2024年度後期　修士論文

深層学習による口唇音声変換に関する研究

A Study on the Conversion from Lip-Video to Speech Using a Deep Learning Technique

2025年月日

九州大学芸術工学府音響設計コース

2DS23095M

南 汰翼

MINAMI Taisuke

研究指導教員　鏑木 時彦　教授

概要

目 次

1　序論

1.1　背景

音声は基本的なコミュニケーションの手段であり、人と人とのコミュニケーションの場面において、重要な役割を果たしている。音声は、肺からの呼気流による声帯の振動が音源波を生成し、声道特性に伴ったフィルタリングと口唇からの放射特性に従って生成される。これより、音声の生成には音源を作り出す声帯やその制御のための喉頭、舌や口唇といった調音器官の働きが重要となる。しかし、癌などの重い病気で喉頭を摘出した場合、音源波を生成することができなくなるため、これまで通り発声を行うことが不可能になってしまう。このようなコミュニケーション機能の喪失に対し、現在でも電気式人工喉頭や食道発声、シャント発声といった代用音声手法が存在する。電気式人工喉頭では、専用の発振器を顎下に当てて振動を加えることにより、それを音源とした発声を行う。発振器を用意すれば容易に発声することが可能であるが、生成される音声のピッチが発振器の振動に依存してしまうため、抑揚のない単調な音声になってしまう。食道発声では、まず口や鼻から食道内に空気を取り込み、その空気を逆流させることで食道入口部の粘膜を振動させることによって発声する。電気式人工喉頭と違って道具を必要とせず、ピッチも本人が調節できるが、その習得に長期間の訓練を要する。シャント発声では、手術によって気管と食道を繋ぐ管を設ける。これにより、息を吐き出す際に設けられた喉の穴を手で塞ぐことによって肺からの呼気流が食道に流れる。そのため、食道発声と同様に粘膜の振動を音源とし、発声することが可能となる。習得は容易であり、比較的自然に話すことが可能となるが、設けられた管を交換するための定期的な手術が必要となる。このように、現在用いられている代用音声手法にはそれぞれデメリットが存在する。

そのため、本研究ではビデオカメラで撮影した口唇の動きから音声合成を行うことによる、新たな代用音声手法を検討する。本来、音声は声帯の振動や声道の形状に依存して生成されるものであり、口唇の動きのみから音声波形を直接推定することは困難である。そこで、近年画像や自然言語処理、音声といった分野において成果を上げている深層学習を使用し、データ駆動型の方法で口唇の動きと音声の間の関係を学習することで推定を行う。これにより、従来の代用音声手法よりも自然性の高い音声を、訓練や定期的な手術の必要なく提供することを目指す。

これまでの動画音声合成は英語が中心に検討が進んでおり、近年では YouTube 上のデータを収集、処理することによって構築した大規模データセット [1, 2] を用いることで、大規模で表現力の高いモデルが構築可能となっている。これにより、従来行われてきた教師あり学習のみならず、動画と音声の関係性を自己教師あり学習（Self Supervised Learning; SSL）によって学習し、そのモデルを動画音声合成や、動画からテキストを推定する Visual Speech Recognition（VSR）に FineTuning するアプローチが提案され、その有効性が示されている。自己教師あり学習モデルにもいくつかの種類があり、近年多くの研究で応用例のある AVHuBERT [3] は、動画・音声の入力領域においてマスクされた区間の予測と、予測対象の更新を繰り返して学習を進めていくモデルである。予測対象の更新は 5 回行われ、1 回目は音声波形から計算される MFCC をクラスタリングした結果を利用するが、2 回目以降はモデルの中間特徴量をクラスタリング

した結果を新たな予測対象に設定する。更新のたびに再度モデルをランダム初期化して再学習するが、その予測対象の複雑さが増していくことによって学習を促進するようなメカニズムとなっている。また、これに類似した VATLM [4] は、動画と音声のみならずテキストも加えた学習によって、精度改善を達成した。その他、Student と Teacher という二つのネットワークを利用し、Teacher から出力される特徴量を Student がマスクされた入力から予測することによって学習を進める RAVEn [5] や AV-data2vec [6]、RAVEn の改善版として提案された BRAVEn [7] など、多くのモデルが提案されている。

近年の動画音声合成や VSR では、こういった SSL モデルを動画からの特徴抽出器として活用しつつ、さらなる工夫によって精度改善を達成している。動画音声合成について、 [8] では予測対象として従来用いられてきたメルスペクトログラムに加え、テキストを予測するマルチタスク学習手法を提案した。損失においては上記の二つに加え、予測したメルスペクトログラムを事前学習済みの ASR（Automatic Speech Recognition）モデルに入力して得られる特徴表現も採用した。音声波形はメルスペクトログラムに対して Griffin-Lim アルゴリズムを適用することで獲得しており、従来のメルスペクトログラムのみを損失とする手法に対して客観評価指標における改善を達成した。これに続き、 [9] では前述した手法がテキストアノテーションされたデータのみにしか用いることができないという課題を解消するため、テキストと同様に言語的な情報を持つと考えられている、音声 SSL モデルの HuBERT[10] から得られた離散特徴量（HuBERT 離散特徴量とする）を利用する手法を提案した。また、予測されたメルスペクトログラムと HuBERT 離散特徴量の両方を入力とする Multi-input Vocoder、Multi-input Vocoder の学習時にメルスペクトログラムにノイズをかけるデータ拡張を合わせて提案し、客観評価と主観評価の両方で改善を達成した。加えて、ここでは AVHuBERT の転移学習についても合わせて検討が行われ、これによってさらに性能を改善できることを示した。手法 [9] に関連して、上記のようなマルチタスク学習手法以外にも、HuBERT 離散特徴量や HuBERT 連続特徴量（離散化しない場合を指す）を音声波形までの中間特徴量として扱う手法は提案されている。例えば、 [11] ではメルスペクトログラムの推定を行わず、HuBERT 離散特徴量のみを推定して音声波形に変換する手法が提案された。 [9] では離散化におけるクラスタ数を 200 にしていたのに対して、 [11] ではクラスタ数を 2000 と大きく取っている点で実装が異なっている。メルスペクトログラムを省略する分情報圧縮の程度を軽減することで、音声波形への変換に十分な情報を保持する目的があると考えられる。また、[9] や [11] では AVHuBERT を直接動画音声合成に FineTuning していた一方で、 [12] では AVHuBERT を VSR によって FineTuning し、その後重みを固定した上で特徴抽出器として利用するアプローチを提案している。動画音声合成モデルは、VSR で FineTuning した AVHuBERT から得られる動画特徴量を入力とし、HuBERT 特徴量を予測するネットワークを導入して、HuBERT 特徴量のみから音声波形に変換するボコーダを利用することで構築される。ここでは HuBERT 特徴量として連続値および離散値の両方が検討され、連続値を用いる場合の方が客観評価指標が改善することを報告している。検討された離散値のクラスタ数が 100 であったため、[11] の結果と合わせると、HuBERT 離散特徴量のみで音声波形に変換するアプローチを取るのであれば、クラスタ数を十分大きく取る必要があ

ることが予想される。

一方 VSR について、 [13] では音声認識を利用して言語情報を格納したメモリを用意し、メモリと動画特徴量の間でアテンションをとることによって、ネットワーク内部で言語情報との関連を考慮する構成を提案した。また、 [14] では AVHuBERT が動画あるいは音声のどちらを入力とした場合でもクロスモーダルな特徴量を返すことに着目し、音声認識デコーダに組み合わせる AVHuBERT の Few-shot Learning、Zero-shot Learning による転移学習を検討した。加えて、同様に音声認識デコーダを転移学習するアプローチであるが、事前学習済みモデルの重みを固定し、動画特徴量から音声認識モデルの中間特徴量を予測するネットワークのみを新たに学習することで、両者を合併するようなアプローチ [15] も提案されている。さらに、静止画像と音声から動画を合成するネットワークを構築し、音声認識用のデータセットを用いて VSR の学習データを大量に合成するデータ拡張手法 [16] や、事前学習済みの音声認識モデルによって教師なしデータにラベリングを行うデータ拡張手法 [17]、10 万時間分の教師ありデータを新たに増強した研究 [18] など、大規模な学習データを確保することで精度改善を達成した例も報告されている。

上記の研究は英語データを用いたものであったが、VSR においては英語以外の言語に焦点を当てた研究や、多言語対応モデルの構築も検討が進んでいる。 [19] では RAVEn を利用し、英語に加えてスペイン語、イタリア語、ポルトガル語など計 6 種類の言語が含まれるデータセット [20, 21, 22] を用いて多言語モデルの構築を検討した。結果として、教師ありデータの少ない英語以外の言語に対する、多言語モデルの有効性が明らかとなった。また、 [23] では英語データで学習された AVHuBERT を用いつつ、特定の言語ごとに構築した音声認識モデルのデコーダを転移学習することで、特定言語ごとにモデルを構築するアプローチを提案した。さらに、[24] では音声認識モデルである Whisper を利用し、教師なしデータへのラベリングによるデータ拡張を行うことで、上記二つのアプローチを超える精度を達成した。

本研究では、近年の主流とも言える英語大規模データセットを用いた実験は計算機のスペックの都合上難しく、世界的に見て日本語での動画音声合成の検討例が少ないことも考慮して、文献 [25, 26] で収録された日本語データを用いて研究を行うこととした。英語データと比較して小規模なデータである分性能に課題を抱えたが、予備実験として英語データで学習された AVHuBERT の FineTuning を検討したところ、スクラッチで構築したモデルと比較して、より高い精度を示すことが明らかとなった。これは、英語データを用いた事前学習済みモデルの多言語対応を検討した先行研究の傾向にも一致する結果であり、日本語においても同様に有効なモデルだと考えられる。しかしながら、それでも依然として合成音声の品質は低く、自然音声に迫る合成音は実現されていないことが課題である。

1.2 目的

本研究の目的は、動画音声合成によって得られる合成音声の品質を向上させることである。近年高い精度を達成した手法 [9] では、AVHuBERT の利用および、メルスペクトログラムと音

声 SSL 離散特徴量を利用したマルチタスク学習が採用されている。その他にも近年高い精度を達成したモデルは存在 [11, 12, 27] するが、手法 [9] が採用しているマルチタスク学習の有効性は、テキストを用いた先行研究 [8] でも同様に示されている。これより、このアプローチが現状特に有効そうだと判断し、本研究においてはこの手法をベースラインとして、さらなる改善を狙う形で研究を進めることとした。この手法では、動画を入力としてメルスペクトログラムと音声 SSL 離散特徴量を推定し、これら両方を Multi-input Vocoder に入力することで音声波形へと変換する。しかし、動画と音声の間には、同様の口の動きであっても声道形状の違いによって生じる発話内容の曖昧さや、話者によるパターンの多様さが存在すると考え、推定を動画のみに依存した先行研究の手法ではこういった側面への対処が難しいと考えた。これに対して本研究では、音声 SSL モデルである HuBERT を利用した動画音声合成モデルを提案し、合成音声の推定残差を HuBERT を利用した後処理によって軽減することで、合成音声の品質改善を狙った。HuBERT は、音声波形を畳み込み層を通すことによってダウンサンプリングしつつ特徴量に変換し、ここでマスクをかけた上で Transformer 層を通す。そして、マスクされたフレームにおける予測対象を推定する、Masked Prediction を行うことで学習する。大規模な音声データを用いてこの自己教師あり学習を行うことで、音声のコンテキスト自体をデータそのものから学習することが可能であり、音声認識において有効性が確認されている。本研究では、大規模日本語音声データで事前学習済みの HuBERT を活用し、動画音声合成モデルにおいて生じる推定残差を、音声自体のコンテキストを考慮する形で補うようなアプローチを検討した。

1.3 本論文の構成

2　音声信号処理

音声にはフォルマントや基本周波数（ピッチ）など、様々な周波数的な特徴が存在している。フォルマントは母音や子音を知覚するため、ピッチはアクセントやイントネーションを表現するために重要なものである。このような音声信号の持つ複雑さから、時間波形のままその特徴を分析することは困難である。これに対し、本節では音声の特徴を捉えやすくするための信号処理について説明する。

2.1　音声のフーリエ変換

音声の時間波形に対して、周波数領域での情報を得るためにはフーリエ変換（Fourier Transform）を使用する。特に、音声はマイクロフォンで収録され、コンピュータ内で処理されることが多い。この時、音声はアナログ信号ではなく、サンプリング周波数と量子化ビット数に従って離散化されたデジタル信号として扱われる。このような場合、離散信号に対してのフーリエ変換である離散フーリエ変換（discrete Fourier transform; DFT）が用いられる。また、信号の系列長をゼロパディングして 2 の冪乗の長さに調整することで、計算量を抑えた高速フーリエ変換（fast Fourier transform; FFT）を用いることができる。

離散信号を $x[n]$、それに対するフーリエ変換を $X[k]$ とする。ここで、n はサンプルのインデックス、k は周波数インデックスである。$X[k]$ は複素数であり、以下のように極座標表示することができる。

$$X[k] = \mathrm{Re}(X[k]) + j\mathrm{Im}(X[k]) \tag{2.1}$$

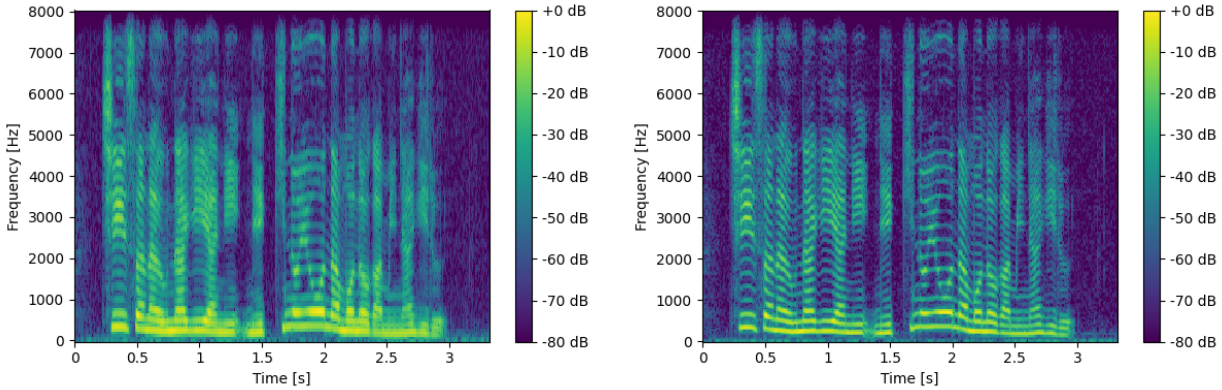
$$= |X[k]|e^{j\angle \mathrm{X[k]}} \tag{2.2}$$

