第8回：ニューラルASRモデル

Conformer / WhisperによるEnd-to-End認識

小野原彩香｜北星学園大学

> Zoom録画を後日公開します。課題はメール/Google Driveでも提出可。

本日の到達目標

1. End-to-End モデルの概念と従来型との違いを説明できる
2. CTC・Attention-based Encoder-Decoder の仕組みを理解する
3. Conformer と Whisper のアーキテクチャ的特徴を説明できる
4. OpenAI Whisper を使って実際に音声認識を行える

従来型とEnd-to-Endモデル

項目	従来型（GMM-HMMなど）	End-to-Endの代表例
構成	音響モデル・発音辞書・言語モデルを統合	入力から出力列を学ぶニューラルモデル
入力	MFCCなどの特徴量	波形やログメル特徴量など
出力	複数モデルを用いてテキストを探索	文字・サブワードなどの列
学習	モデルごと・段階ごとの学習	入力と出力列の対応から学ぶ
時間位置	HMM内で推定する方法や強制アライメント	文字ごとの正解時刻を事前に与えず学習できる

> 「アライメント不要」は、正解の文字に時間位置を付けなくてよいという意味。対応関係そのものがなくなるわけではない。

代表的なEnd-to-Endの学習・出力方式

音声とテキストのペアから、系列の認識を目的とした損失で学習する。

方式	概要
CTC	時間順序を保つ複数の対応経路をまとめる
Attention-based Encoder-Decoder（AED）	出力履歴とエンコーダ出力から次のラベルを生成
RNN-T（Transducer）	音響情報と出力履歴を組み合わせ、対応経路を扱う
ハイブリッド	複数の学習目的・復号方式を組み合わせる

1つのモデルでも、特徴量抽出・エンコーダ・デコーダなど複数のモジュールを含む。外部言語モデルや辞書を併用する構成もある。

CTC：時間位置を与えずに出力列を学ぶ

エンコーダ出力の時刻数T（例：300）と、正解ラベル数L（例：20）は一般に一致しない。

ブランク（_）を含む出力パスを、次の順で変換する。

1. **隣り合う同じラベルをまとめる**
2. **ブランクを除く**

```
c c a a t t  →  c a t       →  cat
c _ a _ t _  →  c _ a _ t _ →  cat
a _ b _ _ b  →  a _ b _ b   →  abb
```

ブランクを挟んだ2つのbは、別のラベルとして残る。
ブランクは、無音を直接表す記号ではない。

CTC損失：同じ答えになるパスの確率を足す

$$\mathcal{L}_{\mathrm{CTC}} = -\log P(Y \mid X) = -\log \sum_{\pi \in \mathcal{B}^{-1}(Y)} \prod_{t=1}^{T} p(\pi_t \mid X)$$

$\mathcal{B}$：繰り返しの統合とブランク除去。$\mathcal{B}^{-1}(Y)$：正解Yになる全パス。

項目	内容
計算	正解長Lに対し、標準的な動的計画法はO(TL)
独立性	入力Xを条件として、各時刻のラベル出力を独立に扱う
ストリーミング	未来の参照を制限したエンコーダとの組合せが必要
復号	最良の1パスと、パスの和が最大のラベル列は区別する

音声フレーム自体が独立という仮定ではない。Graves et al. (2006)

Attention-based Encoder-Decoder（AED）

エンコーダが音声の特徴列hを作り、デコーダが**過去の出力ラベル**とhを参照して次を生成する。

$$e_{ij} = \mathrm{score}(\mathbf{s}_{i-1}, \mathbf{h}_j), \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$$

$$\mathbf{c}_i = \sum_{j=1}^{T} \alpha_{ij} \mathbf{h}_j$$

基本的な再帰型デコーダの式。sは前ステップの状態、αは参照の重み、cは文脈ベクトル。

- 自己回帰の**推論**では、出力を順に生成する
- Transformer型の**学習**では、正解の出力履歴を使って並列計算できる
- 全音声を参照する構成をストリーミングで使うには、先読み範囲などの設計変更が必要

TransformerとSelf-Attentionの復習

RNNでは同じ層の状態を時刻順に更新する。Self-Attentionは位置間の関係を直接計算できる。

$$\mathrm{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_k}}\right)\mathbf{V}$$

- 各ヘッドでは、重みの和が1になる値ベクトルの加重平均
- 学習時の並列化が可能。自己回帰推論は順次生成
- 全位置を比較する計算量は、特徴次元を固定すると$O(T^2)$

> **Conformerも大域的Self-Attentionを使う。**
> 二乗の計算量を解消する方式ではなく、畳み込みで局所情報も扱う構造。

Vaswani et al. (2017)／Gulati et al. (2020)

