りましたが、学習に使用したデータ量が凄いのです。総合計して68万時間、そのうちマルチリンガル用途に12万時間分の音声データを使用して学習させたとのこと（さらに日本語はその中の7,054時間とのこと）で、時間をかけて苦労して学習させてこのモデルが出来上がったということなのでしょう。

音声データの取り扱い

音声データは、そのままの音声波形を使うのではなく、対数メルスペクトログラムと呼ばれるグラフに変換し、音声の周波数、時系列、そして音の大きさの3つの要素に基づいてベクトル化します。丁度メルスペクトログラムというものは以下のようないmageを持っていただければよいかなと思います。

音声データ

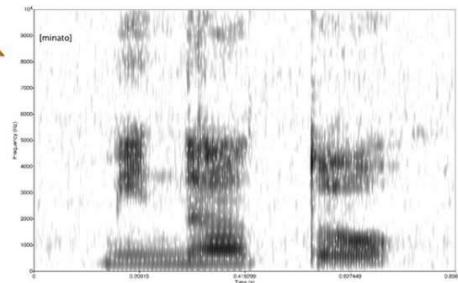
メルスペクトログラムとは

周波数が人間の近くに近いメル尺度(低周波の音を良く知覚する)に変換されたSTFT(短時間フーリエ変換:Short-Time Fourier Transform:関数に窓関数をすらしながらかけて、それをフーリエ変換する手法)

時系列を横軸、周波数を縦軸として、その輝度で音量を表現したスペクトルとなっている。

女性が「みなと」と発音したときのスペクトログラム表現。ここではモノクロ表現のため、色の濃淡で音の大きさを表している。(Wikipediaより引用)

音声データをメルスペクトログラムに変換する処理は、Whisper自身に備わっており、今回はそれを使用している。



音声をこのメルスペクトログラムに変換することで、音声を「画像化している」と考えると良いかもしれません。学習済みパターンにて特徴づけられた音声の高さ・強さのパターンとそのCaptionをベースにして文章を書き起こしていくまします。入力データの次元数は80次元に固定されており、入力する音声データのサンプリングレートも16KHzに指定されています。よって、音声が入る際に他のフォーマットでやってきた場合は内部的にwavフォーマット、16KHzサンプリングレートに変換しているようです。

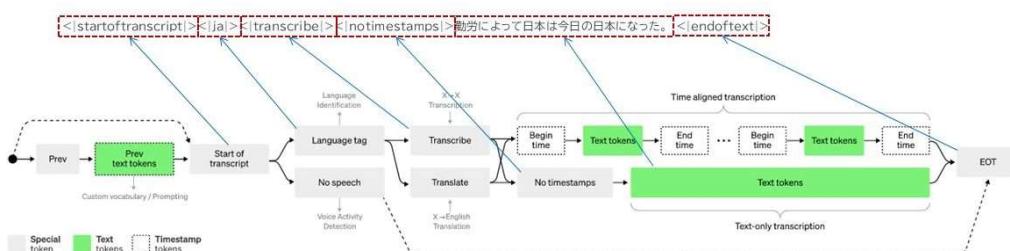
テキスト情報について

テキスト情報はそのままデコーダーに投入するのではなく、付加情報をタグという形で追加します。

学習データの構造

学習データにはタグが付与され、それぞれ読み込むデータがどういうデータであるかを定義している。
「書き起こし対象はどこからか」「言語は何か」「書き起こしか、翻訳か」「時系列を意識した書き起こしか、時系列は無視するか」「該当パターンに対するテキストトークンは何か?」「どこまでがテキストか?」という所を作成している。

現在の書き起こしファインチューニングプログラムとその出力プログラムでは原則No Timestampsというオプションを付けた状態で実行されており、これを時系列書き起こしで実行した場合、コマンド実行時と同じ品質での書き起こしが可能になるものと推察される。



何のためにこうしたタグが付くのかというと

- 情報の開始部分、終端部分の宣言

- 言語種別の宣言
- 何の機能で使用するテキストかの宣言（Whisperには他に翻訳、無音検知などがあります）
- 時系列情報の盛り込み

あたりが主な目的になっています。その宣言文の構造を示すのが上図下半分にある遷移図です。

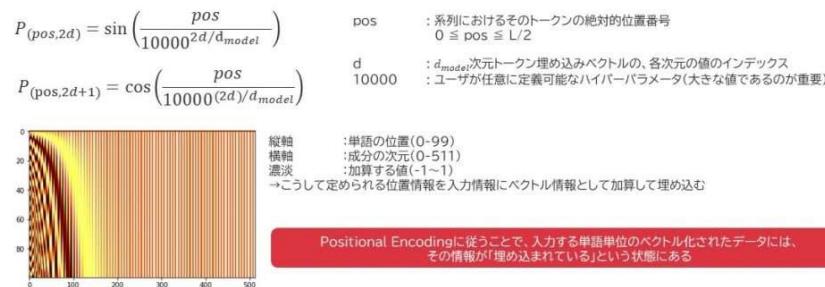
このタグ付け処理はwhisper.tokenizerというライブラリが持っていて、ソースを覗いてみると結構いろいろなタグの付与処理が記載されています。

Positional Encoding

これら入力データはエンコーダ・デコーダそれぞれの入り口から入るわけですが、Transformer系のモデルは再帰的には処理を行わない構造になっているので、位置情報は別途ベクトル化して入力情報に加算する形で組み込まれています。この手法がPositional Encodingという手法です。

Positional Encoding

- Encoder/Decoderに対してまず組み込まれる情報が「位置情報」
 - Seq2SeqはNNの構造そのものが単語の順序を示していた
 - TransformerベースのNNではそれがなくなるため、別途この仕組みが必要になる。



Whisperの学習について

そもそも私は人工知能に対する追加学習をさせたことがありません。

きっとまあGoogle先生に聞けばいっぱいネタがあるだろうと思ったら意外や意外、ネタがありませんぬ。。

原因は単純で、私が思う「追加学習」というものに別の単語が定義されていたからです。一つがファインチューニングです。それに気づくまでに実に2日要しまして、見つけた記事がこちらです。

OpenAIの音声認識モデル Whisperの解説/Fine Tuning方法

<https://zenn.dev/kwashizz/articles/ml-openai-whisper-ft>

Zenn.devに日本語での解説も記載されている

OpenAIの音声認識モデルWhisperの解説 / Fine Tuning方法

<https://zenn.dev/kwashizz/articles/ml-openai-whisper-ft>

ここで解説されている内容に従い、追加学習処理をくみ上げてみることにしました。

Fusic Co., LTD 先進技術部門機械学習チーム所属のk-washiさんはTwitter経由でコメントを取らせていただき、コードの掲載、カスタマイズについて了承を得ることができました。

ちなみにFusicさん(<https://fusic.co.jp/>)は私の所属する山口県九州支社と同じく福岡で活動されているIT企業になります。



Fusic Co., Ltd.

The screenshot shows a Zenn.dev article titled "OpenAIの音声認識モデル Whisper の解説 / Fine Tuning 方法". The content discusses the process of fine-tuning the Whisper model for Japanese. It includes sections on data loading, learning rate scheduling, and validation. A sidebar on the right lists related topics like "Whisperとは", "学習用データ", "Fine Tuning", and "Google Colab実験室".