ここで、$|X[k]|$ は振幅特性（振幅スペクトル）、$\angle$X[k] は位相特性（位相スペクトル）であり、以下の式で表される。

$$|X[k]| = \sqrt{\mathrm{Re}(X[k])^2 + \mathrm{Im}(X[k])^2} \tag{2.3}$$

$$\angle \mathrm{X[k]} = \tan^{-1} \frac{\mathrm{Im}(X[k])}{\mathrm{Re}(X[k])} \tag{2.4}$$

また、$|X[k]|^2$ はパワースペクトルと呼ばれる。これにより、信号中にどのような周波数成分がどれくらい含まれているかを調べることができる。しかし、音声はフォルマントやピッチが時々刻々と変化するため、信号全体に対して直接フーリエ変換を適用したとしても有用な結果が得られない。このような音声の持つ非定常性の問題に対して、十分短い時間幅においては信号の定常性が成り立つという仮定のもと、短時間フーリエ変換（short-time Fourier transform; STFT）が用いられる。STFT では、音声信号に対して窓関数による窓処理を適用し、短時間に区切られた信号それぞれに対して DFT を適用する。ここで、窓処理とはある特定の窓関数と音声信号を時間領域でかけ合わせることであり、窓関数の時間幅を窓長という。また、窓関



(a) 窓長 12.5ms、シフト幅 5ms

(b) 窓長 25ms、シフト幅 10ms

(c) 窓長 50ms、シフト幅 20ms

(d) 窓長 100ms、シフト幅 40ms

図 2.1: 「小さな鰻屋に、熱気のようなものがみなぎる」と発話した音声から計算された対数パワースペクトログラム

数を時間方向にシフトするときの時間幅をシフト幅という。STFT には時間分解能と周波数分解能の間に不確定性が存在し、両者の間にトレード・オフの関係がある。窓長が長い場合には周波数分解能が向上する一方、時間分解能が低下する。窓長が短い場合にはその逆となる。音声信号 $x[n]$ の STFT を時刻 j、周波数インデックスを k として $X(j,k)$ と表すと、$X(j,k)$ は時間周波数領域における複素数となる。これを複素スペクトログラムと呼ぶ。また、$|X(j,k)|$ を振幅スペクトログラム、$\angle \mathrm{X(j,k)}$ を位相スペクトログラム、$|X(j,k)|^2$ をパワースペクトログラムと呼ぶ。「小さな鰻屋に、熱気のようなものがみなぎる」と発話した音声に対し、窓関数としてハニング窓を用いた上で、複数の窓長・シフト幅によって計算した対数パワースペクトログラムを、図 2.1 に示す。窓長が 100ms と長い場合には周波数分解能が高いが、時間分解能が低下することでスペクトルの時間変化が滑らかでないことがわかる。一方、窓長が 12.5ms と短い場合には時間分解能が高いが、周波数分解能が低下することでスペクトルがぼやけていることがわかる。これが窓長に対する時間分解能と周波数分解能とトレード・オフであり、窓長 25ms や 50ms が程よいパラメータであることがわかる。

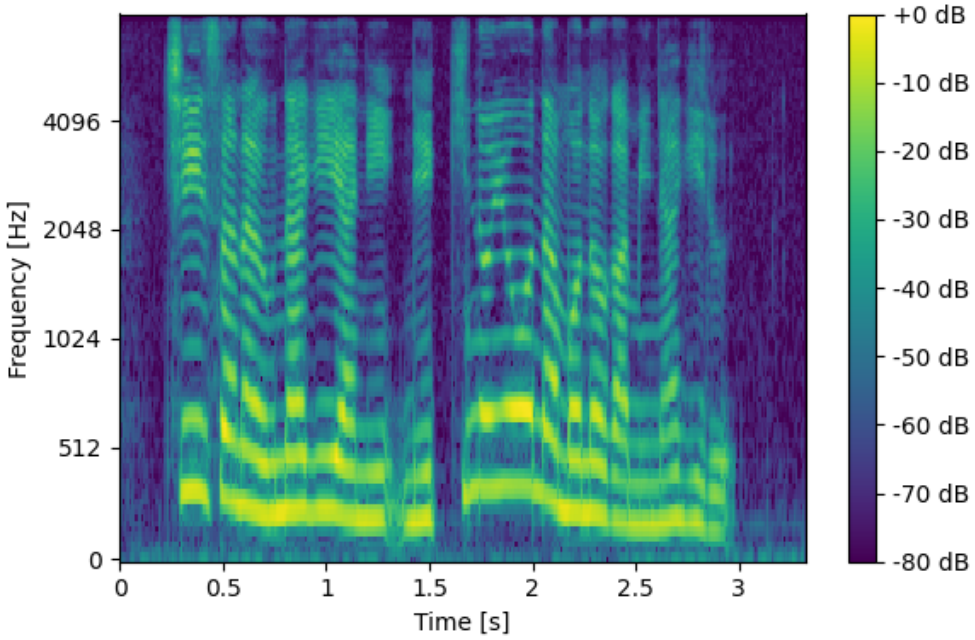


図 2.2: 「小さな鰻屋に、熱気のようなものがみなぎる」と発話した音声に対する対数メルスペクトログラム

2.2 メルスペクトログラム

メルスペクトログラムは、パワースペクトログラムを人間の聴感特性を考慮したメル尺度に変換することによって得られる。周波数軸をメル尺度に変換する際、以下の式を用いる。

$$Mel(f) = 2595 \log_{10} \left(1 + \frac{f}{700}\right) \tag{2.5}$$

メル尺度は、1000 Hz、40 dB の純音を 1000 mel とする比率尺度である。メル尺度を用いることにより、低い周波数ほど細かく、高い周波数ほど荒い特徴量になる。メルスペクトログラムは、パワースペクトログラムに対してメルフィルタバンクを適用することによって得られる。メルフィルタバンクの数は任意に決定できるパラメータであり、メルスペクトログラムの周波数方向の次元はこれに一致する。音声合成においては、音声のサンプリング周波数を 16 kHz とするとき、メルフィルタバンクの数を 80 とし、8000 Hz までの帯域に対して適用することが多い。「小さな鰻屋に、熱気のようなものがみなぎる」と発話した音声に対し、窓関数にハニング窓を用い、窓長 25ms、シフト幅 10ms としてパワースペクトログラムを計算した上で、80 次元のメルフィルタバンクを適用して得られた対数メルスペクトログラムを、図 2.2 に示す。

3　深層学習

深層学習とは、人間の神経細胞の仕組みを模擬したニューラルネットワークを用いる機械学習手法のことである。特に近年ではその層を深くしたディープニューラルネットワーク（Deep Neural Network; DNN）が用いられ、大量のパラメータを最適化することによって得られる表現力により、自然言語処理や画像処理、音声認識や音声合成など様々な分野で成果を上げている。本章では、本研究で使用するニューラルネットワーク及び、構築した DNN の学習方法について説明する。

3.1　ニューラルネットワーク

3.1.1　全結合層

全結合層は、すべての入力ノードがすべての出力ノードに接続される層である。全結合層の出力は、入力に対して学習可能な重みによる線形変換を適用することで得られる。入力 $\boldsymbol{x} \in \mathbb{R}^n$ に対し、出力 $\boldsymbol{y} \in \mathbb{R}^m$ は、

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{3.1}$$

で計算される。ここで、$\boldsymbol{W} \in \mathbb{R}^{m \times n}$ は重み行列で、各要素 w_{ij} は入力の j 番目の成分と、出力の i 番目の成分の間の重みである。$\boldsymbol{b} \in \mathbb{R}^m$ はバイアスベクトルで、各要素 b_i は出力の i 番目の成分のバイアス項である。全結合層は特徴量の次元の変換に用いられ、特に最終層において、ネットワークの特徴量を所望の特徴量の次元に変換する場合に便利である。

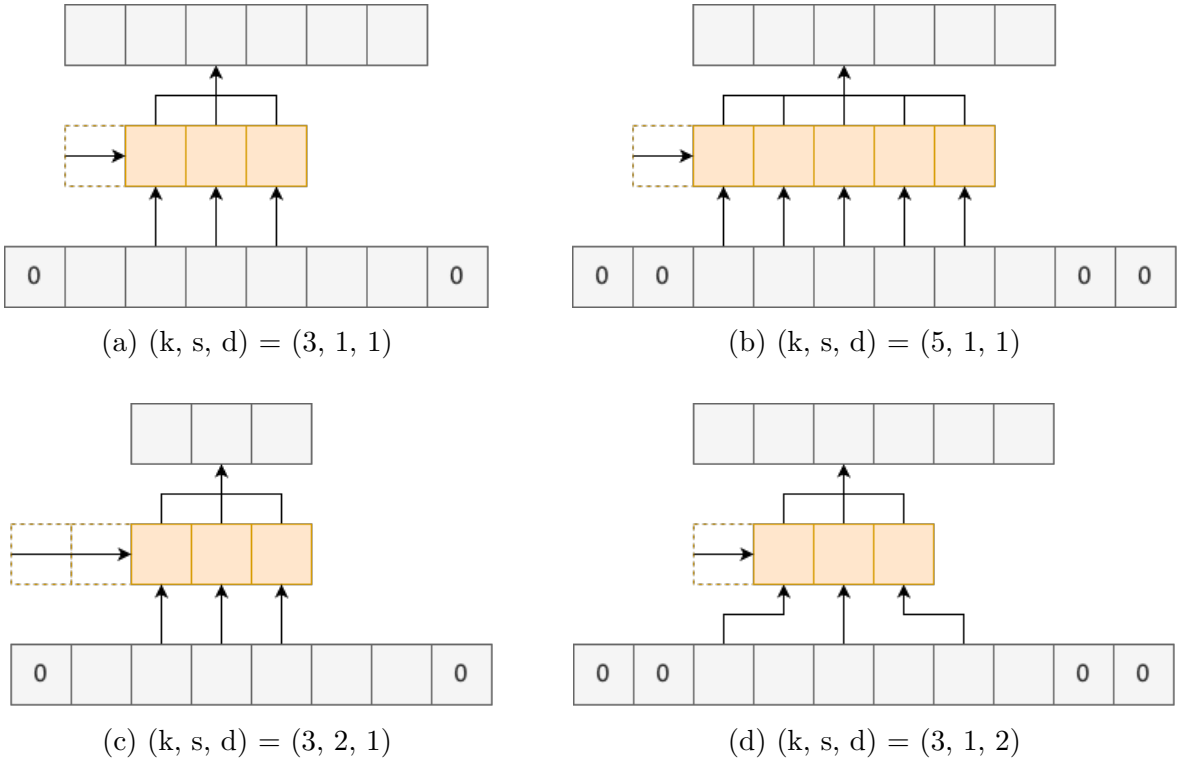
3.1.2　畳み込み層

畳み込み層は、入力に対して畳み込み演算を行う層である。一次元畳み込み層について、入力 $\boldsymbol{x} \in \mathbb{R}^{C_{in} \times T_{in}}$ に対し、出力 $\boldsymbol{y} \in \mathbb{R}^{C_{out} \times T_{out}}$ は、$y_k[i]$ を出力テンソルの k 番目のチャネルの i 番目の成分とすると、

$$y_k[i] = b_k + \sum_{c=0}^{C_{in}-1} \sum_{m=0}^{M-1} x_c \left[i - \left\lfloor \frac{M}{2} \right\rfloor + m \right] w_{k,c}[m] \tag{3.2}$$

で計算される。ここで、$x_c \left[i - \left\lfloor \frac{M}{2} \right\rfloor + m \right]$ が入力テンソルの c 番目のチャネルの $\left[i - \left\lfloor \frac{M}{2} \right\rfloor + m \right]$ 番目の成分、$w_{k,c}[m]$ が出力チャネル k と入力チャネル c に対応するカーネルの m 番目の成分、M がカーネルサイズである。上式より、一次元畳み込み層の i 番目の出力は、i 番目の入力を中心とし、カーネルサイズの範囲分が考慮されて得られる値だと解釈できる。一次元畳み込みは、自然言語や音声といった一次元系列に対する特徴抽出のために用いられることが多い。

これに加えて、カーネルを二次元配列とすれば二次元畳み込み層、三次元配列とすれば三次元畳み込み層となる。二次元畳み込み層は主に画像に用いられることが多く、三次元畳み込みは動画に用いられることが多い。



(a) (k, s, d) = (3, 1, 1)　(b) (k, s, d) = (5, 1, 1)

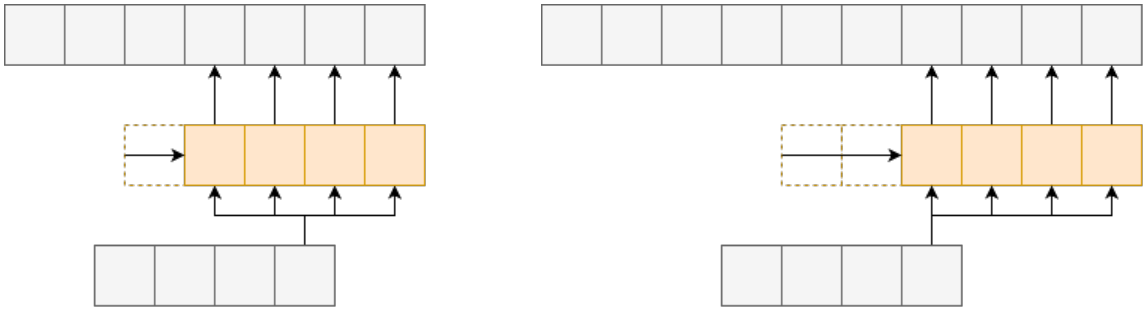
(c) (k, s, d) = (3, 2, 1)　(d) (k, s, d) = (3, 1, 2)

図 3.1: ある入出力チャネル間における一次元畳み込み層の様子。k はカーネルサイズ、s はストライド、d はダイレーションを表し、図中の 0 はパディング部。