Conformer：大域的な関係と局所的な構造

Conformerは**エンコーダの構造**。CTCやTransducerなどの出力方式と組み合わせる。

順序	モジュール	役割
1	Feed-Forward	位置ごとの変換。出力を1/2倍して残差へ加える
2	Multi-Head Self-Attention	大域的な関係
3	Convolution	近隣フレームのパターン
4	Feed-Forward	もう一度、出力を1/2倍して残差へ加える
5	LayerNorm	ブロックの最後に正規化

$$\mathbf{x}_{\text{out}} = \mathbf{x} + \tfrac{1}{2}\mathrm{FFN}(\mathbf{x})$$

Macaron形式の1/2は**残差へ加える出力の係数**。FFNの隠れ次元を半分にする意味ではない。

Conformerの性能：外部言語モデルを区別する

Gulati et al. (2020), Table 2。Conformer Transducer、LibriSpeechのWER（%）。

モデル	パラメータ	LMなし clean / other	LMあり clean / other
S	10.3M	2.7 / 6.3	2.1 / 5.0
M	30.7M	2.3 / 5.0	2.0 / 4.3
L	118.8M	2.1 / 4.3	1.9 / 3.9

clean / otherはtest-clean / test-other。LMは外部言語モデル。

- 学習データ、モデルサイズ、復号条件、評価データを確認する
- 言語モデルの有無を混ぜて、モデル構造だけの差と解釈しない
- この表は指定された英語読み上げ音声の条件での結果

Whisper：弱教師あり学習による音声認識

Radford et al. (2022, OpenAI)。ここでは原論文と、実習の多言語版smallを中心に扱う。

- Web上の音声とテキストの対応を使って学習
- 文字起こし・英語への音声翻訳・言語識別を切り替える
- 言語ごとの学習量と精度は異なる

2022年の原論文：合計680,000時間	内訳（概算）
英語の文字起こし	438,000時間
英語以外の文字起こし	117,000時間
他言語の音声から英語への翻訳	125,000時間

後発のモデルは学習データが異なる場合がある。版を区別する。

Whisperのアーキテクチャ

TransformerのEncoder-Decoder。実習のsmallを例にする。

処理	内容
音声の前処理	16 kHz、最大30秒の窓 → 80チャネルのログメル特徴量
2層Conv + GELU	時間方向を1/2に縮小
Transformerエンコーダ	音声表現を計算
Transformerデコーダ	過去の出力と音声表現から自己回帰生成

```
<|startoftranscript|>  # 開始
<|ja|>                 # 言語
<|transcribe|>          # 文字起こし
<|translate|>          # 英語へ翻訳
<|notimestamps|>       # 時刻を出さない場合
```

large-v3は128メルチャネル。80チャネルという説明を全モデルへ一般化しない。

Whisperのモデルサイズ

モデル	パラメータ数	GPUメモリの概数
tiny	39M	約1 GB
base	74M	約1 GB
small	244M	約2 GB
medium	769M	約5 GB
large系	1550M	約10 GB
turbo	809M	約6 GB

公式READMEの概数。実際の速度とメモリ量は機器・設定・音声によって変わる。

実習は`small`。メモリ・待ち時間に応じて`base`や`tiny`へ変更できる。日本語では`.en`なしを使う。

`turbo`は翻訳用には訓練されていないので、翻訳実習には`small`を使う。

Whisperの機能と、結果を確認する観点

機能・特性	意味と確認点
ゼロショット評価	対象データセットで追加学習せずに評価。未見の条件での精度保証ではない
多言語・多タスク	文字起こし、英語への翻訳、言語識別
タイムスタンプ	通常はセグメント単位。単語時刻はattentionとDTWによる追加推定
長い音声	30秒窓を順次処理。既定では前の認識テキストも条件に使う
認識誤り	雑音・沈黙での架空のテキスト、固有名詞などの誤りを確認

`word_timestamps=True` はlarge-v2以降だけの機能ではない。 `language="ja"` は認識条件の指定で、正答を保証しない。

公式transcribe実装。自動言語検出は最初の最大30秒を使う。

ASRフレームワーク・ライブラリの例

名前	用途
ESPnet	End-to-End音声処理のモデルと学習レシピ
SpeechBrain	ASR・TTS・話者処理などのPyTorchツール
Kaldi	GMM-HMMやDNN-HMMなどの音声認識
fairseq	wav2vec 2.0・HuBERTなどの系列モデル
Hugging Face Transformers	対応モデルを共通APIで利用するライブラリ

```
from transformers import pipeline
asr = pipeline("automatic-speech-recognition",
               model="openai/whisper-small", device="cpu")
result = asr("audio.wav", generate_kwargs={
    "language": "japanese", "task": "transcribe"})
print(result["text"])
```

短い日本語音声の例。transformers・PyTorch・FFmpegの導入が必要。次の実習は`openai-whisper`を直接使う。

実習：Whisperを使ってみよう（Step 1〜3）

1：Colabでインストール

```
%pip install -q openai-whisper jiwer
!apt-get -qq update
!apt-get -qq install -y ffmpeg
```