詳細は実際に記事とGoogleColabo上のコードを参照していただきたいのですが、目を引いたのは

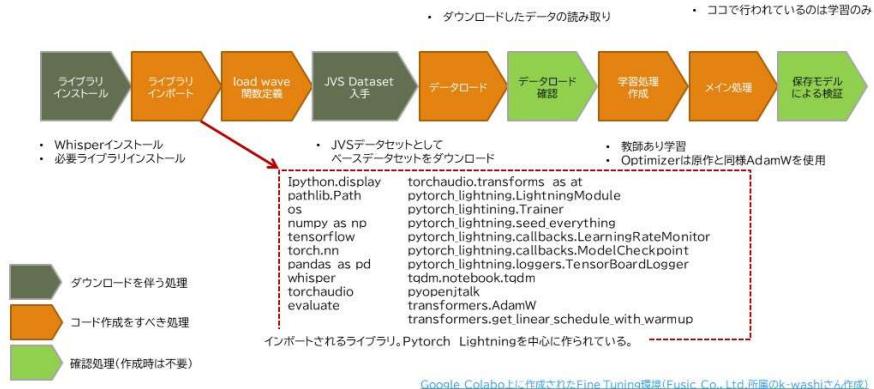
- JVSデータセットの構造
- JVSデータセットの読み出し方
- 学習用データ、教師用データの構造
- 学習用・検証用の仕分け方
- 追加学習の際にすべきお膳立て
- チェックポイントファイルの取り扱い

以上の点でした。このサイトで紹介されていた全体の流れとしては以下のようになっています。ところでJVSデータセットはJapanese versatile speechという総称を持ったデータセットで、高道 慎之介さん(東京大学 大学院情報理工学系研究科 助教)が作成されたデータセットのことです。およそ音声と言語の組み合わせを1セットとして見ると、およそ13,000セットほどあり、コーパスデータとしては大規模です。

GoogleColaborateにあったFine Tuning例

<https://zenn.dev/kwashizz/articles/ml-openai-whisper-ft>

恐らく2022年10月3日時点で唯一日本語で提供されているFine Tuning方法



Google Colab上に作成されたFine Tuning環境(Fusic Co., Ltd 所属のk-washiさん作成)
<https://colab.research.google.com/drive/1P4CILkPmfsaKn2tBbRoOnViGMRKR-EWz?usp=sharing>

全体はPytorch_Lightningというフレームワークを使用しており、モデルに対して各種決められた関数を作成することにより、素のPytorchを使用するよりも簡単にトレーニング（学習）、検証、推論処理が行えるというものでした。元々先期の取り組みでTensorflowについては少しばかりかじったんですが、Pytorchはさっぱりわからず・・・このフレームワークの存在も一つ、無知な私の助けになってくれました。

データロードから学習処理まで



モデル作成を行い、そこからトレーニング（学習）処理を行うまでの流れは上図の通りです。

オレンジの矢印はメインルーチンで実施すること、青色矢印はWhisperModelModuleというクラスで実行されている内容です。

学習時の処理の流れ

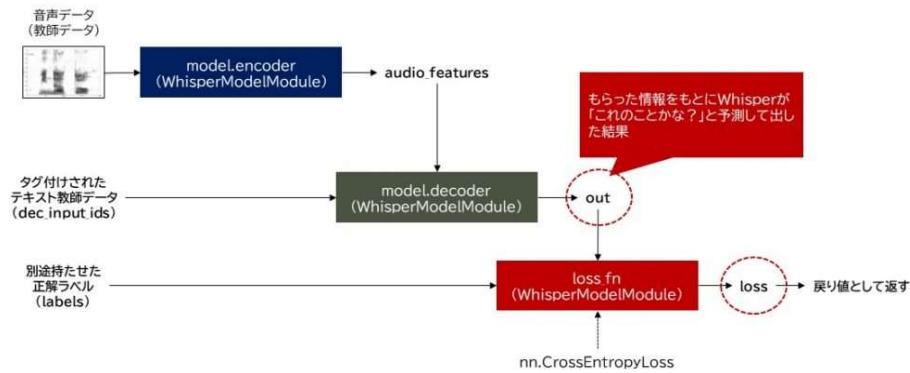
学習時の処理の流れはこんな風になっていました。

学習処理をする前に、k-washiさんスクリプトでは以下のようなことを行っています。

- JVSデータセットからすべてのtranscripts_utf8.txtの絶対パスを掘り起こす
 - こういう時、ほんとうにglobは便利ですね。
 - JVSデータセットはそのコーパス種別によってtranscripts_utf8.txtが設置されている
- audio_idをキーとして、テキストと音声ファイルを結びつける
- TRAIN_RATEに従って学習用・検証用とにデータを振り分ける
 - なお、ケース振り分けはtranscripts_utf8.txtの数をベースにしている
 - 厳密には、transcripts_utf8.txtの数 × TRAIN_RATEとした分の件数が学習用、残りが検証用になる。
- 学習用データセットと検証用データセットのファイル・テキストペアを定義したリスト配列を作る（これがモデルに投入される）

その上で、Whisperをベースとしてモデル作成をし、これを学習モデルとして使うWhisperModelModuleというクラスが定義されています。

学習フェーズ

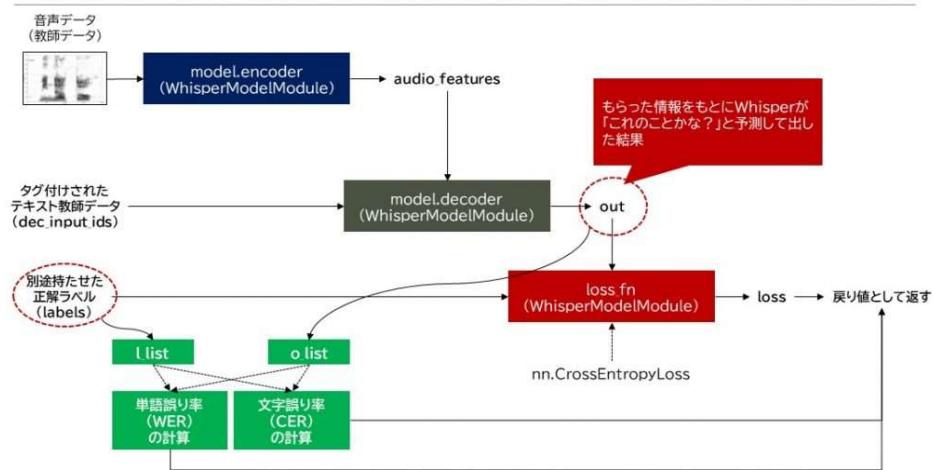


Whisperってとても優秀で、単純にメルスペクトログラム化した音声データをencoderへ放り投げることで簡単にベクトル化された特徴情報として変数に吐きだしてくれます。（そしてそのメルスペクトログラム化の処理もWhisper自身が実装しています）さらに、このAudio_featuresとタグ付けされた教師データをdecoderへ放り込むことで学習処理としてその予測データをout変数へ吐出し、これと正解ラベルをCrossEntropyLossを通じてLOSS値を算出します。

検証フェーズ

検証フェーズは以下のように構成されていました。

検証処理



基本的にはLOSS値算出のロジックは学習時と同じでしたけど、それ以外に単語誤り率(WER)・文字誤り率(CER)の算出が行われていることが分かりました。これらは、のちにTensorboardというツールでグラフ化できるように、所定の領域へログ出力するように作られています。

オプティマイザの影響度を設定する

LOSS値はオプティマイザに送られます。オプティマイザはLOSS値を極力0.0へ持っていくために各種パラメータの変更を行う処理を担います。

これが数億あるいは十数億にわたるニューラルネットワーク内のパラメータを表面上自律的に調整してくれるため、人工知能の挙動も自律的に見えると言っても過言ではない気がします。