畳み込み層における主要なパラメータは三つある。一つ目は、カーネルサイズである。この値が、畳み込み演算におけるフィルタのサイズを決定しており、考慮できる入力特徴量の範囲が定まる。二つ目は、ストライドである。この値はフィルタのシフト幅にあたり、2 以上の値を設定すると、出力特徴量のサイズは入力特徴量のサイズよりも小さくなる。実際、特徴量をダウンサンプリングしたい場合には、2 以上の値が用いられる。三つ目は、ダイレーションである。これはフィルタ内の要素間の距離を表し、その値を 2 以上とすれば、入力特徴量の飛び飛びの値を考慮した畳み込み演算が行われる。ダイレーションを大きくすることによって、カーネルサイズが同じでも考慮できる範囲が広がることが特徴であり、系列長の長大な音声波形を扱う場合などに用いられる。また、出力の系列長を入力に対して整数倍に保つためには、上記のパラメータに対して適切なパディング長を指定する必要がある。例えば、カーネルサイズを 3、ストライドとダイレーションを 1 とした場合には、入力の両端に 1 ずつゼロパディングすることで入出力の系列長が保たれる。図 3.1 に、ある入出力チャネル間における一次元畳み込み層の様子を示す。

3.1.3　転置畳み込み層

転置畳み込み層は、畳み込み層の逆の演算を行うような層である。図 3.2 に、ある入出力チャネル間における一次元畳み込み層の様子を示す。転置畳み込み層では、i 番目の入力とカーネ



(a) (k, s) = (4, 1)　　(b) (k, s) = (4, 2)

図 3.2: ある入出力チャネル間における一次元転置畳み込み層の様子。k はカーネルサイズ、s はストライド。

ルの積を計算し、その結果を i 番目から $i+M-1$ 番目までの出力とする。ここで M はカーネルサイズを表す。また、i 番目の入力から計算された出力が、$i-1$ 番目までの入力によってすでに得られている出力とオーバーラップする場合、これらは加算される。図 3.2a は、カーネルサイズを 4、ストライドを 1 とした場合の様子である。転置畳み込み層は特に、入力をアップサンプリングしたい場合に用いられることが多い。その例を図 3.2b に示す。ここでは、カーネルサイズを 4、ストライドを 2 としており、入力系列長が 4 であるのに対して、出力系列長が 10 まで拡大されていることがわかる。ここで、出力系列長を入力系列長の整数倍にするためには、パディングの値を適切に設定する必要がある。転置畳み込み層におけるパディングは、出力の両端を何個落とすかを指定するパラメータである。上述の例においては、パディングを 1 とすることで、出力系列長を入力系列長の 2 倍である 8 に調整することができる。

3.1.4　活性化関数

活性化関数は、ニューラルネットワークの各層の出力に非線形性を導入するために用いられる関数である。これにより、ネットワークは単純な線形変換だけでは表現できない複雑な入出力の関係を学習可能になる。以下、代表的な活性化関数を述べる。また、活性化関数とその一階導関数のグラフを図 3.3 に示す。

一つ目は、シグモイド関数である。シグモイド関数は

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{3.3}$$

で与えられ、その一階導関数は

$$\frac{d}{dx}\sigma(x) = \frac{\exp(-x)}{(1+\exp(-x))^2} \tag{3.4}$$

となる。図 3.3b より、シグモイド関数の一階導関数の最大値は $x=0$ における 0.25 であり、入力が 0 から離れるほど微分係数が小さくなることが分かる。DNN の学習では、損失関数の重みに対するの勾配を逆伝播する必要があり、これは合成関数の微分における連鎖律に従って行わ

れる。シグモイド関数は取り得る微分係数の値が小さく、特に層数が深い DNN においては勾配が小さくスケーリングされていくことで、浅い層の重みに対する勾配のノルムが小さくなりすぎる可能性がある。この問題は、勾配消失と呼ばれる。

二つ目は、tanh 関数である。tanh 関数は

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{3.5}$$

で与えられ、その一階導関数は

$$\frac{d}{dx}\tanh(x) = \frac{4}{(\exp(x) + \exp(-x))^2} \tag{3.6}$$

$$= \frac{1}{\cosh(x)^2} \tag{3.7}$$

となる。シグモイド関数と比較すると、シグモイド関数の値域が $[0, 1]$ であったのに対し、tanh 関数の値域は $[-1, 1]$ となっている。シグモイド関数や tanh 関数を実際に活性化関数として利用するニューラルネットワークとして、後述する LSTM や GRU が挙げられる。

三つ目は、ReLU である。ReLU は

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3.8}$$

で与えられ、その一階導関数は

$$\frac{d}{dx}\mathrm{ReLU}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x <= 0 \end{cases} \tag{3.9}$$

となる。ReLU は入力が 0 以上であれば恒等写像となり、0 未満であれば 0 に写す。一階導関数は 0 あるいは 1 のみを取り、特に入力が正の値であれば常に微分係数は 1 となることから、シグモイド関数よりも勾配消失問題が起こりづらくなっている。一方、ReLU は 0 以下の入力に対して微分係数が 0 となるため、勾配も 0 となる。従って、入力が負の値に偏った場合には勾配が 0 になり続け、重みの更新が進まなくなる可能性がある。この問題を、Dying ReLU 問題と呼ぶ。

四つ目は、LeakyReLU である。LeakyReLU は

$$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{if } x <= 0 \end{cases} \tag{3.10}$$

で与えられ、その一階導関数は

$$\frac{d}{dx}\mathrm{LeakyReLU}(x) = \begin{cases} 1 & \text{if } x > 0 \\ a & \text{if } x <= 0 \end{cases} \tag{3.11}$$

となる。ここで、a は negative slope と呼ばれるパラメータである。ReLU と比較すると、0 以下の入力に対しても 0 でない値を取り、また微分係数も 0 にならない点が異なっている。この工夫により、勾配が 0 になり続ける恐れがなくなるため、前述した Dying ReLU 問題に対処することが可能である。具体的な利用ケースとしては、敵対的生成ネットワーク（Generative adversarial network; GAN）における Generator や Discriminator が挙げられる。

五つ目は、PReLU である。これは四つ目に述べた LeakyReLU と非常に似た活性化関数であるが、LeakyReLU の negative slope を学習可能なパラメータに変更したという点で異なっている。

六つ目は、GELU である。GELU は

$$\mathrm{GELU}(x) = x\Phi(x) \tag{3.12}$$

で与えられる。ここで、$\Phi(x)$ は標準正規分布の累積分布関数である。GELU の一階導関数は、

$$\frac{d}{dx}\mathrm{GELU}(x) = \Phi(x) + \frac{x}{\sqrt{2\pi}}\exp(-\frac{x^2}{2}) \tag{3.13}$$

となる。GELU は入力に対して標準正規分布に基づく確率値を利用したスケーリングを行う活性化関数であり、後述する Transformer など大規模なモデルにおいて採用される場合がある。

3.1.5 再帰型ニューラルネットワーク

再帰型ニューラルネットワーク（Recurrent Neural Network; RNN）は、自身の過去の出力を保持し、それをループさせる再帰的な構造を持ったネットワークである。

近年よく用いられる RNN として、長・短期記憶（Long Short-Time Memory; LSTM） [28] がある。LSTM では入力ゲート、忘却ゲート、出力ゲートの 3 つを持ち、これらゲートによってネットワーク内部の情報の取捨選択を行うことで、長い系列データからの学習を可能にした。LSTM のネットワーク内部で行われる計算を以下に示す。

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_f) \tag{3.14}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_i) \tag{3.15}$$

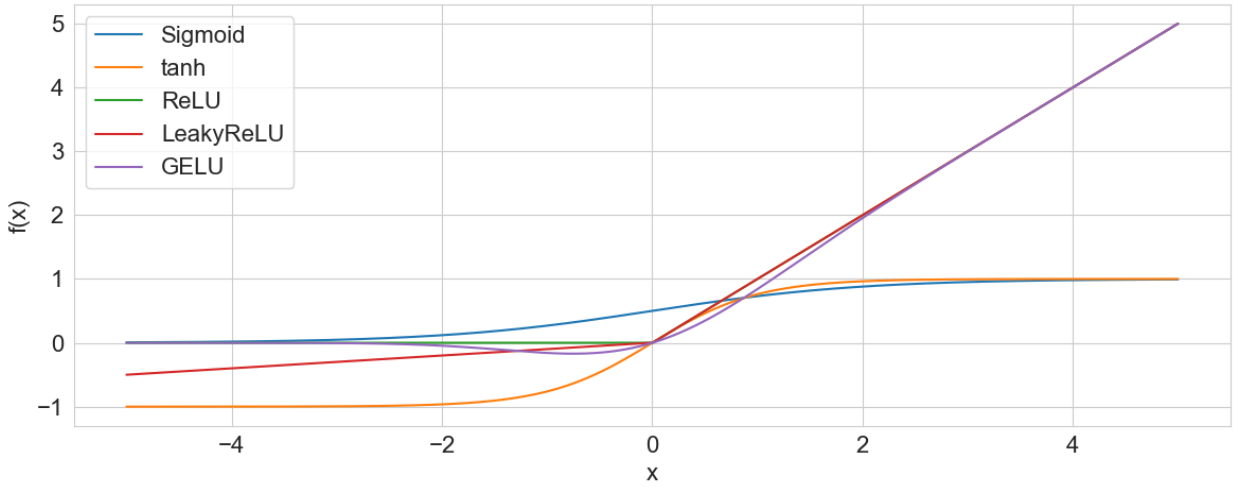
$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_h[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_h) \tag{3.16}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{3.17}$$

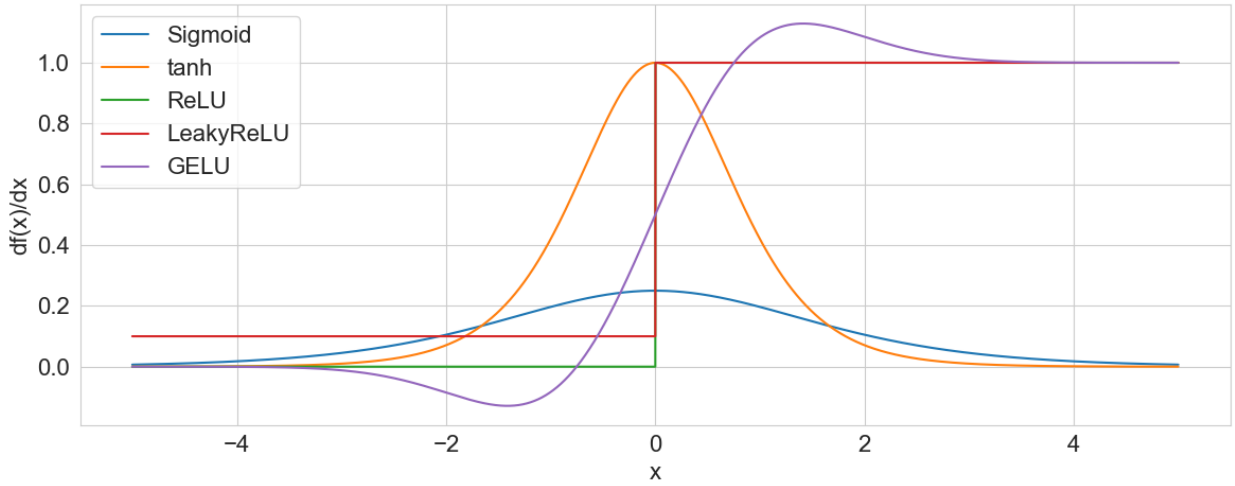
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_o) \tag{3.18}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{3.19}$$

ここで、$\boldsymbol{x}_t$ は時刻 t の入力、$\boldsymbol{f}_t$ は忘却ゲートの出力、$\boldsymbol{i}_t$ は入力ゲートの出力、$\boldsymbol{c}_t$ が時刻 t におけるセルの状態、$\boldsymbol{o}_t$ が出力ゲートの出力、$\boldsymbol{h}_t$ が時刻 t における隠れ状態を表し、それぞれベクトルである。また、$\boldsymbol{W}$、$\boldsymbol{b}$ はそれぞれ学習可能な重み行列、バイアスベクトルである。忘却ゲートの出力 $\boldsymbol{f}_t$ は、前時刻のセル状態 $\boldsymbol{c}_{t-1}$ との要素積に用いられ、これによってネットワーク内



(a) 活性化関数



(b) 活性化関数の一階導関数

図 3.3: 活性化関数の例

に保持されていた長期記憶に対して忘却するべき情報を与え、取捨選択を行う。入力ゲートの出力 $\boldsymbol{i}_t$ は時刻 t のセル状態の候補となる $\tilde{\boldsymbol{c}}_t$ との要素積に用いられ、これによって新たにネットワークが長期記憶するべき情報が選択され、セル状態 $\boldsymbol{c}_t$ として保持される。出力ゲートの出力 $\boldsymbol{o}_t$ は時刻 t のセル状態 $\boldsymbol{c}_t$ に tanh を適用した値との要素積に用いられ、これによって時刻 t において出力するべき情報を選択する。最後に、隠れ状態 $\boldsymbol{h}_t$ は、時刻 t の出力と tanh を適用したセル状態 $\boldsymbol{c}_t$ から決定される。このように、$\boldsymbol{h}_t$ 及び $\boldsymbol{c}_t$ を入力に対して適応的に変化させながら情報を取捨選択することで、長い系列長のデータに対しての学習を可能にした。

また、LSTM では 3 つのゲートを必要とするが、ゲートを 2 つに減らすことでネットワークの軽量化を図ったネットワークとして、ゲート付き回帰型ユニット（Gated Recurrent Unit; GRU）[29] がある。GRU ではリセットゲートと更新ゲートの 2 つのゲートを用いて隠れ状態 $\boldsymbol{h}_t$ を再

帰的に更新する。また、LSTM と違ってセル状態 $\boldsymbol{c}_t$ を省いており、よりシンプルな構造となっている。GRU のネットワーク内部で行われる計算を以下に示す。

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_z) \tag{3.20}$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_r) \tag{3.21}$$

$$\tilde{\boldsymbol{h}_t} = \tanh(\boldsymbol{W}_h[\boldsymbol{x}_t, \boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}] + \boldsymbol{b}_h) \tag{3.22}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}_t} \tag{3.23}$$