2：デバイスを確認

```
import whisper
import torch

device = "cuda" if torch.cuda.is_available() else "cpu"
print("使用デバイス:", device)
```

3：モデルをロード

```
model = whisper.load_model("small", device=device)
decode_options = {"fp16": device == "cuda"}
```

GPUの割り当ては環境による。CPUでも動くが時間がかかる場合がある。

実習：音声ファイルの認識（Step 4）

```
from google.colab import files

# 短い音声ファイルを1つ選ぶ
uploaded = files.upload()
if not uploaded:
    raise ValueError("音声ファイルを1つ選んでください")
file_path = next(iter(uploaded))

result_file = model.transcribe(
    file_path, language="ja", task="transcribe", **decode_options)
print("認識結果:", result_file["text"])

# これはセグメント単位のタイムスタンプ
for segment in result_file["segments"]:
    start, end = segment["start"], segment["end"]
    print(f"[{start:.1f}s → {end:.1f}s] {segment['text']}")
```

`language="ja"` は日本語として認識する条件を指定する。ここで表示する時刻は**セグメント単位**。

実習：マイク録音して認識（Step 5）

本文のStep 5にある`RECORD_JS`と`record_audio()`の定義を先に実行する。

段階	処理
録音	eval_jsで録音完了のPromiseを待つ
形式の扱い	MediaRecorderの実際のMIME形式で保存し、FFmpegでPCM WAVへ変換
終了	成功・失敗のどちらでもマイクのtrackを停止

```
recorded_path = record_audio(sec=5)
result_mic = model.transcribe(
    recorded_path, language="ja", task="transcribe",
    **decode_options)
print("認識結果:", result_mic["text"])
```

録音データを最初からWAVと仮定しない。変換後は16 kHz・モノラル。

ブラウザ録音を利用できない場合は、端末で録音したファイルをStep 4でアップロードできる。

実習：オプション比較とCER（Step 6〜7）

同じ音声で設定を比較

```
audio_path = file_path  # マイク音声なら recorded_path
result_ja = model.transcribe(
    audio_path, language="ja", task="transcribe", **decode_options)
result_auto = model.transcribe(
    audio_path, task="transcribe", **decode_options)
result_en = model.transcribe(
    audio_path, language="ja", task="translate", **decode_options)
```

日本語の文字起こしをCERで評価

```
from jiwer import cer
reference = "今日の天気はとても良くて気持ちがいい"
# 上の文を audio_path の実際の発話へ書き換える
hypothesis = result_ja["text"].strip()
cer_value = cer(reference.strip(), hypothesis)
print(f"CER: {cer_value:.1%}")
```

両端の空白だけを除き、句読点などは残す設定。全条件で同じ規則を使う。英語翻訳を日本語の正解文とのCERで評価しない。

演習：Whisperの誤りパターン分析

条件例から用意できる2〜3条件を試す。速度・雑音の比較は、同じ文・話者・モデル・認識設定で行う。

条件	正解文	認識結果	CER	誤りの例
読み上げ	…	…	…%	…
速い発話	…	…	…%	…
雑音を加える	…	…	…%	…
自然な方言・アクセント	…	…	…%	…
専門用語	…	…	…%	…

- 実際に話された内容を正解文にする
- 置換・削除・挿入と、表記だけの違いを観察する
- 今回の録音での結果を、方言や条件全体へ一般化しない

CERは挿入が多いと100%を超えることもある。

課題の提出について

提出方法：Google Drive（または メール添付）
締め切り：翌週末 23:59

提出内容：

- 誤りパターン分析表（条件別 CER と誤りの種類）
- 考察（100〜200字程度）

任意追加：最も興味深い誤りパターンを1つ選んで、なぜその誤りが起きたかを推測して書く。

本日のまとめ

1. **End-to-End ASR**：文字ごとの時間位置を事前に与えず、系列の損失で学習できる
2. **CTC**：同じ正解になる対応パスの確率を足す
3. **AED**：出力履歴と音声表現を使って自己回帰デコード
4. **Conformer**：Self-Attentionと畳み込みを組み合わせたエンコーダ
5. **Whisper**：原論文は680,000時間の弱教師あり学習。版・言語・評価条件を区別する
6. **実習**：認識設定と評価の前処理をそろえ、誤りの具体例を分析する

次回予告（第9回）

音声認識システムの評価指標（WER・CER・SLUE ベンチマーク）

- WER・CER を数式に基づいて手計算する
- 編集距離（Levenshtein 距離）との関係を理解する
- SLUE ベンチマークによる意味理解の評価

第9回は動画回です。
Google Forms テストがあります。事前に今日の実習でやった CER の計算に慣れておきましょう。