オプティマイザは当初SGD（最急降下法）などが登場し、2011年あたりからRMSPropやAdagradなどの方式が登場してきまして、急速に発展を遂げています。私が知ってるオプティマイザで最も新しいものは2014年に発表されたAdamでしたが、これをさらに重み減衰式の変更を行い、より早く収束可能とするようなAdamWというオプティマイザが流行っているようで、Whisperもそれを使用していました。

このオプティマイザはLOSS値をより低い値にするべくパラメータ変更をiterごとに行うわけですが、あまり大きな変化をさせると勾配爆発が起きたり、過学習が発生したりして誰も幸運にはなれません。そこで、ハイパーパラメータと呼ばれる「あらかじめ人が定めなければならないパラメータ」にも気を配る必要があります。スクリプトではConfigというクラスを作成し、これを呼び出すことでWhisperModelModuleへ渡すパラメータを適用していました。

クラス作成時に渡されるハイパーパラメータ

Configクラスに格納。WhisperModel処理実行時に使用される初期パラメータ。

```
class Config:
    learning_rate = 0.0005
    weight_decay = 0.01
    adam_epsilon = 1e-8
    warmup_steps = 2
    batch_size = 2
    num_worker = 8
    num_train_epochs = 10
    gradient_accumulation_steps = 1
    sample_rate = 16000
```

- learning_rate = 0.0005 → 直訳すると学習率。1回の学習でオプティマイザがどの程度weight値、bias値を更新できるかの比率を示す
- weight_decay = 0.01 → 直訳すると重み減衰。オプティマイザのAdamWが使用するオプティマイザにあらかじめ割り当てるパラメータ ϵ の値 $1e-8=0.00000001$ であることを示す。
- adam_epsilon = 1e-8 → 最初の学習時のステップ数
- warmup_steps = 2 → 何ケースを1バッチ単位としてトレーニングさせるかを示す値であり、Batch Sizeが大きくなるほどIteration回数は減少するが、処理にかかる時間も長くなる。一般にはバッチサイズは小さい方が収束しやすいと言われる。
- batch_size = 2 → トレーニングを統括するCPUワーカーの数を指定する。左記ケースは2回だが、CPUコア数に従って、一定量引き揚げることが可能(Ryzen5の場合は12まで指定可能だが、OS処理分の空きを考慮し10とすることが割と多い)
- num_worker = 8 → 一連のIterationを1epochとし、そのトレーニング回数。左記ケースは10epoch実行することを示している。
- num_train_epochs = 10 → 勾配推定の制御値(前頁参照)
- gradient_accumulation_steps = 1 → 音声サンプリングの周波数

特に学習率は結構繊細なパラメータで、オプティマイザが値を調整しようとする際にその影響力をどの程度にするかが決まります。値が大きすぎれば急激にパラメータが変更されて内部構造のバランスが崩れますし、小さすぎるとなかなか誤差修正がきかなくなったりします。また、num_workerは実体のCPUコアの数に合わせて調整します。あまり低すぎるとCPUのとりまとめが追い付かなくなり、パフォーマンスダウンにつながりますし、フルコア指定するとOSが身動き取れなくなつてハングアップする危険性がありますので、だいたい2コアをOS向けに残して残りを割り当てるという傾向があるように思えます。

そして以下がトレーニング時に指定するパラメータです。

トレーニング処理時に渡すパラメータ

これらの値が渡されている。

環境系パラメータともいえる

```
trainer = Trainer(
    precision=16,
    accelerator="gpu",
    max_epochs=cfg.num_train_epochs,
    accumulate_grad_batches=cfg.gradient_accumulation_steps,
    logger=tfllogger,
    callbacks=callback_list
)
```

- precision=16 → 計算にはfloat16を使用する。
- accelerator="gpu" → トレーニング処理にはGPUを使用する
- max_epochs=cfg.num_train_epochs → Configで定めたパラメータを使用する(詳細は前項参照)
- accumulate_grad_batches=cfg.gradient_accumulation_steps → トレーニング処理で出力するログの出力先はTensorboard(tfllogger)とする
- logger=tfllogger → コールバック処理を必要とする際のリスト LearningRateMonitor(学習進捗の表示処理) checkpoint callback(epoch単位で発生するチェックポイント保存処理)
- callbacks=callback_list →

必要なこと

つまり、このスクリプトをうまく活用して追加学習をさせるには以下のようなことをする必要があります。

- アノテーションを行い、音声データと正解テキストを関連付け、データセットを作り出す
- k-washiさんのスクリプトを実行させて、データセットを学習させる
- 学習したデータセットはチェックポイントとして所定の場所に保存する

その上で、「書き起こしを実際にやる」処理はk-washiさんスクリプトには片鱗こそあれ、もっと簡単な方法がありましたがのでそれに従って書き起こしスクリプトを作っています。

アノテーションって

アノテーションというのは、今回は音声データに対応する注釈をつける行為です。

まず、追加で覚えてほしい言葉について学んでもらわなければならないので、会社のミーティング議事を録音したデータを上長から頂くわけですが、このデータにどうやって正解データを刻むんだ？という所から難航しました。なんとなくぼんやりイメージしていたのは、その音声データに何かしらのメタデータタグを付与するのだろうということですが、それに関する情報がまず少ない。そして情報があっても読んでも分からない。

ただ、k-washiさんが作成したサンプルスクリプトの中で教師データとして使用しているJVSデータセットは次項のような構造になっており、音声データとは別にテキストファイルを起こせばわざわざ音声内のメタデータに仕込む必要がないとわかりました。す、素敵！！

JVSデータセットのファイル構造

JVSデータセットは以下のような作りになっています。第一階層はjvsXXX (X:連番) という形式で100個程度に分かれて保存されており、の中にはその声の特質により凡そ4つのディレクトリに分割されています。その配下にはテキストデータであるtranscripts_utf8.txtというテキストファイルとwav24kHz16bitという名前のディレクトリが存在し、そのwav(以下略)ディレクトリにwavファイルが転がっています。

transcripts_utf8.txtファイルには、WAVファイルから「.wav」を外した名前、所謂Audio-IDというものが存在し、コロン区切りを経て音声ファイルで発言している内容のテキストが記載されています。本当にほっとしたのは、別に音声データの中にCaptionを埋め込む必要がなかったという所です。

JVSのデータ構造



そしてJVSデータセットに合わせた構成でデータ構造を組めば、k-washiさんのスクリプトにあるデータローダーがそのまま使えるわけです。

音声データの加工

その為には以下の取り組まねばならぬ壁がありました。

- まずベースデータがMPEG4なので、これをWAVにする
- WAVデータは24kHz16bitで保存する

- 本来はWhisper、16kHzで受け取る必要があるんですが、JVSデータセットが24kHzなのでそれに合わせました。
- ffmpegはディスティネーションファイルの拡張子で何フォーマットにするかを判断しますので、出力ファイルをhogehoge.wavとかにすると勝手にWAVフォーマットしてくれます。
- WAVデータに1対1で対応する正解を作る必要がある（しかもちゃんと人の耳で聞いて）