ここで、$\boldsymbol{x}_t$ が時刻 t における入力、$\boldsymbol{z}_t$ が更新ゲートの出力、$\boldsymbol{r}_t$ がリセットゲートの出力、$\boldsymbol{h}_t$ が時刻 t における隠れ状態を表し、それぞれベクトルである。また、$\boldsymbol{W}$ は学習可能な重み行列、$\boldsymbol{b}$ はバイアスベクトルである。更新ゲートの出力 $\boldsymbol{z}_t$ はネットワーク内で新たに記憶するべき情報の割合を決定する役割を果たす。一方、リセットゲートの出力 $\boldsymbol{r}_t$ は前時刻の隠れ状態 $\boldsymbol{h}_{t-1}$ との要素積に用いられ、これによって忘却するべき情報が選択される。リセットゲートの出力によって処理された内部表現から、時刻 t の隠れ状態の候補となる $\tilde{\boldsymbol{h}_t}$ が計算される。最後に、前時刻の隠れ状態 $\boldsymbol{h}_{t-1}$ と隠れ状態の候補 $\tilde{\boldsymbol{h}_t}$ に対して更新ゲートの出力 $\boldsymbol{z}_t$ を用いた重み付け和を計算することで、隠れ状態 $\boldsymbol{h}_t$ を決定する。

3.1.6 Transformer

Transformer [30] は、自己注意機構（Self-Attention）を用いて、入力系列全体に渡る依存関係を捉えることができるニューラルネットワークである。特に、再帰的な計算を必要とする RNN と比較して、Transformer は並列計算のみ行うため、GPU による計算の高速化が可能である。Transformer の内部構造について説明するため、入力系列 $\boldsymbol{X} \in \mathbb{R}^{T \times d_{model}}$ をとる。ここで、T は入力系列の系列長、d_{model} は入力系列の次元である。また、Transformer 層の構造を図 3.4 に示す。

まず、Self-Attetion について、これは以下の二つの要素からなる。

1. クエリ、キー、バリューの計算
2. Attention スコアの計算

まず、クエリ、キー、バリューの計算は、それぞれを $\boldsymbol{Q} \in \mathbb{R}^{T \times d_k}$、$\boldsymbol{K} \in \mathbb{R}^{T \times d_k}$、$\boldsymbol{V} \in \mathbb{R}^{T \times d_v}$ とおくと、

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \tag{3.24}$$

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K \tag{3.25}$$

$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V \tag{3.26}$$

で与えられる。ここで、d_k はキーとクエリの次元、d_v はバリューの次元、$\boldsymbol{W_Q} \in \mathbb{R}^{d_{model} \times d_k}$ はクエリに対する重み行列、$\boldsymbol{W_K} \in \mathbb{R}^{d_{model} \times d_k}$ はキーに対する重み行列、$\boldsymbol{W_V} \in \mathbb{R}^{d_{model} \times d_v}$ は

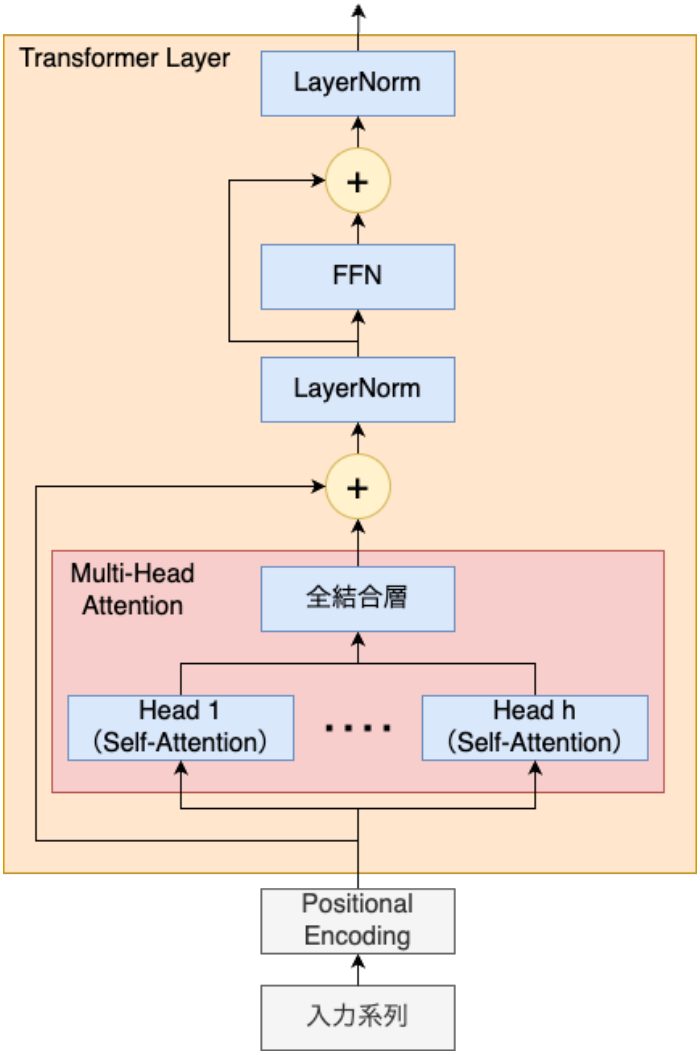


図 3.4: Transformer 層の構造

バリューに対する重み行列である。次に、ここで得られたクエリ、キー、バリューを利用した Attention スコアの計算は、

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{3.27}$$

で与えられる。式中の $\boldsymbol{Q}\boldsymbol{K}^\top \in \mathbb{R}^{T\times T}$ について、この行列の (i,j) 成分 $(\boldsymbol{Q}\boldsymbol{K}^\top)_{i,j}$ は、クエリベクトル $Q_i \in \mathbb{R}^{d_k}$ と、キーベクトル $K_j \in \mathbb{R}^{d_k}$ の内積であり、Q_i と K_i の関連度を表していると解釈できる。また、softmax 関数は列方向に適用されるため、アテンション重み $\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)$ の (i,j) 成分 $\left(\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\right)_{i,j}$ は、

$$\left(\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\right)_{i,j} = \frac{\exp\left(\frac{(\boldsymbol{Q}\boldsymbol{K}^\top)_{i,j}}{\sqrt{d_k}}\right)}{\sum_{j=1}^{T}\exp\left(\frac{(\boldsymbol{Q}\boldsymbol{K}^\top)_{i,j}}{\sqrt{d_k}}\right)} \tag{3.28}$$

となる。softmax 関数を適用することでアテンション重みの行ベクトルは確率値として扱うこ

とができるようになり、これは i 番目のクエリからのキーの全要素に対する注意度が、確率分布として表されていると解釈できる。最後にアテンション重みをバリュー $\boldsymbol{V}$ にかけることにより、各クエリからの注意度をもとにバリューベクトル $\boldsymbol{V}_j$ の重み付け和が計算され、出力が得られる。

Transformer では、Self-Attention をただ行うだけではなく、これを複数のヘッドで並列に計算し、各ヘッドの出力を結合して最終出力を得る Multi-Head Attention が行われる。ヘッド数を h とすると、各ヘッドにおけるクエリ、キー、バリューの計算は、

$$\boldsymbol{Q}^i = \boldsymbol{X}\boldsymbol{W}_Q^i \tag{3.29}$$

$$\boldsymbol{K}^i = \boldsymbol{X}\boldsymbol{W}_K^i \tag{3.30}$$

$$\boldsymbol{V}^i = \boldsymbol{X}\boldsymbol{W}_V^i \tag{3.31}$$

で与えられる。ここで、$\boldsymbol{W}_{\boldsymbol{Q}}^i \in \mathbb{R}^{d_{model} \times \frac{d_k}{h}}$ は i 番目のヘッドのクエリに対する重み行列、$\boldsymbol{W}_{\boldsymbol{K}}^i \in \mathbb{R}^{d_{model} \times \frac{d_k}{h}}$ は i 番目のヘッドのキーに対する重み行列、$\boldsymbol{W}_{\boldsymbol{V}}^i \in \mathbb{R}^{d_{model} \times \frac{d_v}{h}}$ は i 番目のヘッドのバリューに対する重み行列である。よって、i 番目のヘッドのクエリ、キー、バリューの形状はそれぞれ、$\boldsymbol{Q}^i \in \mathbb{R}^{T \times \frac{d_k}{h}}$、$\boldsymbol{K}^i \in \mathbb{R}^{T \times \frac{d_k}{h}}$、$\boldsymbol{V}^i \in \mathbb{R}^{T \times \frac{d_v}{h}}$ となる。i 番目のヘッドにおける Attention スコアの計算は

$$\text{Attention}(\boldsymbol{Q}^i, \boldsymbol{K}^i, \boldsymbol{V}^i) = \text{softmax}\left(\frac{\boldsymbol{Q}^i(\boldsymbol{K}^i)^\top}{\sqrt{d_k}/h}\right)\boldsymbol{V}^i \tag{3.32}$$

で行われる。その後、すべてのヘッドからの Attention スコアを結合し、さらに線形変換を適用することで Multi-Head Attetion の出力が得られる。これは、

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Concat}(\text{head}^1, \ldots, \text{head}^h)\boldsymbol{W}_o \tag{3.33}$$

と表される。ここで、$\text{head}^i \in \mathbb{R}^{T \times \frac{d_v}{h}}$ が各ヘッドから出力された Attention スコア、$\text{Concat}(\text{head}^1, \ldots, \text{head}^h) \in \mathbb{R}^{T \times d_v}$ が全ヘッドの Attention スコアを結合した特徴量、$\boldsymbol{W}_o \in \mathbb{R}^{d_v \times d_{model}}$ が最終的な線形変換に用いられる重み行列である。ヘッドを分割することで複数パターンのアテンションが可能になり、これが入出力間の複雑な関係性を考慮するのに役立っていると考えられる。Multi-Head Attention 後には、残差結合と後述するレイヤー正規化を適用する。これは、出力系列を $\boldsymbol{Y} \in \mathbb{R}^{T \times d_{model}}$ とすると、

$$\boldsymbol{Y} = \text{LayerNorm}\left(\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) + \boldsymbol{X}\right) \tag{3.34}$$

で与えられる。

Multi-Head Attention によって系列全体の依存関係を考慮した後は、各フレームごとに独立して全結合層を適用する。これは、

$$\text{FFN}(\boldsymbol{Y}) = \text{ReLU}(\boldsymbol{Y}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{3.35}$$

で与えられる。ここで、$\boldsymbol{W}_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$、$\boldsymbol{W}_2 \in \mathbb{R}^{d_1 \times d_{model}}$ は学習可能な重み行列、$\boldsymbol{b}_1 \in \mathbb{R}^{d_{ff}}$、$\boldsymbol{b}_2 \in \mathbb{R}^{d_{model}}$ はバイアスベクトルである。d_{ff} は d_{model} の 4 倍とされることが多い。全結合層適用後は、Multi-Head Attention 後と同様に、残差結合とレイヤー正規化を適用する。

上述した Multi-Head Attention とフレームごとに適用される全結合層を合わせて、Transformer 層と呼ぶ。実際には、Transformer 層を多層積み重ねて用いることが多い。

最後に、Transformer では RNN と違い、並列計算によって系列全体を一度に処理することが可能であるが、それと引き換えに入力の順序情報を考慮することができなくなる。この問題に対し、入力系列に対して位置情報を与えるために行われるのが、Positional Encoding である。Positional Encoding は sin 関数と cos 関数に基づいて計算される値であり、

$$\mathrm{PE}_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{3.36}$$

$$\mathrm{PE}_{pos,2i+1} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{3.37}$$

で与えられる。ここで、pos は入力系列の位置を表し、i は入力系列の次元を表す。

3.1.7 プーリング層

3.2 学習方法

3.2.1 損失関数

損失関数は、DNN によって推定された結果と、正解値との間の誤差を求める関数のことであり、扱う問題によって様々である。例えば、回帰問題において用いられる関数の一つに、MAE（Mean Absolute Error） Loss がある。推定対象を $\boldsymbol{Y} \in \mathbb{R}^{T\times D}$、DNN による推定結果を $\hat{\boldsymbol{Y}} \in \mathbb{R}^{T\times D}$ とすると、MAE Loss は

$$L_{MAE}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \frac{1}{TD}\sum_{t=1}^{T}\sum_{d=1}^{D}|\boldsymbol{Y}_{t,d} - \hat{\boldsymbol{Y}}_{t,d}| \tag{3.38}$$

で与えられる。ここで、T が系列長、D が次元である。

一方、分類問題において用いられる関数の一つに、Cross Entropy Loss がある。C クラス分類の問題について、推定対象を $\boldsymbol{y} \in \mathbb{R}^{C}$、DNN による推定結果を $\hat{\boldsymbol{y}} \in \mathbb{R}^{C}$ とすると、Cross Entropy Loss は

$$L_{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{c=1}^{C}\boldsymbol{y}_c \log(\hat{\boldsymbol{y}}_c) \tag{3.39}$$

で与えられる。また、推定対象がテキストなどの系列長の次元を持ったテンソルとなる場合、推定対象を $\boldsymbol{Y} \in \mathbb{R}^{T\times C}$、DNN による推定結果を $\hat{\boldsymbol{Y}} \in \mathbb{R}^{T\times C}$ とすると、Cross Entropy Loss は

$$L_{CE}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{c=1}^{C}\boldsymbol{Y}_{t,c} \log(\hat{\boldsymbol{Y}}_{t,c}) \tag{3.40}$$

で与えられる。すなわち、各トークンごとの損失の平均値となる。実際の分類問題において推定対象は、正解となるクラスのみを 1、それ以外を 0 とした One-hot ベクトルとされることが

多い。すなわち、推定対象の One-hot ベクトルを $\boldsymbol{y} \in \mathbb{R}^C$ とすると、この各クラス c ごとの値 $\boldsymbol{y}_c$ は、その正解クラスを $c_{correct} \in \{1, \ldots, C\}$ とすると、

$$\boldsymbol{y}_c = \begin{cases} 1 & \text{if } c = c_{correct} \\ 0 & \text{if } c \neq c_{correct} \end{cases} \tag{3.41}$$

で与えられる。

3.2.2 勾配降下法

勾配降下法は、損失関数を最小化する DNN の重みとバイアスを得るための最適化手法である。モデルの重みとバイアスをまとめて $\boldsymbol{\theta}$、損失関数を $L(\boldsymbol{\theta})$ とすると、勾配降下法による $\boldsymbol{\theta}$ の更新は

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla L(\boldsymbol{\theta}) \tag{3.42}$$

で表される。ここで、η は学習率と呼ばれるパラメータであり、$\boldsymbol{\theta}$ の一回の更新による変化量を制御する役割を果たす。上式より、勾配降下法は損失関数の $\boldsymbol{\theta}$ についての勾配を利用し、損失関数の値をより小さくする方向へ $\boldsymbol{\theta}$ を更新する手続きを繰り返すことによって、損失関数の値を最小化する DNN を得ようとする最適化手法だと解釈できる。