綺麗に一言を切り取ることはできませんが、オーディオ編集ソフトであるffmpegというモノを使用して、データの変換と分割ができることが分かりました。変換は割と簡単にオプション設定でどうにかなりますが、無音区間の検出とそれによるファイルの分割はちょっと手間なので以下掲載します。

silencedetectの検出

以下のようにして検出が可能

```
ffmpeg -i MJAM.mp3 -af "silencedetect=n=0.1:d=0.5:m=0.ametadata=mode=print:file=ametadata.txt" -f null -
```

```
frame:203 pts:232751 pts time:7.27347
lavfi.silence start=-6.78834
frame:217 pts:2488797 pts time:7.77747
lavfi.silence start=52.4831
lavfi.silence end=53.71769
lavfi.silence duration=1.01769
frame:760 pts:874415 pts time:27.3255
lavfi.silence start=-26.837
frame:771 pts:887087 pts time:27.7215
lavfi.silence end=27.729
lavfi.silence duration=0.00000
frame:1311 pts:1516079 pts time:47.3775
lavfi.silence start=46.8858
frame:1401 pts:1612847 pts time:50.4015
lavfi.silence end=50.4332
lavfi.silence duration=3.54744
```

```
frame:1472 pts:1694639 pts time:52.9575
lavfi.silence start=52.4831
frame:1493 pts:1701770 pts time:53.7135
lavfi.silence end=53.7179
lavfi.silence duration=1.23484
frame:2589 pts:2981423 pts time:93.1695
lavfi.silence start=-92.6758
frame:2673 pts:3078191 pts time:96.1935
lavfi.silence start=22.55122
lavfi.silence end=22.55122
frame:3090 pts:3558575 pts time:111.205
lavfi.silence start=-110.73
frame:3110 pts:3581615 pts time:111.925
lavfi.silence end=111.947
lavfi.silence duration=1.21706
```

-af	オーディオフィルタの使用
n	許容ノイズ(パーセントあるいはデシベル単位で記述する。デフォルト値0.001は-60dBを指す)
d	検知間隔(sec)
m	チャンネル単位の解析設定
ametadata	メタデータの使い方

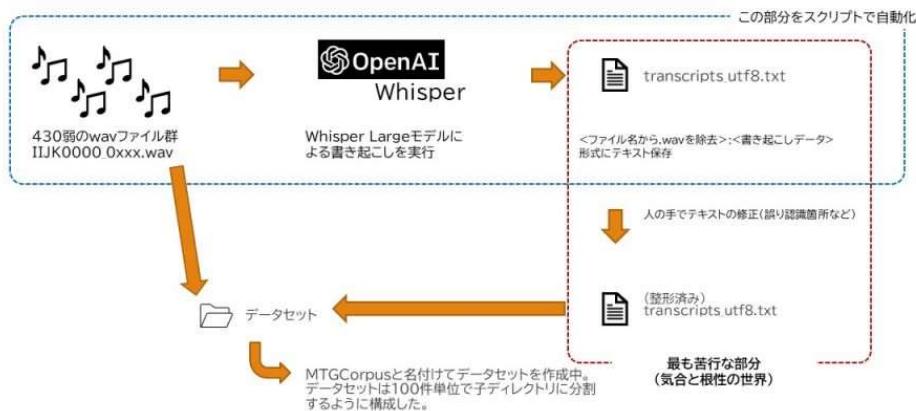
上記ffmpegコマンドを使用することで、どこで条件に合致した区切りができるかというのがわかりますので、次はそれをファイル分割するんですが、ffmpegではこうした無音区間の検出と検出したタイムフレームによってファイルを分けることが1つの操作ではできません。ちょうどGitHubに良いコードを見つけましたのでこれを使用させていただきました。

GitHubGist-vi/split_by_silence.sh

<https://gist.github.com/vi/2fe3eb63383fcfdad7483ac7c97e9deb>

ここまでではサクッと分かったんですが、いざ正解テキストを作ろうとするとこれがなかなか地獄です。一つ一つ手で仕上げようすると、半日かけてわずか60ケースぐらいしか組めませんでした。一番大変なのはリモートの音声の不明瞭さです。事情が分かってないと分からない単語が多すぎて何がなんだかわけが分からぬのです。日本語として成り立たないケースもあり正直困り果てたのですが、「あ、ベース部分をWhisperに書き起こせたらいいじゃん」という気付きを得て、ベースのデータを起こすスクリプトを作成しました。仕組みは以下のよう感じです。

教師データの作成



スクリプト本体としてはこんな感じです。動けばいいやで作ったものなので、なんだか駄コードで申し訳ありません・・

```

1. #
2. #データセット JVS風のベース作成
3. #
4.
5. import os
6. import sys
7. import glob
8. import whisper
9. import pandas as pd
10. from pathlib import Path
11. import pprint
12. import re
13. import shutil
14.
15. # WhisperはLargeモデルを使用する
16. model = whisper.load_model("large")
17. # 元々のデータとして、対象ディレクトリ内にはWAVデータが全て入っていることが条件
18. #DATASET_DIR="/home/whisper/create_dataset/test004"
19. DATASET_DIR= sys.argv[1]
20. dataset_dir=Path(DATASET_DIR)
21.
22. # まずはID順にWAVデータを並べてリスト化する
23. p_temp = sorted(dataset_dir.glob("*.wav"))
24.
25. # サブディレクトリ内のWAVファイル数カウント
26. counter = 0
27.
28. for p in p_temp:
29.     p_filename=str(p.name)
30.     p_audioid=p_filename.replace('.wav', '')
31.
32.     #カウンタがゼロなら、ディレクトリ作成を行う。あつたらあつたでそのまま使う
33.     if counter == 0:
34.         createDir=Path(DATASET_DIR + "/corpus-" + p_audioid)
35.         try:
36.             os.mkdir(createDir)
37.         except:
38.             pass
39.         DATASET_W_FILE = str(createDir) + "/transcripts_utf8.txt"
40.
41.     if counter == 0:
42.         createWavDir = Path(str(createDir) + "/wav24kHz16bit")
43.         try:
44.             os.mkdir(createWavDir)
45.         except:
46.             pass
47.         WAVDATA_FILES = str(createWavDir)
48.
49.     #WAVを再生して書き起こしを実行する
50.     tmp_result = model.transcribe(str(p),

```

```

51.     language="ja",
52.     temperature=0.6,
53.     logprob_threshold=0.15,
54.     no_speech_threshold=0.5)
55.
56.     transtext = str(tmp_result["text"]).replace('\n', '')
57.
58.     #JVSのお作法に従って、オーディオID:テキストという形で書き起こしを行う
59.     transcribe_textfile=open(DATASET_W_FILE,'a',encoding="UTF-8")
60.     transcribe_textfile.write(p_audioid + ':' + transtext + '\n')
61.     transcribe_textfile.close()
62.
63.     #読み終わったファイルはサブディレクトリへ移動
64.     shutil.move(str(p),str(createWavDir) + "/")
65.
66.     counter = counter + 1
67.     print("Counter: " + str(counter))
68.
69.     # サブフォルダ内にWAVデータが100件溜まったらカウンタをゼロリセットする
70.     if counter == 100:
71.         counter = 0

```

ここまでやってみたものの、やっぱりWhisperには厳しいデータが多く、こっから先は人間の手間暇をかけて「聞く」「修正する」の繰り返しです。書き起こされたデータに関しては、特に固有名詞が弱く、「上期」を「紙木」、「下期」を「下木」と書くケースが非常に多かったり、推論の弊害と言いますか、同じ単語がループして収拾付かなくなったりなどいろいろな問題が起きていました。

そこを修正して聞いたまま聞いた通りのニュアンスに1つ1つWAVデータの書き起こしをやっていくところはまさに単純作業なようでとても繊細で集中力を要する作業です。現時点でおよそ1,000ケース程作成したところ（1つの録音データでおよそ500ケース程生成可能）です。