勾配降下法には三種類の方法がある。一つ目は、バッチ勾配降下法である。これは、$\boldsymbol{\theta}$ の更新一回につき、全学習データを利用して損失関数の勾配を計算する方法である。データ一つ一つのノイズに影響を受けづらく安定した更新が期待できるが、一回の更新にかかる計算コストが大きいというデメリットがある。二つ目は、確率的勾配降下法である。これは、$\boldsymbol{\theta}$ の更新一回につき、ある一つの学習データのみによって損失関数の勾配を計算する方法である。バッチ勾配降下法と比較して一回の更新にかかる計算コストは低くなるが、各データのノイズによる影響が大きくなることで学習が不安定になる可能性がある。三つ目は、ミニバッチ勾配降下法である。これは、$\boldsymbol{\theta}$ の更新一回につき、学習データの中から指定した個数のデータをサンプリングしてミニバッチを作成し、これを利用して損失関数の勾配を計算する方法である。ここで、サンプリングするデータの個数はバッチサイズと呼ばれる。ミニバッチ勾配降下法は、バッチ勾配降下法と確率的勾配降下法の間をとったような方法であり、バッチサイズによって一回の更新に利用するデータの数を調整可能であることが両者にないメリットである。実際、DNN の学習においてはミニバッチ勾配降下法が用いられることが多い。

3.2.3 誤差逆伝播法

前述した勾配降下法による DNN の最適化のためには、各重み及びバイアスについての損失関数の勾配を計算する必要があり、これは、微分の連鎖律に基づいて計算することが可能である。ここで、誤差逆伝播法は、各重み及びバイアスについての損失関数の勾配を、出力から入力へと DNN を遡る方向に再帰的に計算するアルゴリズムである。ここでは例として、全結合

層と活性化関数のみからなる L 層の DNN を構築し、ミニバッチ勾配降下法によって最適化する場面を考え、勾配逆伝播法の計算方法を説明する。

まず、$w_{i,j}^k$ を $k-1$ 層目の i 番目のノードから k 層目の j 番目のノードに割り当てられた重み、b_i^k を k 層目の i 番目のノードに割り当てられたバイアスとすると、k 層目の i 番目のノードにおける出力 a_i^k は、

$$a_i^k = b_i^k + \sum_{j=1}^{N_{k-1}} w_{j,i}^k o_j^{k-1} \tag{3.43}$$

で与えられる。ここで、N_{k-1} は $k-1$ 層目の全結合層のチャンネル数、o_j^{k-1} は $k-1$ 層目の j 番目のノードにおける出力 a_j^{k-1} に活性化関数 f を適用した結果を表す。すなわち、

$$o_j^{k-1} = f(a_j^{k-1}) \tag{3.44}$$

で与えられる。ここで、式 (3.43) に対し、$w_{0,i}^k = b_i^k$ とし、$o_0^{k-1} = 1$ とすれば、

$$a_i^k = b_i^k + \sum_{j=1}^{N_{k-1}} w_{j,i}^k o_j^{k-1} = \sum_{j=0}^{N_{k-1}} w_{j,i}^k o_j^{k-1} \tag{3.45}$$

と整理できる。

次に、DNN の学習用データを $\{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$ とおく。ここで、N はデータの数、$\boldsymbol{x}_n \in \mathbb{R}^{d_x}$ は DNN への入力ベクトル、$\boldsymbol{y}_n \in \mathbb{R}^{d_y}$ は x_n に対する DNN の推定対象を表す。さらに、DNN の推定結果を $\hat{\boldsymbol{y}}_n \in \mathbb{R}^{d_y}$ とし、損失関数を $L(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n; \boldsymbol{\theta})$ とする。ここで、$\boldsymbol{\theta}$ は DNN の全ての重みとバイアスをまとめて表した変数である。この時、ミニバッチ勾配降下法におけるバッチサイズを N_{batch} とすると、あるミニバッチにおける損失関数の重み $w_{i,j}^k$ についての勾配は、

$$\frac{\partial L}{\partial w_{i,j}^k} = \frac{\partial}{\partial w_{i,j}^k} \left(\frac{1}{N_{batch}} \sum_{n=1}^{N_{batch}} L(\boldsymbol{y_n}, \hat{\boldsymbol{y}}_n; \boldsymbol{\theta}) \right) \tag{3.46}$$

$$= \frac{1}{N_{batch}} \sum_{n=1}^{N_{batch}} \frac{\partial}{\partial w_{i,j}^k} L(\boldsymbol{y_n}, \hat{\boldsymbol{y}}_n; \boldsymbol{\theta}) \tag{3.47}$$

$$= \frac{1}{N_{batch}} \sum_{n=1}^{N_{batch}} \frac{\partial L_n}{\partial w_{i,j}^k} \tag{3.48}$$

となる。ここで、式 (3.48) では $L(\boldsymbol{y_n}, \hat{\boldsymbol{y}}_n; \boldsymbol{\theta}) = L_n$ とおいた。式 (3.48) より、ミニバッチ全体における損失関数の勾配は、サンプルごとに計算される勾配の平均値であることがわかる。ここで、$\partial L_n / \partial w_{i,j}^k$ は、

$$\frac{\partial L_n}{\partial w_{i,j}^k} = \frac{\partial L_n}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{i,j}^k} \tag{3.49}$$

となる。ここで、式 (3.45) より

$$\frac{\partial a_j^k}{\partial w_{i,j}^k} = \frac{\partial}{\partial w_{i,j}^k} \left(\sum_{l=0}^{N_{k-1}} w_{l,j}^k o_l^{k-1} \right) = o_i^{k-1} \tag{3.50}$$

であり、

$$\delta_j^k = \frac{\partial L_n}{\partial a_j^k} \tag{3.51}$$

とおけば、式 (3.49) は

$$\frac{\partial L_n}{\partial w_{i,j}^k} = \delta_j^k o_i^{k-1} \tag{3.52}$$

と書ける。ここで、まず出力層の重み $w_{i,j}^L$ についての L_n の勾配は、

$$\begin{aligned}
\frac{\partial L_n}{\partial w_{i,j}^L} &= \delta_j^L o_i^{L-1} && (3.53)\\
&= \frac{\partial L_n}{\partial a_j^L} o_i^{L-1} && (3.54)\\
&= \frac{\partial}{\partial a_j^L} L(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n; \boldsymbol{\theta}) \cdot o_i^{L-1} && (3.55)\\
&= \frac{\partial}{\partial a_j^L} L(\boldsymbol{y}_n, f(\boldsymbol{a}^L)_n; \boldsymbol{\theta}) \cdot o_i^{L-1} && (3.56)\\
&= \frac{\partial}{\partial f(\boldsymbol{a})_n} L(\boldsymbol{y}_n, f(\boldsymbol{a}^L)_n; \boldsymbol{\theta}) \cdot \frac{\partial}{\partial a_j^L} f(\boldsymbol{a}^L)_n \cdot o_i^{L-1} && (3.57)
\end{aligned}$$

で計算できる。次に、隠れ層の重み $w_{i,j}^k$ $(1 \leq k < L)$ についての L_n の勾配は、

$$\begin{aligned}
\frac{\partial L_n}{\partial w_{i,j}^k} &= \delta_j^k o_i^{k-1} && (3.58)\\
&= \frac{\partial L_n}{\partial a_j^k} o_i^{k-1} && (3.59)\\
&= o_i^{k-1} \sum_{l=0}^{N_{k+1}} \frac{\partial L_n}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k} && (3.60)\\
&= o_i^{k-1} \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_j^k} && (3.61)\\
&= o_i^{k-1} \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} \frac{\partial}{\partial a_j^k} \left(\sum_{m=0}^{N_k} w_{m,l}^{k+1} o_m^k \right) && (3.62)\\
&= o_i^{k-1} \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} \frac{\partial}{\partial a_j^k} \left(\sum_{m=0}^{N_k} w_{m,l}^{k+1} f(a_m^k) \right) && (3.63)\\
&= o_i^{k-1} \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} w_{j,l}^{k+1} f'(a_j^k) && (3.64)\\
&= o_i^{k-1} f'(a_j^k) \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} w_{j,l}^{k+1} && (3.65)
\end{aligned}$$

で計算できる。以上より、出力層の重みについての損失関数の勾配は直ちに計算可能であるのに対して、隠れ層 k の重みについての損失関数の勾配は、自身の次の層 $k+1$ についての δ_l^{k+1}

に依存していることがわかる。従って、出力から入力へと遡るように勾配を計算していき、以前に求めた計算結果をメモしておけば、効率よく勾配を計算することが可能だとわかる。

3.2.4 最適化手法

3.2.2 節において、DNN の重みが勾配降下法によって最適化されること、3.2.3 節において、勾配降下法に必要な、DNN の重みについての損失関数の勾配を計算するアルゴリズムである、誤差逆伝播法について述べた。ここで、勾配降下法では学習率 η が一回の更新による重みの変化量を制御するパラメータであった。学習率が大きいほど一度の更新で大きく重みが変化するため、その分学習が早くなる可能性がある。一方、損失関数の最小値に近づいていくと、一回の重みの更新幅が大きすぎることから、局所最適解への収束が困難になる。これに対して、学習率が小さい場合には、更新幅が小さい分特定の局所最適解に収束しやすくなるが、学習は遅くなる。加えて、特定の局所最適解に収束しやすくなる一方で、実行可能領域にはより損失関数の値を小さくできる解が存在していた可能性もあり、それに劣る解から抜け出せなくなる可能性が高まるとも言える。従って、学習率を適切な値に設定することは、DNN の最適化において非常に重要だと言える。これに対し、近年よく用いられる最適化手法として Adam [31] がある。Adam の計算過程をアルゴリズム 1 に示す。ここで、$\boldsymbol{g}_t$ は重み $\boldsymbol{\theta}$ についての損失関数 $L(\boldsymbol{\theta})$ の勾配、$\boldsymbol{m}_t$ が勾配の移動平均、$\boldsymbol{v}_t$ が勾配の二乗値の移動平均であり、$\hat{\boldsymbol{m}}_t$ および $\hat{\boldsymbol{v}}_t$ は移動平均 $\boldsymbol{m}_t$ と $\boldsymbol{v}_t$ の初期値が 0 であることに起因するバイアスを防ぐための計算を行った結果である。また、β_1 および β_2 は、移動平均 $\boldsymbol{m}_t$ および $\boldsymbol{v}_t$ を求める際の重み付け係数となるパラメータである。Adam では重みの更新の際、勾配 $\boldsymbol{g}_t$ そのものではなく、その移動平均である $\boldsymbol{m}_t$ を利用することで勾配の振動を抑制する。また、勾配の二乗値の移動平均 $\boldsymbol{v}_t$ によって学習率 η を割ることにより、勾配が振動している際に学習率が適応的に小さくなる。こういった工夫により、重みの更新の程度を学習の進行に伴って変化させることで、重みの更新をスムーズに行うことを可能にしている。また、λ は後述する正則化手法の一つである、重み減衰の程度を調整するパラメータである。また、Adam では正則化として重み減衰を、勾配にパラメータの値を加算するような形で行っている。これは、λ が 0 より大きい値であれば勾配が本来の値よりも大きくなることになり、Adam の強みである勾配の移動平均を利用した適応的な学習率の設定に悪影響を与える可能性がある。これに対し、AdamW [32] では、重み減衰を勾配に加算する形で行うのではなく、重み自体に加算するように変更した。これにより、λ が 0 より大きい値であっても勾配の値には影響を与えないため、Adam が持つ適応的な学習率調整の利点を損なうことなく、重み減衰による正則化を適用できるようにした。AdamW の計算過程をアルゴリズム 2 に示す。

3.2.5 学習率のスケジューリング

3.2.4 節では、DNN の学習における学習率の重要性および、学習過程で適応的に重みの更新幅を変更できる最適化手法について述べた。このような最適化手法に対して学習率のスケジュー

Algorithm 1 Adam

Input: η, β_1, β_2, λ, $\boldsymbol{\theta}_0$, $L(\boldsymbol{\theta})$
Initialize: $\boldsymbol{m}_0 \leftarrow 0$, $\boldsymbol{v}_0 \leftarrow 0$
for $t = 1$ to $\ldots$ **do**
    $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_{t-1})$
    $\boldsymbol{g}_t \leftarrow \boldsymbol{g}_t + \lambda \boldsymbol{\theta}_{t-1}$
    $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \boldsymbol{g}_t$
    $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \boldsymbol{g}_{t^2}$
    $\tilde{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$
    $\tilde{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$
    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \tilde{\boldsymbol{m}}_t / (\sqrt{\tilde{\boldsymbol{v}}_t} + \epsilon)$
end for
Return $\boldsymbol{\theta}_t$

Algorithm 2 AdamW

Input: η, β_1, β_2, λ, $\boldsymbol{\theta}_0$, $L(\boldsymbol{\theta})$
Initialize: $\boldsymbol{m}_0 \leftarrow 0$, $\boldsymbol{v}_0 \leftarrow 0$
for $t = 1$ to $\ldots$ **do**
    $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_{t-1})$
    $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \boldsymbol{g}_t$
    $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \boldsymbol{g}_{t^2}$
    $\tilde{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$
    $\tilde{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$
    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \tilde{\boldsymbol{m}}_t / (\sqrt{\tilde{\boldsymbol{v}}_t} + \epsilon) - \eta \lambda \boldsymbol{\theta}_{t-1}$
end for
Return $\boldsymbol{\theta}_t$

リングは、学習率 η の値自体を学習の進行に伴って変更するものであり、より安定した学習を促すための手段である。以下、三つのスケジューラを例として述べる。また、各スケジューラを用いた場合における学習率の遷移を図 3.5 に示す。

一つ目は、StepLRScheduler である。これは、初期学習率を η_0 として、エポック t における学習率 η_t を

$$\eta_t = \eta_0 \gamma^{\left\lfloor \frac{t}{\text{step\_size}} \right\rfloor} \tag{3.66}$$

で与えるスケジューラである。これは、学習が step_size エポック進むごとに学習率を γ 倍することで、学習率を段階的に変化させる。シンプルで分かりやすいが、学習率の変化が不連続的になる。

二つ目は、ExponentialLRScheduler である。これは、η_t を

$$\eta_t = \eta_0 \cdot \exp\left(-\gamma t\right) \tag{3.67}$$

で与えるスケジューラである。これは、学習が 1 エポック進むごとに学習率を指数関数的に変化させる。StepLRScheduler と比較して、学習率を連続的に変化させられる特徴がある。

三つ目は、Cosine Annealing with Warmup である。これは、η_t を

$$\eta_t = \begin{cases} \eta_{\min} + \left(\frac{t}{\text{warmup\_steps}}\right)(\eta_{\max} - \eta_{\min}) & \text{if } t < \text{warmup\_steps} \\ \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{(t - \text{warmup\_steps})\pi}{t_{\max} - \text{warmup\_steps}}\right)\right) & \text{if } t \geq \text{warmup\_steps} \end{cases} \tag{3.68}$$

で与えるスケジューラである。ここで、$\eta_{\min}$ は最小学習率、$\eta_{\max}$ は最大学習率、$t_{\max}$ は最大エポックである。warmup_steps は学習率を $\eta_{\min}$ から $\eta_{\max}$ まで線形に増加させるのにかけるエポック数を指定するパラメータである。エポック数が warmp_steps 以上となれば、cos 関数に従って学習率を減衰させる。Cosine Annealing with Warmup は不安定になりがちな学習初期、学習率が低い状態から開始して徐々に学習率を大きくし、十分大きくしてから再び学習率を小さくすることで学習の収束を促すスケジューラである。

3.2.6 正規化

DNN の学習過程では学習の進行に伴って重みが変化するため、その度に各層への入力の分布が変わってしまう。これは内部共変量シフト（Internal Covariate Shift）と呼ばれ、ネットワークの学習を不安定にする原因となる。これに対し、バッチ正規化（Batch Normalization）[33] が有効である。バッチ正規化は、ミニバッチ学習を行う際にそのミニバッチ内における平均と標準偏差を計算し、特徴量を標準化する。バッチサイズを N、バッチ正規化への入力を

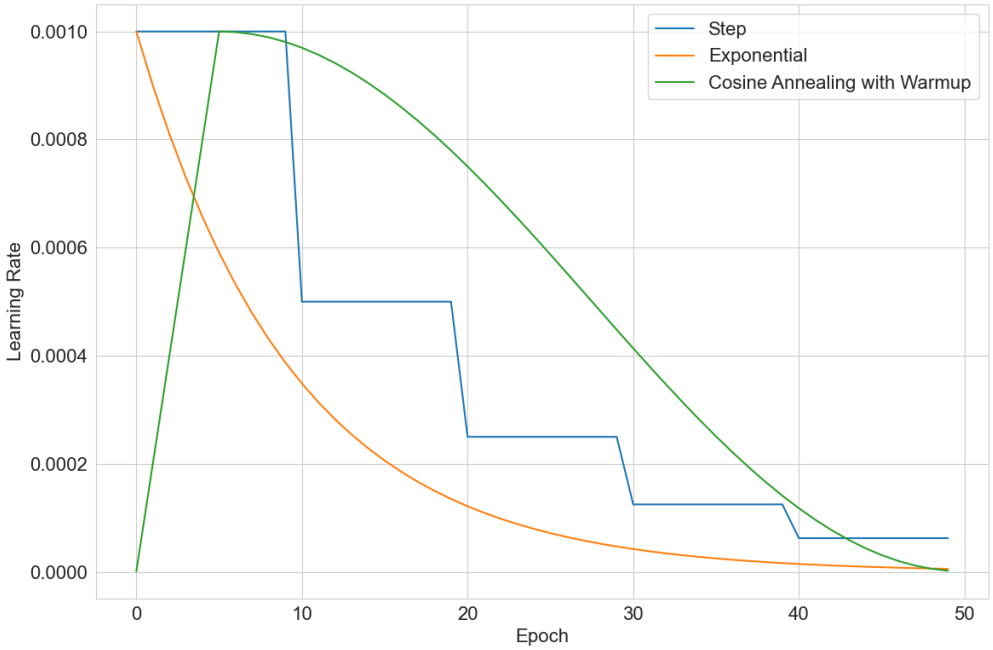


図 3.5: スケジューラによる学習率の変化

$\{\boldsymbol{x}_i \in \mathbb{R}^{D\times T} | 1 \leq i \leq N\}$ とすると、出力 $\boldsymbol{y}_i \in \mathbb{R}^{D\times T}$ $(1 \leq i \leq N)$ は

$$\boldsymbol{\mu}_{\text{batch}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i \tag{3.69}$$

$$\sigma^2_{\text{batch}} = \frac{1}{N}\sum_{i=1}^{N} (\boldsymbol{x}_i - \boldsymbol{\mu}_{\text{batch}})^2 \tag{3.70}$$

$$\hat{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \boldsymbol{\mu}_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}} \tag{3.71}$$

$$\boldsymbol{y}_i = \boldsymbol{\gamma}\hat{\boldsymbol{x}}_i + \boldsymbol{\beta} \tag{3.72}$$

で与えられる。ここで、$\boldsymbol{\mu}_{\text{batch}}, \boldsymbol{\sigma}_{\text{batch}} \in \mathbb{R}^D$ はそれぞれミニバッチにおける平均ベクトルおよび分散ベクトル、$\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^D$ は学習可能なベクトルである。バッチ正規化では入力を一度ミニバッチ内の平均と分散を用いて標準化した上で、学習可能なベクトル $\boldsymbol{\gamma}$ の乗算と $\boldsymbol{\beta}$ の加算により、表現力を向上させている。バッチ正規化は、畳み込み層や全結合層と合わせて用いられることが多い。

バッチ正規化は DNN の学習の安定化に貢献するが、一方で、バッチサイズが小さい場合はデータの分布を安定させることが難しくなるという課題がある。これに対し、ミニバッチ内の各データごとに平均と分散を求めて標準化する、レイヤー正規化（Layer Normalization） [34] がある。上記と同じ変数を用い、特にデータごとにチャンネル方向に平均と分散を求める場合、

これは

$$\boldsymbol{\mu}_{\text{layer}} = \frac{1}{D}\sum_{d=1}^{D}\boldsymbol{x}_{i,d} \tag{3.73}$$

$$\boldsymbol{\sigma}^2_{\text{layer}} = \frac{1}{D}\sum_{d=1}^{D}(\boldsymbol{x}_{i,d} - \boldsymbol{\mu}_{layer})^2 \tag{3.74}$$

$$\hat{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \boldsymbol{\mu}_{\text{layer}}}{\sqrt{\boldsymbol{\sigma}^2_{\text{layer}} + \epsilon}} \tag{3.75}$$

$$\boldsymbol{y}_i = \boldsymbol{\gamma}\hat{\boldsymbol{x}}_i + \boldsymbol{\beta} \tag{3.76}$$

で与えられる。ここで、$\boldsymbol{\mu}_{\text{layer}}, \boldsymbol{\sigma}_{\text{layer}} \in \mathbb{R}^T$ が各データごとの平均ベクトルおよび分散ベクトルである。レイヤー正規化は、LSTM や GRU、Transformer といった一次元系列に対して適用されるニューラルネットワークと合わせて用いられることが多い。

上述したバッチ正規化およびレイヤー正規化はニューラルネットワーク内部で特徴量に対して適用する正規化であった。一方、ニューラルネットワークの重み自体を正規化する手法として、重み正規化（Weight Normalization） [35] がある。これは、ある層の重みベクトルを $\boldsymbol{\theta}$ とすると、

$$\boldsymbol{\theta} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}g \tag{3.77}$$

で与えられる。これは、重みベクトル $\boldsymbol{\theta}$ を単位ベクトル $\boldsymbol{v}/\|\boldsymbol{v}\|$（ベクトルの向き）とスカラー g（ベクトルの大きさ）に分解することを意味する。重み正規化を適用した層については、重みの更新を $\boldsymbol{v}$ と g それぞれで行う。重みの向きが大きさから分離されているため、損失関数の勾配に対して安定した重みの更新を行うことができるメリットがある。例えば、損失関数の勾配が大きい場合、通常は重みベクトルの向きおよび大きさを一括に動かして更新するが、重み正規化を適用しておけば、向きの更新の程度と大きさの更新の程度を分けて考えることができるため、学習が安定する可能性がある。重み正規化は、学習が不安定になりやすい GAN の Generator や Discriminator に対して適用される場合がある。

3.2.7 正則化

DNN は大量のパラメータにより高い表現力を持つが、その分学習データに過剰に適合し、未知データに対する汎化性能が低いモデルとなる、過学習を引き起こす可能性がある。正則化は、このような DNN の過学習を防ぐための手段である。以下、具体的な方法を三つ述べる。

一つ目は、Weight Decay である。Weight Decay は、損失関数に DNN の重みおよびバイアスの二乗和を加算することにより、重みが極端大きくなることを防ぐ手法である。損失関数を L、重みとバイアスをまとめた変数を $\boldsymbol{\theta}$ とすると、Weight Decay を適用した場合における損失関数 L_w は

$$L_w = L + \sum_{i=1}^{|\boldsymbol{\theta}|}\boldsymbol{\theta}_i^2 \tag{3.78}$$

で与えられる。ここで、$|\boldsymbol{\theta}|$ は重みとバイアスの総数である。

二つ目は、Dropout [36] である。Dropout は、学習時に一定の確率で一部のノードをランダムに無効化（出力を 0 に変換）する手法である。Dropout は学習時にランダムにノードを無効化する一方で、推論時は全ノードの出力を利用する。この挙動の違いに対し、Dropout への入力を $\boldsymbol{h} \in \mathbb{R}^D$、ノードが無効化される確率を p とすると、学習時の Dropout 出力 $\boldsymbol{h}_{\text{training}} \in \mathbb{R}^D$ および推論時の Dropout 出力 $\boldsymbol{h}_{\text{inference}} \in \mathbb{R}^D$ は、

$$\boldsymbol{h}_{\text{training}} = \frac{\boldsymbol{h} \odot \boldsymbol{m}}{1-p} \tag{3.79}$$

$$\boldsymbol{h}_{\text{inference}} = \boldsymbol{h} \tag{3.80}$$

で与えられる。ここで、$\boldsymbol{m} \in \{0,1\}^D$ は各要素 $\boldsymbol{m}_i$ が確率 $1-p$ で 1、確率 p で 0 をとるベルヌーイ分布に従う確率変数であり、$\odot$ は要素積を表す。式 (3.79) より、学習時は Dropout 出力を $1/(1-p)$ でスケーリングしていることがわかる。この理由は、学習時の出力と推論時の出力の期待値を一致させるためである。実際、確率変数 $\boldsymbol{m}$ の従う確率分布上で $\boldsymbol{h}_{\text{training}}$ の期待値をとれば、

$$\mathbb{E}[\boldsymbol{h}_{\text{training}}] = \mathbb{E}\left[\frac{\boldsymbol{h}\boldsymbol{m}}{1-p}\right] \tag{3.81}$$

$$= \frac{\boldsymbol{h}}{1-p}\mathbb{E}[\boldsymbol{m}] \tag{3.82}$$

$$= \frac{\boldsymbol{h}}{1-p}(1-p) \tag{3.83}$$

$$= \boldsymbol{h} = \boldsymbol{f}_{\text{inference}} \tag{3.84}$$

となり、スケーリングによって学習時の Dropout 出力の期待値と推論時の Dropout 出力が一致することがわかる。

三つ目は、Early Stopping である。Early Stopping は、検証データに対する損失の増加を監視し、設定したエポック数だけ増加し続けた場合に学習を停止する手法である。これにより、学習データに対する過度なフィッティングを防止する。

3.2.8 学習の安定化

DNN の学習は勾配降下法によって行われるが、ここで勾配が大きくなりすぎると重みの更新幅が過剰に大きくなり、学習が不安定にある可能性がある。これに対して、Gradient Clipping が有効である。Gradient Clipping は、勾配のノルムが設定した閾値を超える場合に、勾配をスケーリングしてそのノルムを抑制する方法である。勾配の更新式は、勾配を $\boldsymbol{g}$、設定した閾値を c とすると、

$$\boldsymbol{g} \leftarrow \boldsymbol{g} \cdot \frac{c}{\max(\|\boldsymbol{g}\|_2, c)} \tag{3.85}$$

で与えられる。ここで、$\|\boldsymbol{g}\|_2$ は勾配の L2 ノルムである。損失関数がスムーズに下がらない場合に、学習を安定させる一つの手段として有効である。

また、近年は数億単位のパラメータを持つ大規模なモデルも提案されており、こういった規模間のモデルを構築して学習する場合、それ相応のメモリが必要になる。これに対し、同一のマシンでもバッチサイズを小さくすることによって計算可能になると考えられるが、バッチサイズを小さくすることは各データのノイズの影響を強くする要因となり、学習が不安定になる可能性がある。このような状況では、Gradient Accumulation が有効である。Gradient Accumulation は、小さなバッチサイズで計算した勾配を複数回に渡って累積し、設定した回数ごとに重みの更新を行う手法である。各イテレーション t において得られる勾配を $\boldsymbol{g}_t$、累積される勾配を $\boldsymbol{g}_{\text{accum}}$ とすると、$\boldsymbol{g}_{\text{accum}}$ の更新は

$$\boldsymbol{g}_{\text{accum}} \leftarrow \boldsymbol{g}_{\text{accum}} + \boldsymbol{g}_t \tag{3.86}$$

で与えられる。ここで、設定した累積回数を k、学習率を η とすると、k 回に一回行う重み $\boldsymbol{\theta}$ の更新は

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\boldsymbol{g}_{\text{accum}}}{k} \tag{3.87}$$

で与えられる。k 回分の勾配を累積した分、重みを更新する際には $1/k$ にスケーリングすることで、実質的に k 倍のバッチサイズにおける学習が可能になる。また、重み更新後は累積した勾配を 0 にリセットして、次の k 回の累積に備える。

3.2.9 自己教師あり学習

近年、音声や動画を用いる分野では、自己教師あり学習（Self-Supervised Learning; SSL）を事前に行ったモデルを解きたい問題に FineTuning する、転移学習の有効性が確認されている。FineTuning するタスクは、下流タスク（Downstream Task）と呼ばれることが多い。ここで、自己教師あり学習とは、ラベルのないデータから特徴を学習するために、ラベルなしデータ自体から擬似的にラベルとなるものを作成して、それを元に教師あり学習を行う方法である。ラベルのないデータからそのデータの特徴を学習させるという点で、クラスタリングや次元削減などの教師なし学習に類似している。しかし、教師なし学習においては、ラベルなしデータから擬似ラベルを作成するという手順は踏まない。この点で、自己教師あり学習は教師あり学習に近い教師なし学習だと解釈できる。本研究においては、自己教師あり学習モデルの FineTuning を精度改善を目指したアプローチとして検討したため、本節では特に、本研究で検討対象とした事前学習済みモデルの行っている、Masked Prediction という自己教師あり学習の方法について述べる。