いざお試し

さて、実際のコードなのですが、ぶっちゃけ先述したk-washiさんがGoogle Colaborate上に展開されているコードをそのまま丸コピーしただけです。強いて挙げれば、そこから漢字/ひらがな構成からカタカナに変換する処理を除外しただけなので、本体の掲載や説明は基本的に省くんんですけど、k-washiさんスクリプトにないものを作成する必要がありました。それが書き起こし処理です。

書き起こし処理の作成

学習処理を経て作成されたチェックポイントは.ckptという拡張子を付けた形で保存されており、Smallモデルだと凡そ3GB程度、Mediumモデルだと6GB、Largeモデルだと13GB程度のファイルになります。これをただ読み込めばいいのではなくて、実はモデルを事前に同じような形で再構築してそこに上書きをする必要がありました。

```

1. import IPython.display
2. from pathlib import Path
3. import os
4. import numpy as np
5. try:
6.     import tensorflow # required in Colab to avoid protobuf compatibility issues
7. except ImportError:
8.     pass
9.
10. import torch
11. from torch import nn
12. import pandas as pd
13. import whisper
14. import torchaudio
15. import torchaudio.transforms as at
16. from pytorch_lightning import LightningModule
17. from pytorch_lightning import Trainer, seed_everything
18. from pytorch_lightning.callbacks import LearningRateMonitor, ModelCheckpoint
19. from pytorch_lightning.loggers import TensorBoardLogger

```

```

20.  from tqdm.notebook import tqdm
21.  import pyopenjtalk
22.  import evaluate
23.  from transformers import (
24.      AdamW,
25.      get_linear_schedule_with_warmup
26.  )
27.
28.  class Config:
29.      learning_rate = 0.0005
30.      weight_decay = 0.01
31.      adam_epsilon = 1e-8
32.      warmup_steps = 2
33.      batch_size = 2
34.      num_worker = 2
35.      num_train_epochs = 10
36.      gradient_accumulation_steps = 1
37.      sample_rate = 48000
38.
39.  class WhisperModelModule(LightningModule):
40.      <ここに関してはk-washiさんスクリプトの中身を参照してくださいね。割愛しております。>
41.
42.      cfg = Config()
43.      checkpoint_path = "/home/whisper/content/artifacts/checkpoint/checkpoint=epoch=0008-v4.ckpt"
44.      state_dict = torch.load(checkpoint_path)
45.      state_dict = state_dict['state_dict']
46.
47.      whisper_model = WhisperModelModule(cfg)
48.      whisper_model.eval()
49.      whisper_model.freeze()
50.
51.      whisper_model.load_state_dict(state_dict)
52.      wavdata= whisper.load_audio(f"/home/whisper/corpus/jvs_ver1/jvs002/falset10/wav24kHz16bit/VOIC
53.
54.      result = whisper_model.model.transcribe(wavdata, verbose=True, temperature=0.8, language="ja")
55.      #print(result["text"])

```

WhisperModelModuleはk-washiさんスクリプトの丸コピーで、私が自身で作ったのはその先にあるメインルーチンだけです。

チェックポイントデータから所謂「パラメータ」と呼ばれるニューラルネットワーク上で動作する重みやバイアス情報を上書きせすることにより、学習した結果を反映させることができます。

学習出来たら、オーディオをロードさせて、元々Whisper自体が持ってるtranscribeという関数を実行するだけです。

本来はちゃんとDecoderやTokernizer使ってもう少し低レイヤーで操作して書き起こしたかったのですが、既にそれを駆使して可視化できるように作られた関数があったのでそれを利用した感じです。

おや？

で、動くには動いたんですけど・・・なんだかちょっとおかしい。学べば学ぶほど壊れていくのですよね・・・文章がガガ

しかしやればやるほどおかしくなる現象が…

学習すればするほど日本語が崩れる。一般的な声を書き起こしできない

■Epoch数によるおなじ言語データに対する予測データの違い(JVSコーパスのデータを使用)

・学習なし(元々のWhisper)

ニューイングランド風は牛乳をベースとした白いクリームスープであり、ポストンクラムチャウダーとも呼ばれる。

・epoch1回目

「うイングングングイングング」には、給農をーストした、白いクリムススヌヌツで、トないポストンクラム中どうとも予る。

・epoch4回目

「新イングラウェーは、琉入をバイスとした、白いクリムスープでありオリジェイルトもよいればれる。」

・epoch7回目

「さ新一ネグラウントを テースとして 白いクリムスープであり ポストンクラウンスとも要れる」

・epoch10回目

「の入ingランドFは、流入をベースとして、白いクリムスバーであり、ポストンクラムちアウダーともいよバルえる。」

おいはあさん、クラムチャウダーの話はどこ行ったんじゃ？

実はJVSデータセットに含まれる音声データの中でもVOICEACTRESS系（声優さんを使用した音声データ）を使ってるので、Smallモデルでも十分日本語が理解できるわけなんですが、なぜかこれがepoch増やすに従っておかしくなっていく。なんだこれは・・・ということで調べた結果、圧倒的に学習ケースが少ないということに気づきました。この時点で学習させたケース数は430件程度しかなかったのです。

ファインチューニングじゃなくて転移学習

学習ケースが少なく、かつ、取り込んだ学習モデルが既に高いレベルで完成させている場合、どういう風にしたらよいのかを考えまして、Google先生の検索に頼るも中々それっぽいものが見つかりません。2日ほど試行錯誤して「これもうだめじゃね？」って思ったころにちょうど「転移学習」というキーワードがファインチューニングと併存していることに気づきこれを調べると、今回私がやろうとするものに対しては転移学習が適しているということが分かりました。

そもそもファインチューニングでは現在挿入されたデータの件数も品質もあっていいことが判明。
追加学習方式としてファインチューニングの事しか知らなかつたという無知が露呈

実は「転移学習」の方が適していることが分かった。

→我々としては、Whisperの日本語能力を維持しつつ、よく使われる独特な固有名詞を学ばせたい。

今回参考にしたk-washiさんのスクリプトも実は一種の転移学習のロジックを取っている。
さらに言うと、k-washiさんの場合、サンプルで学習に使用しているものがケース数の多いJVSコーパスである上、処理コストを抑えるためにカタカナ変換処理が組み込まれていた。

基にしたスクリプト:
デコーダー全域のパラメータを
チューニングする手法を採用

本来目指すべきスクリプト:
デコーダー最終層のwaitとbias
だけをチューニングする手法が必要

その為には、実際の構造を一度読み取る必要がある。

そうだ、もっと中身を知ろう！

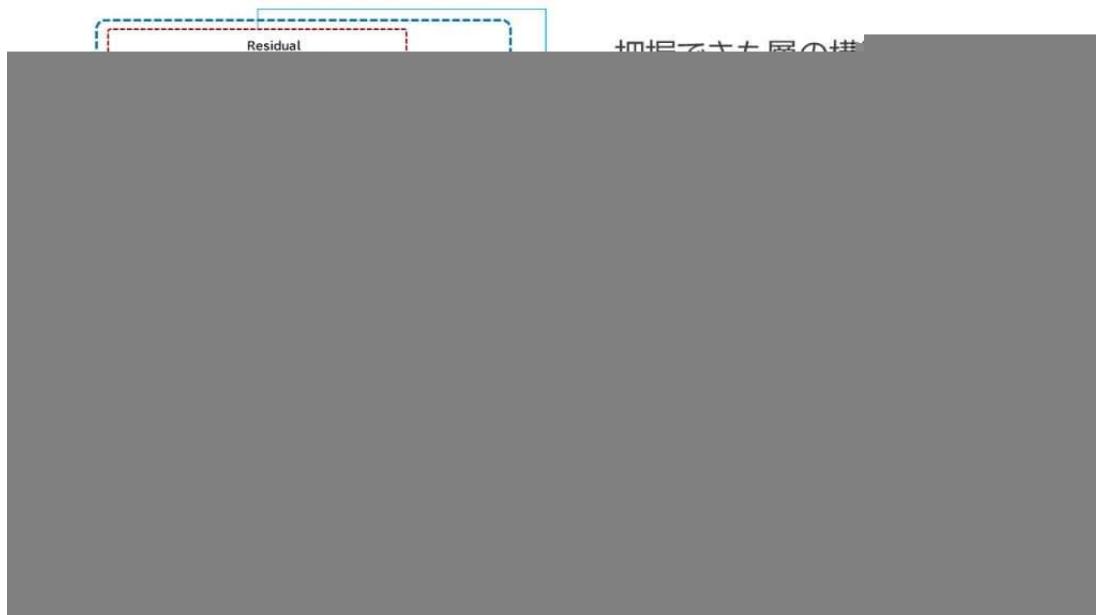
で、そもそもあまり見なくてもよいだろうと思ってたパラメータに対してしっかり理解しなければならないじゃないかということで、もう少し中身を掘り下げて確認してみることにしました。