Masked Prediction では、入力データの一部をマスクし、マスクされた領域の内容を予測するようにモデルを学習させる。穴埋め問題を解かせるイメージである。入力データを $\boldsymbol{X} \in \mathbb{R}^{T \times D}$、このうちマスクされるインデックスの部分集合を $\mathcal{M} \subset \{1, \dots T\}$ とする。この時、マスクされた入力 $\boldsymbol{X}^{\text{masked}} \in \mathbb{R}^{T \times D}$ は、

$$\boldsymbol{X}_t^{\text{masked}} = \begin{cases} \boldsymbol{X}_t & \text{if } t \in \mathcal{M} \\ \boldsymbol{m} & \text{if } t \notin \mathcal{M} \end{cases} \tag{3.88}$$

で与えられる。ここで、$\boldsymbol{X}_t, \boldsymbol{X}_t^{\text{masked}} \in \mathbb{R}^D$ はそれぞれ位置 t における入力ベクトル、マスクされた入力ベクトルで、$\boldsymbol{m} \in \mathbb{R}^D$ はマスクベクトルである。Masked Prediction ではマスキングによって得られた $\boldsymbol{X}^{\text{masked}}$ をモデルへの入力とし、$\boldsymbol{X}$ を擬似ラベルとして学習を行う。ここで、音声の分野における HuBERT [10] や、動画音声の分野における AVHuBERT [3] では、扱うデータ自体が連続値となっているため、クラスタリングによる離散化を適用して擬似ラベルを作成する。ここで、クラスタ数を C とするクラスタリングによって得られた擬似ラベルを $\boldsymbol{X}^{\text{discretized}} \in \{0,1\}^{T\times C}$ とすると、モデルの予測値 $\hat{\boldsymbol{X}} \in [0,1]^{T\times C}$ に対する損失は、

$$L_{CE}(\boldsymbol{X}^{\text{discretized}}, \hat{\boldsymbol{X}}) = -\frac{1}{|\mathcal{M}|}\sum_{t\in\mathcal{M}}\sum_{c=1}^{C}\boldsymbol{X}_{t,c}^{\text{discretized}}\log(\hat{\boldsymbol{X}}_{t,c}) \tag{3.89}$$

で与えられる。すなわち、マスクされた位置に限定した Cross Entropy Loss である。この損失を最小化するようにモデルを学習させることで、特に音声や動画における文脈的な構造を学習させることが可能である。実際、HuBERT は音声からその発話内容を推定する音声認識、AVHuBERT は動画から発話内容を推定する VSR などで、スクラッチで学習した場合と比較して高い精度を達成できることが示されている。近年ではこういったモデルを転移学習することが、下流タスクにおけるスタンダードになりつつある。

4　動画音声合成モデルの検討

4.1　音声合成法

提案手法の構築手順は 3 段階に分かれる。ネットワークの構造を図 4.1 に示す。一段階目では、動画を入力として、メルスペクトログラムと HuBERT 離散特徴量、HuBERT 中間特徴量を推定するネットワーク A を学習する（図 4.1 の A）。ここで、HuBERT 離散特徴量は HuBERT Transformer 層から得られる特徴量を k-means 法によってクラスタリングすることで離散化した値、HuBERT 中間特徴量は HuBERT における畳み込み層出力で、HuBERT の事前学習時にマスク対象となる値のことを指す。図 4.2 にこれらの取得位置を示す。第一段階では、AVHuBERT を動画からの特徴抽出に利用した。これにより、動画の空間情報は完全に圧縮され、768 次元の一次元系列となる。その後、事前学習済みの話者識別モデル [37] によって音声波形から得られる 256 次元の話者 Embedding を、各フレームでチャンネル方向に結合する。これによって特徴量は 1024 次元に拡張され、全結合層によって再度 768 次元に圧縮する。その後、畳み込み層と全結合層からなるデコーダを通すことによって、話者 Embedding を結合した特徴量に対する変換を施した。これにより、特にメルスペクトログラムにおいて話者性が正しく反映されることを狙った。複数話者モデルであっても、入力である動画の見た目から話者性を判別できる可能性があったが、将来的な未知話者対応への拡張性も考慮して、本研究では補助特徴量として入力することとした。デコーダは残差結合を利用したブロック単位で構成され、各ブロックに 2 層の畳み込み層を設けた。各畳み込み層のチャンネル数は 768、カーネルサイズは 3 であり、3 ブロック積み重ねた。最後に全結合層を通し、所望の次元に変換することで予測対象を得た。ネットワーク A の役割は、続くネットワーク B の入力である HuBERT 中間特徴量を提供することである。これに対し、メルスペクトログラムと HuBERT 離散特徴量の推定を同時に行った理由は、先行研究においてマルチタスク学習の有効性が確認されていることを考慮し、ネットワーク A でもマルチタスク学習を採用しておこうと考えたからである。

二段階目では、一段階目に学習されたネットワーク A の重みを固定した状態で HuBERT 中間特徴量を推定し、それを入力としてメルスペクトログラムと HuBERT 離散特徴量を推定する、HuBERT Transformer 層を中心としたネットワーク B の学習を行う（図 4.1 の B）。HuBERT Transformer 層出力は AVHuBERT 出力と同じ 768 次元の特徴量となるため、これに対してネットワーク A と同様に話者 Embedding を結合し、デコーダを通すことで予測値を得た。ネットワーク B の役割は、音声波形への変換に必要となるメルスペクトログラムと HuBERT 離散特徴量の予測である。HuBERT Transformer 層の転移学習を検討した狙いについて、HuBERT は自己教師あり学習時、畳み込み層出力にマスクを適用し、Transformer 層を通すことによってマスクされた部分を推定しようとする。これにより、音声の文脈を考慮するのに長けた学習済み重みが、特に Transformer 層で獲得されると仮定した。これに基づき、本研究では HuBERT Transformer 層を動画音声合成に Fine Tuning することにより、動画を入力とした AVHuBERT を中心とするネットワーク A における推定残差を、音声自体の文脈を考慮することによって軽減し、動画から直接推定しきれなかった部分を補うことでの精度改善を狙った。

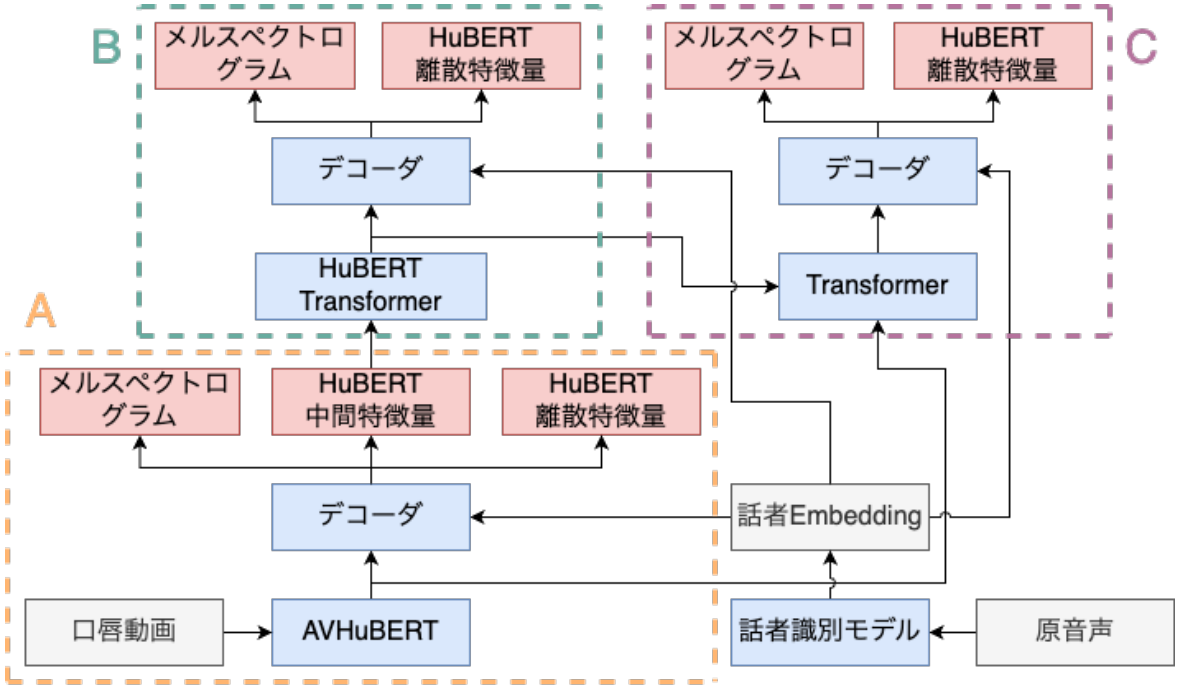


図 4.1: 提案するネットワークの構造

三段階目では、二段階目までに学習されたネットワーク A とネットワーク B の重みを固定した状態で、AVHuBERT から得られる特徴量と、HuBERT Transformer 層から得られる特徴量の二つを結合し、それらを入力として再びメルスペクトログラムと HuBERT 離散特徴量の予測を行うネットワーク C を学習した（図 4.1 の C）。ネットワーク C では、はじめに前述した二つの特徴量をチャンネル方向に結合することで、1536 次元の入力特徴量を得る。これに対して全結合層を施すことで再度 768 次元に圧縮し、4 層の Transformer 層を通すことで系列全体を考慮した特徴抽出を改めて行った。その後、ネットワーク A,B と同様に話者 Embedding を結合し、デコーダを通すことによって予測値を得た。ここで、ネットワーク C の Transformer 層におけるパラメータについては、AVHuBERT や HuBERT と同様にチャンネル数を 768、ヘッド数は 12 とした。ネットワーク C の役割は、ネットワーク B と同様に音声波形への変換に必要な特徴量の予測である。ここでの狙いについて、まず、AVHuBERT から得られる特徴量と HuBERT Transformer 層から得られる特徴量は、どちらもデコーダへの入力となる点で同じである。一方、AVHuBERT は動画を入力、HuBERT Transformer 層は HuBERT 中間特徴量を入力とするため、これら特徴量の元となる入力は異なっている。ここでは、概ね同じ予測対象のために利用される二つの特徴量（ネットワーク A では HuBERT 中間特徴量の予測も行っているため、全く同じではない）が、入力の違いに依存して内部の Self Attention により注意される部分が変化し、何らかの異なった情報を持っている可能性があると仮定した。よって、両方の特徴量を考慮し、単一特徴量への依存を解消することで、汎化性能向上による予測精度の改善を狙った。

以上が提案手法の全体像であるが、今回ベースラインとする先行研究 [9] に基づいたマルチタスク学習手法は、本研究におけるネットワーク A で、HuBERT 中間特徴量を推定しないもの

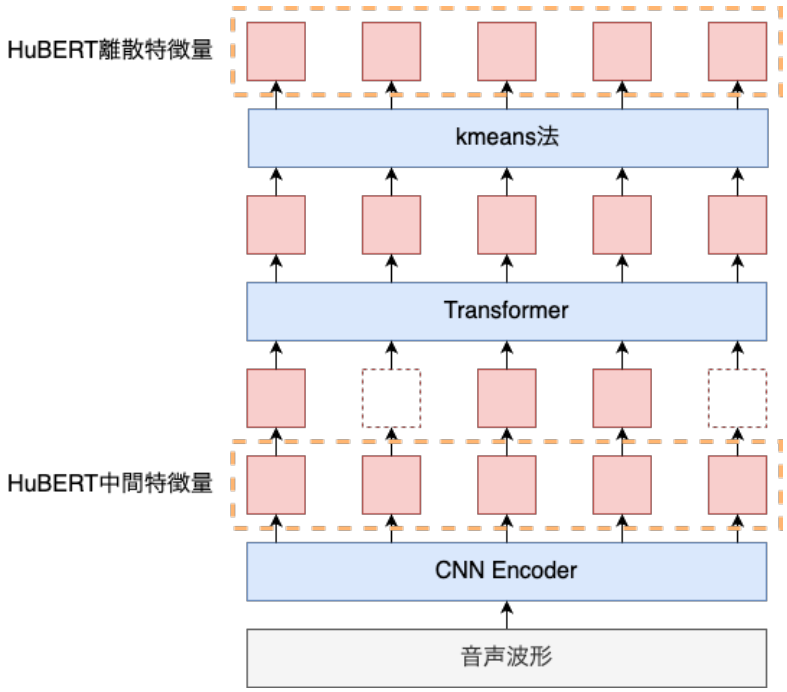


図 4.2: HuBERT 中間特徴量と HuBERT 離散特徴量の取得位置

に当たる。

以上のモデルにより、動画からメルスペクトログラムと HuBERT 離散特徴量が推定可能となる。その後、先行研究 [9] に基づく Multi-input Vocoder を用い、メルスペクトログラムと HuBERT 離散特徴量を入力として音声波形に変換することで、最終的な合成音声を得た。Multi-input Vocoder は HiFi-GAN [38] をベースとしたモデルであり、音声波形を生成する Generator と、Multi-Period Discriminator（MPD）および Multi-Scale Discriminator（MSD）という二つの Discriminator によって構成される。

Generator の構造を図 4.3 に示す。左の特徴量予測モデルは、動画からメルスペクトログラムと HuBERT 離散特徴量を予測する、本研究において主な検討対象となる部分を表す。Generator の内部構造について、まず、前処理層はメルスペクトログラムと HuBERT 離散特徴量を入力として受け取り、その後のレイヤーに入力するための形状に変換する役割を持つ。メルスペクトログラムに対しては、時間方向に隣接した 2 フレームを次元方向に縦積みすることによって、100 Hz・80 次元の特徴量から 50 Hz・160 次元の特徴量に変換した後、全結合層によって 128 次元の特徴量に変換する。一方、HuBERT 離散特徴量は 50 Hz のインデックス系列であり、インデックスから 128 次元のベクトルへと変換する。その後、これらをチャンネル方向に結合することで 50 Hz・256 次元の特徴量を構成し、これをその後のレイヤーへの入力とする。この特徴量は、転置畳み込み層と複数種類の畳み込み層から構成されるブロックを通過していく。各ブロックについて、まず、転置畳み込み層は特徴量を時間方向にアップサンプリングする役割を果たす。実際、本研究では 50 Hz の入力特徴量から 16 kHz の音声波形まで、時間方向に 320 倍のアップサンプリングを行う必要がある。Generator では、これを複数のブロックを通して段階的に行っている。また、各ブロックの転置畳み込み層に積まれた複数種類の畳み込み層は、