モデル構造をもう少し細かく見よう

モデル構造の把握



上図はモデル構築後にそれを単純にprintしただけなんですが、こうすることで簡単にニューラルネットワーク上でどんな層が存在し、それが何次元でデータを受け取り何次元でデータを次の層に渡すのかを確認することができます。これをまとめて図式化したのが下図です。



対数メルスペクトログラムデータとして30秒単位で切り出された音声データは、1次元畠み込み処理2回と単語位置の関連付け処理を経て、Encoder本体である32層にもわたるTransformer-Encoderに渡されていきます。これら各層で最終的にエンコードされたベクトルは隣にあるTransformer-Decoder側の各クロスアテンションレイヤーに関連付けられています。これによって、メルスペクトラムという画像と正解情報から抽出した同時系列の言葉が関連付けられます。

変更すべきパラメータとして指定するもの

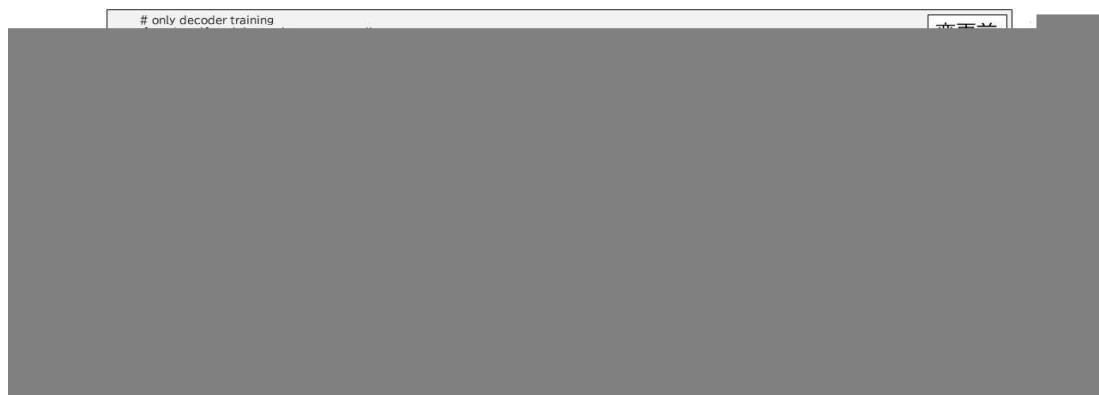
k-washiさんの学習処理では恐らくリソース消費を小さく留めつつファインチューニングを実現するために、Decoder側の全域に対してパラメータ更新を有効にしていたのですが、私の場合はこのうち1層だけ、最後の32層目の既存情報を蓄えることに関連するKey, Value, Outputだけに目を付けてこれに関連するWeightとBiasの値だけを更新してみました。Key/Value/Outputの選択のプロセスは先述した記述ではさも論理的に決定したような書き方をしていますが、実際にはかなり動物的な勘に頼って選択しています。



そして、これら赤字で表現されているパラメータだけ更新するための処理として、既存のWhisperModelModuleについて、以下の通り変更を加えています。

パラメータ更新の抑制(Largeモデルの場合)

WhisperModelModule(LightningModule)内のdef __init__内、
パラメータ定義をしている箇所を書き換える



上記のようにすることで、update_param_namesとして指定された配列内のパラメータ以外は更新対象から外されます。学習結果はDecoder側の32層目に対してのみ反映される形となります。

パラメータ数減少＝消費リソース減少

これにより、うれしいことがまた一つ増えました。私の部署にあるローカル検証用PCでもLargeモデルの学習ができるようになったのです。ローカルPCに搭載しているGPUは世代こそAmpereなんですが、VRAMが12GBしかなく、Mediumモデルですらメモリ不足で学習させられなかったのが、Largeモデルでもギリギリ学習させることができるようになったのです。これは、学習反映先のパラメータが大幅に減少したことが理由のようです。

そして、実際に学習させてみると、上記にもあるようにチェックポイントのサイズもだいぶ減りました。

- Mediumサイズの場合 : 6GiB->3GiB
- Largeサイズの場合 : 13GiB->6GiB

およそ半分ぐらいに減ったようです。

書き起こしだけがなぜかできない

で、早速書き起こしてみようと思ったわけですが、CUDAメモリが足りない！と言われて処理が失敗してしまいます。どうやら、モデル自体Largeモデルは10GBVRAMが必要と言われていますが、それではほとんどVRAMを食いつぶし、チェックポイントファイル内の重み情報やバイアス情報を載せる余裕がなかったようです。これは困った。

すると、「VRAMが足りないの？じゃあDRAMに載せればいいじゃない」という天啓が。確かに、検証PCには16GBのDDR4 DRAMが搭載されています。デフォルトだとModelの設定に従って全部CUDAデバイス上に載せようと動作するわけですが、これをCPUに変更し、いざ書き起こすタイミングでCUDAデバイスへ移動させればいいじゃないということに気づきます。

そこで、以下のように修正を施してメモリの置き場所変更を行いました。

まずは、モデルの置き場所であるDEVICE定数をCPUへ明示的に指定します。

続いてチェックポイントを読み込む際、`torch.load`処理で読みだすのですが、そのマップの配置場所を明示的にCPUに指定します。

チェックポイントデータを既存モデルに上書いた後で、そのモデルをCUDAへ移す処理を追加します。こうして無事、書き起こしまでたどり着くことができました。

書き起こして比較じゃ！

今回書き起こしで使用したのは、会社内の会議録音データで、日程によって日程A、日程B、日程Cと呼称します。

流石に内容全記載は私の首がすっ飛びますので、影響なさげな話題とか職場環境に対して行ってることの話題の中でもあんまり害のなさそうな箇所を摘出しました。一つは多分どこかのお客様への提案内容を検討してるらしい話、もう一つは弊支社内に古くから独自で設置していたIDゲートウェイ（弊社が以前サービス提供しておりましたVPN接続サービス。現在は変わって全社的にフレックスモビリティサービス（もちろんこれも弊社が提供しているサービスの一つ）に切り替わっている）がついに廃止される話ですかね。

比較対象はMicrosoft Teamsの書き起こし機能を使用しています。まだDALL-E2モデルは組み入れてないんだっけ・・？

結果分かったことがあります。

- Whisperは「えー」 「あのー」といった部分は書き起こしから外している
- Whisperは出来るだけ文語で表現しようとしていて、その為に発言者の言葉から文末等を自ら変更している
- Teamsは取り込んだ声をありのままに出力している

Whisperの書き起こしを見てみんながすげー！と驚く理由ってもしかして、書き起こした内容の「読みやすさ」に起因するのではないか？という気がしました。句読点もなく長々と「あのー」が続くので、Teamsが書き起こした文章はとっても読みづらいのです。所々句点を打ってはくれるのですが、やっぱり文章としてはだらだら続いている印象を受ける。

これに対してWhisperは途中わざと空白を開けたりする箇所があるにせよ、割と無駄な言葉が外されて分かりやすくなっています。本来「コメントというかですね」と発言している個所を「コメントというか、」と短く区切っています。こうした細かい気配りができるようにモデルがくみ上げられていることで、パッと見「しっかり文字起こしができている！」と感じやすいのかなという気がします。

実際ここから議事録や速記録を起こそうとするならば、Teamsの方が手間かかりそうだなーという感じは正直致します。DALL-E2がまだ組み入れられてないのであれば、今後その組み込みによってもう少し世界が変わるかもしれませんけれども。

Whisperの弱点

しかし、Whisperにも弱点があります。結構きついのが。

それは、中途半端な静寂に弱いのです。いわゆる「ワイガヤ」に弱いです。会議開始前にみんなでひそひそ雑談していたり、なんか途中でぽろっと聞こえてくるような音のパターンがあると、何が何でも言葉にしようとします。

その際、あまりにも聞き取ったスペクトラムデータが不明瞭になるため、Whisperの恥ずかしい弱点をこれでもかってぐらりとさらけ出します。特に痛いのは単語・短文の無限ループです。下手するとこの無限ループみたいな処理が延々続きすぎてしまうと、本来の発言まで上書きしてしまって重要なポイントの発言が拾えなくなったり、あるいは、書き起こしをする際の発言を何とかひねり出そうとしてたった1秒の間に数分も時間を要してしまうことだってあります。こうしたことは、録音データを投入する際に考慮すべき事項なのだろうと感じています。

データセットの組み方は少し気を付ける必要がありそう

転移学習には、「ドメイン」という考え方があります。既存モデルにおける単語辞書の集まりとしてDsという集合があり、そこへ新たに突っ込もうとするDtという集合を用意する場合、同じ用語に対して異なるベクトルが存在すると結構日本語の構成が崩れることが分かりました。

そこで、今回のケースでは取り込むコーパスデータから、一般的な雑談のような、特定の単語が一度も登場していないケースを探してこれを削除するということも行っています。これによって、少々というレベルでしかありませんでしたが、日本語の崩れを防止することができます。転移学習は少ないケースで辞書の単語を増やすようなアプローチですので、ある程度学習させるケース数は減らしてもよさそうです。

まとめ

取りあえずはお仕事上の固有名詞をある程度反映させることができたということをまとめに入ろうかと思います。

Whisper、確かにすごい！

何が凄いって、あれだけシンプルなニューラルネットワークの構造でここまで書き起こしができるようになってること、その為に実に全言語併せて68万時間分のデータを叩き込んで学習させていること、それだけデータを叩き込みながらもきちんと過学習を回避して高品質なものに仕上がっていることです。日本語はその68万時間の中のわずか7,054時間しか含まれてませんが、それでこの品質はすごいです。漢字変換はよくミスりますが、読み方がミスってないのはさらにすごいですね。（実は、7,054時間という収録時間も他の日本語コーパスと比較して桁が違うほどに多いんですねー）

もう少し掘り下げるところがあるのかなと感じました。

- 音声認識、文法構成力はトップクラスの実力を有する。特に要約能力を併せ持っているように見受けられる。
- 漢字変換に関してはミスが多い。恐らくは漢字変換に伴い必要となるKey/Valueデータが少ない。
 - 雜談系コーパスを使ったものと推察している。雑談系コーパスだと会話で使用する言葉がある程度限られてくるため。
 - この点については今回のように転移学習をする事で一定の改善が見込めるものと予想される。
 - 同じ理由で方言にも弱い。福岡のローカルラジオ番組を録音して適用したところ、書き起こしミスが一定量発生した。（福岡のローカルタレントである朝倉幸男さんの言葉（福岡県の方言の一つ、朝倉弁をよく使う）がうまく解釈できなかった・・・）

追加学習にはデータセットの存在が重要・そして苦行

データセットを作るのは正直骨が折れます。伝え聞いた話ですと、データセットとして教師データを作る際、それを入力するための専用チームが居るんだとかいないんだとか。人手を無茶苦茶要する世界ですよね。

加えて、転移学習の場合は覚えてほしい言葉をとにかく叩き込む必要があるのかなと感じました。ただ、単語そのものを取り込んだだけですとほとんど学習にならず、逆にその単語を使用した会話データを必要としてそうです。よって、データセットの組み方にはそれなりにノウハウが必要そうです。AIを育てるためには人間の叡智がどうしても必要になるのです、多分。

ファインチューニングと転移学習

今回追加学習と言いますか、既存事前学習済みモデルに対して自己流のカスタマイズをするにはどうしたらいいのか？という所を学びました。ファインチューニングはモデル構造そのものを転用して自己流にカスタマイズするものに対して、転移学習は学習済みモデルに「のっかって」少ないサンプルデータでどう環境に適合させるかという手法であることを学びました。

これを通じてPytorch_Lightningというフレームワークに出会えたこと、Fusicのk-washiさんとやり取りできたことを本当にありがとうございます。記事の引用等ご了承いただき誠にありがとうございました。

インフラエンジニアでもナンボかそれなりにAIって扱えるもんなんだなあ

数学の心得は私の場合大学時代でストップしておりましたが、まさかこの年齢になってDeepLearningに手を出すとは思いませんで。

決してレベルは高くありませんが、元来プロダクトソリューションをコアにして活動する私のような人間が中身の仕様を理解しながらチューニングができるなどというのは全く想像だにしておりませんでした。何しろ40代半ばの色々物覚えが怪しくなったり老眼が現れたりして私のですが、2022年3月時点でDeepLearningのDの字も知らんような状態から半年ちょっとでここまでできるようになってきてるわけですから。

ただ、エンジニアとして活動する十数年の中でわりかし中身の見えないプロプライエタリな製品の仕様を挙動で読み取り、その仕様を深堀していくことでそれなりの活動をしてきたことが今回の活動では活きたのかなという気がします。次々と新しい概念や製品が、手法が生まれてくる時代にありますが、技術の中核になるものを理解し学んできたその経験が役に立ったケースもあるのかなと感じています。

若手の方々にはぜひ、そうした中核的な技術・知識を学びながら精進していただけたらなあと思います（この手の技術・知識は己の解釈にかかるところが大きいので、誰かしらから教えてもらうとかは多分難しいです）。何年経過してもぶれない知識ってきっと必ずあったりするので、それはおそらくノイマン型コンピュータがこの世からなくなるまで、役に立ち続けるんだろうなあと思っています。

今後

実は他にも様々なパラメータがWhisper関連の関数には多く存在しており、このあたりの把握が必要と考えています。無音区間の検出や、それに基づいて書き起こしする・しないの設定もできうるので、このあたりの試行錯誤をしたうえで、弱点をより少なく強い点をさらに強化して、少なくとも現職場の内部でそれなりに使い物になるものまでブラッシュアップを図っていきたいなと考えているところです。そしたら本番的な利用用途にそれなりの基盤でこのエンジンを実装できるかなあ・・などと考えています。

あとちょっと気になることとしては、どーしても「お〇しょー」「ちょ〇ど」「し〇やん」というとある方々の固有名詞やら、「〇なたむ」とか言うワードが混じてくるのですが、何かイベントなのか発表会なのかのコーパスデータを使ってらっしゃるんでしょうか・・・これらは何をどう学習させても根絶できなかったので、文字列リプレイスで対処することになりました。合掌・・

追伸

Fusic Co., LTD.さんは私が在籍しております九州支社と同じ福岡県に居を構えるIT企業さんでして、Web系システム技術のみならずこうした機械学習・AI技術に対しても専任チームを構えて活動されてるようです。こうした情報の相互なやり取りの中で互いが成長（と言っても私はだいぶ中年突破ですけど）できるようありたいものです。

なお、IIJ九州支社 事業推進部 技術推進課では技術探索大好きな人を絶賛募集中です。というより、我が課は2022年11月時点でたったの2名しかおらず（結構寂しい）。アイデアマンの役割を完全に上長に頼りまくりな状態なので誰か助けてくださいw

KYU:システムエンジニア（技術調査・検証）

当社の場合、こうして課のまとまり単位で募集をかけるという面白スタイルをとっています。一緒に技術検証・そっから少し進んで研究チックなことに取り組める人を絶賛募集しておりますのでどーぞよろしくお願ひします！

追伸その②－2023/02/16追記

どうもこんばんわ、とみーです。追伸では、今回その他どんなチューニングをしてるのかについて簡単にお話しできればと思い記載を追加することにしました。（新しく記事を起こしてもなんだか内容がちみっちゃいので）

2層に個別の学習をさせた転移学習法

本件、転移学習する際に2つのアテンション層にあるKey/Value入力に対してパラメータチューニングを行うように設定しています。当初は最終層である第31層（層の番号は0開始である点に注意）だけに実行してたのですが、これをちょっと方針を変えて、JVS Corpusの転移学習を1つ手前の第30層で、IIJ九州独自コーパスをいつも通り第31層で学習させたらどうだろうか？ということで試したのですが、これが意外と良い結果を生みました。ケースの膨大なものに関しては、全体を変化させるファインチューニングか、より入力層に近い層も巻き込んで転移学習させるかしたほうがよさそうですね。

Whisperに関してはあまりネタが登場してこないんですが、これはこれでやっぱりすごいモデルだと感じています。Huggingfaceのtransformersにもどうやらこれが組み込まれたようで、より扱いやすくなることが予想されます。さらによく言えば、Largeモデルにv2が登場したこともあり、非英語言語に対する適応性が増したとのうわさも。こちらも、コーパスを時々ケース数増やしながら追加学習をさせていたりします。（large-v1モデル+IIJコーパス転移学習、large-v2モデル+IIJコーパス転移学習、先述の2層学習の3パターンで現在学習処理中だったりします：2023/2/16時点）。

参考文献

GitHub Gist-vi/split_by_silence.sh

[https://gist.github.com/vi/2fe3eb63383fcfdad7483ac7c97e9deb ↗](https://gist.github.com/vi/2fe3eb63383fcfdad7483ac7c97e9deb)

OpenAI Whisper

■リポジトリ

[https://github.com/openai/whisper ↗](https://github.com/openai/whisper)

■公式ブログ

[https://openai.com/blog/whisper/ ↗](https://openai.com/blog/whisper/)

■論文

[https://cdn.openai.com/papers/whisper.pdf ↗](https://cdn.openai.com/papers/whisper.pdf)

OpenAIの音声認識モデル Whisper の解説 / Fine Tuning 方法

[https://zenn.dev/kwashizz/articles/ml-openai-whisper-ft ↗](https://zenn.dev/kwashizz/articles/ml-openai-whisper-ft)

k-washiさんスクリプト(Google Colab上)

[https://colab.research.google.com/drive/1P4ClLkPmfsaKn2tBbRp0nvjGMRKR-EWz?usp=sharing ↗](https://colab.research.google.com/drive/1P4ClLkPmfsaKn2tBbRp0nvjGMRKR-EWz?usp=sharing)

最短コースで分かる、Pytorch深層学習プログラミング

赤石雅典 著、日経BP 発行

[https://www.amazon.co.jp/%E6%9C%80%E7%9F%AD%E3%82%B3%E3%83%BC%E3%82%B9%E3%81%A7%E3%82%8F%E3%81%8E PyTorch-%EF%BC%86%E6%B7%B1%E5%B1%A4%E5%AD%A6%E7%BF%92%E3%83%97%E3%83%AD%E3%82%B0%E3%83%83%A9%E3%83%9F %E8%B5%A4%E7%9F%B3-%E9%9B%85%E5%85%B8-ebook/dp/B09G622WB6 ↗](https://www.amazon.co.jp/%E6%9C%80%E7%9F%AD%E3%82%B3%E3%83%BC%E3%82%B9%E3%81%A7%E3%82%8F%E3%81%8E PyTorch-%EF%BC%86%E6%B7%B1%E5%B1%A4%E5%AD%A6%E7%BF%92%E3%83%97%E3%83%AD%E3%82%B0%E3%83%83%A9%E3%83%9F %E8%B5%A4%E7%9F%B3-%E9%9B%85%E5%85%B8-ebook/dp/B09G622WB6)

Shinnosuke Takamichi, Kentaro Mitsui, Yuki Saito, Tomoki Koriyama, Naoko Tanji, and Hiroshi Saruwatari,

“JVS corpus: free Japanese multi-speaker voice corpus,” arXiv preprint, 1908.06248, Aug. 2019.

[https://arxiv.org/abs/1908.06248 ↗](https://arxiv.org/abs/1908.06248)

Ryosuke Sonobe, Shinnosuke Takamichi and Hiroshi Saruwatari,

“JSUT corpus: free large-scale Japanese speech corpus for end-to-end speech synthesis,”

arXiv preprint, 1711.00354, 2017.

[https://arxiv.org/abs/1711.00354 ↗](https://arxiv.org/abs/1711.00354)

IIJ Engineers blog読者プレゼントキャンペーン

Twitterフォロー＆条件付きツイートで、「IoT米」と「バリーくんストラップ」と「バリーくんシール」のセットを
抽選でプレゼント！

応募期間は2022/12/01～2022/12/31まで。詳細は[こちら](#)をご覧ください。

今すぐツイートするならこちら→

フォローもお忘れなく！



とみ (とみーとも言う)

2022年12月08日 木曜日

地方拠点の一つ、九州支社に所属しています。サーバ・ストレージを中心としたSI業務に携わってましたが、2018年に難病を患い、定期的に入退院を繰り返しながら、現在は技術探索・深堀業務を中心に対応しています。

一緒に働く仲間を募集中！

面白そうなことをしているな、高度な技術で社会貢献しているな、などIIJの取り組みに共感した方は、ぜひ一緒に働きましょう

[採用情報はこちら](#)

Related

関連記事



Python の Newspaper3k ライブラリ

【IIJ 2018 TECHアドベントカレンダー 12/10（月）の記事です】こんにちは。ももいです。先日、共著で書いたblog記事「素人がトピックモデルを試してみた(第1回)」で紹介したようなこ…



ももいやすなり

2018年12月10日 月曜日



ケーブルシップをたずねて三千里

【IIJ 2022 TECHアドベントカレンダー 12/14（水）の記事です】皆様お久しぶりです。(初めての方ははじめてまして) ネットワーク技術部の竹崎です。IIJには2020年度に新卒で入社し、I…



竹崎 友哉

2022年12月14日 水曜日



VMware

【IIJ 2020 記事です】開発を担当する年より提供



y:

20