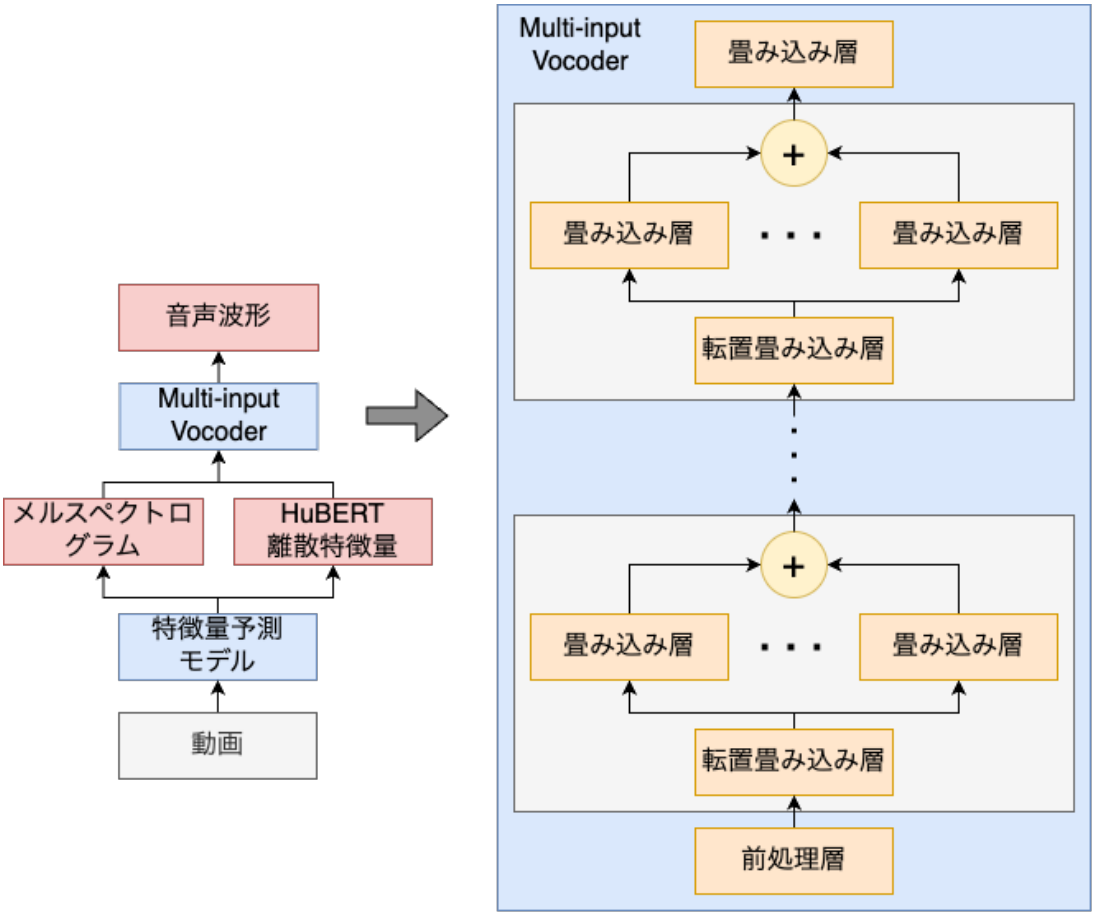


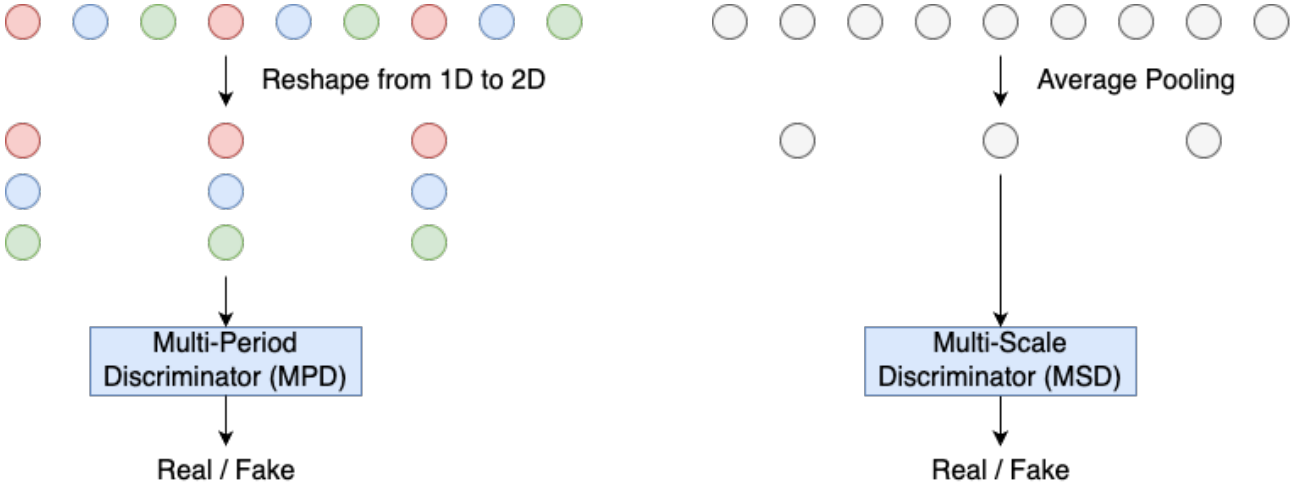
図 4.3: Multi-input Vocoder の構造

そのカーネルサイズとダイレーションが全て異なっている。複数の時間的な受容野を持つ畳み込み層からの出力をすべて加算することで、アップサンプリング後の特徴量からの特徴抽出を行う仕組みとなっている。表 4.1 に、Generator の各ブロックにおけるパラメータとブロックごとの出力特徴量の形状を示す。カラム名の右にある括弧がきが数値の意味を表しており、K はカーネルサイズ、S はストライド、D はダイレーション、C は次元（チャンネル数）、T は系列長である。畳み込み層についてはカーネルサイズとダイレーションを集合として表記しているが、実際はこれらの直積の元、すなわち (3, 1) や (3, 3)、(3, 5) をパラメータとする畳み込み層が存在することを表す。すなわち、転置畳み込み層一層に対し、その後の特徴量抽出は 15 種類の異なるカーネルサイズ、ダイレーションを設定した畳み込み層によって行われる。

Generator の学習時に用いられるのが、MPD および MSD という二つの Discriminator である。これらの概要を図 4.4 に示す。MPD では、一次元の音声波形を指定した周期をもとに Reshape することで二次元に変換し、これに対して二次元畳み込みを適用することで、入力された音声波形の原音声らしさを判定する Discriminator である。異なる周期を設定した Discriminator を複数利用することで、時間的な特徴を考慮できるように構成されている。一方、MSD では、音声波形に対して Average Pooling を適用することでダウンサンプリングし、これに一次元畳み込みを適用することで入力された音声波形の原音声らしさを判定する Discriminator である。MSD

表 4.1: Generator の各ブロックにおけるパラメータ

	転置畳み込み層 (K, S)	畳み込み層 (K, D)	出力特徴量の形状 (C, T)
1	(11, 5)	($\{3,5,7,9,11\}$, $\{1,3,5\}$)	(1024, 250)
2	(8, 4)	($\{3,5,7,9,11\}$, $\{1,3,5\}$)	(512, 1000)
3	(4, 2)	($\{3,5,7,9,11\}$, $\{1,3,5\}$)	(256, 2000)
4	(4, 2)	($\{3,5,7,9,11\}$, $\{1,3,5\}$)	(128, 4000)
5	(4, 2)	($\{3,5,7,9,11\}$, $\{1,3,5\}$)	(64, 8000)
6	(4, 2)	($\{3,5,7,9,11\}$, $\{1,3,5\}$)	(32, 16000)



(a) Multi-Period Discriminator（MPD）　(b) Multi-Scale Discriminator（MSD）

図 4.4: Multi-Period Discriminator（MPD）と Multi-Scale Discriminator（MSD）の概要

は Average Pooling によって系列長を抑えつつ、時間方向の連続的な特徴を考慮する狙いがある。これら二つの Discriminator については、HiFi-GAN と同様のパラメータで用いた。

4.2　実験方法

4.2.1　利用したデータセット

動画音声データセットには、男女二人ずつから収録した合計 4 人分のデータセット [25, 26] を用いた。これは ATR 音素バランス文 [39] から構成され、全話者共通で A から H セットを学習データ、I セットを検証データ、J セットをテストデータとして利用した。各分割ごとの文章数を表 4.2 に示す。

Multi-input Vocoder の学習に利用する音声データセットには、Hi-Fi-Captain（日本語話者二名分）[40] と JVS（parallel100 と nonpara30）[41] を利用した。Hi-Fi-Captain は train-parallel および train-non-parallel を学習データ、val を検証データ、eval をテストデータとして分割した。各分割ごとの文章数を表 4.2 に示す。JVS には話者に対して 1 から 100 まで番号が割り振られて

表 4.2: 利用したデータセットの文章数

	学習	検証	テスト
動画音声データセット	1598	200	212
Hi-Fi-Captain	37714	200	200
JVS	10398	1299	1300

おり、本実験では 1 から 80 番の話者を学習データ、81 番から 90 番の話者を検証データ、91 番から 100 番までの話者をテストデータとした。各分割ごとの文章数を表 4.2 に示す。また、JVS には読み上げ音声の parallel100 および nonpara30 と、裏声の falset10、囁き声の whisper10 が含まれる。本研究では、parallel100 と nonpara30 のみを利用した。

4.2.2 データの前処理

動画データは 60 FPS で収録されたものを ffmpeg により 25 FPS に変換して用いた。その後、手法 [42] により動画に対してランドマーク検出を適用した。このランドマークを利用することで口元のみを切り取り、画像サイズを（96, 96）にリサイズした。モデル入力時は動画をグレースケールに変換し、各フレームに対する正規化および標準化を適用した。正規化では、グレースケールが 0 から 255 までの値を取るため、最大値の 255 で割り、標準化では、AVHuBERT のプログラムで利用されていた平均値 0.421 と、標準偏差 0.165 をそのまま利用して、平均値を引いた後に標準偏差で割った。全体として、今回は事前学習済みの AVHuBERT の転移学習を行うため、そこでの前処理に合わせている。学習時は、ランダムクロップ、左右反転、Time Masking（一時停止）をデータ拡張として適用した。ランダムクロップは、（96, 96）で与えられる画像から（88, 88）をランダムに切り取る処理である。検証およびテスト時は、必ず画像中央を切り取るよう実装した。左右反転は、50%の確率で左右が反転されるよう実装した。Time Masking は、連続する画像の時間平均値を利用することによって、一時停止させるような効果を与えるデータ拡張手法である。動画 1 秒あたり 0 から 0.5 秒の間でランダムに停止区間を定め、その区間における動画の時間方向平均値を計算し、区間内のすべてのフレームをこの平均値で置換した。

音声データは 16 kHz にダウンサンプリングして用いた。それから、窓長 25 ms のハニング窓を用いて、シフト幅 10 ms で STFT を適用することでフレームレート 100 Hz のスペクトログラムに変換した。さらに、パワースペクトログラムに対して 80 次のメルフィルタバンクを適用し、メルスペクトログラムを得た上で対数スケールに変換した。また、Multi-input Vocoder の学習に利用した Hi-Fi-Captain と JVS については、無音区間のトリミング（-40 dBFS 未満かつ 500 ms 継続する区間を 100 ms までカット）を適用した。なぜなら、Multi-input Vocoder の学習時は、全体から 1 秒分をランダムサンプリングするように実装しており、元データに存在する無音区間を除去することによって、学習の安定化および得られる音声の品質改善に繋がったからである。また、話者 Embedding の取得には事前学習済みの話者識別モデル [37] を利用

した。動画音声データセット、Hi-Fi-Captain、JVS ともに、各話者学習データの中から 100 文章をランダムサンプリングし、各発話に対して得られたベクトルの平均値を用いた。この値を学習・検証・テストで一貫して用いるため、学習外の検証データやテストデータには非依存な値となっている。モデルに入力する際には、ベクトルの大きさで割って正規化した。

HuBERT は、HuggingFace に公開されている ReazonSpeech というデータセットによって学習されたモデル [43, 44] を利用した。ReazonSpeech は約 19000 時間の日本語音声からなるデータセットであり、日本語音声のコンテキストを大量のデータから学習したモデルである。今回用いるデータセットが日本語であることから、本研究の検討対象としては日本語音声に関する事前知識を豊富に有するモデルが適していると考え、このモデルを選択した。本研究で用いる HuBERT 中間特徴量および HuBERT 離散特徴量について、HuBERT 中間特徴量は、音声波形に対して畳み込み層を適用した出力で、HuBERT の事前学習時にはマスクの対象となる特徴量を利用した。一方、HuBERT 離散特徴量は、HuBERT Transformer 層の 8 層目出力に、k-means 法によるクラスタリングを適用することで得た。あえて 8 層目出力を選択した理由は、HuBERT のレイヤーごとの特徴量について、音素の One-hot ベクトルおよび単語の One-hot ベクトルとの相関を、Canonical Correlation Analysis（CCA）によって調べた先行研究 [45] より、8 層目出力がそのどちらとも相関が高く言語的な情報に近いと判断したからである。クラスタ数については決め打ちとなるが、今回は離散化した結果が言語的な情報を持つことを狙い、比較的少ない 100 とした。k-means 法の学習には動画音声データセットにおける学習用データ全てを利用し、これを用いて動画音声データ、Hi-Fi-Captain、JVS に対するクラスタリングを実施した。

4.2.3 学習方法

一段階目について、損失関数はメルスペクトログラムの MAE Loss L_{mel} と HuBERT 離散特徴量の Cross Entropy Loss L_{ssl^d} 、HuBERT 中間特徴量の MAE Loss L_{ssl^i} の重み付け和とした。それぞれの重み係数を $\lambda_{mel}, \lambda_{ssl^d}, \lambda_{ssl^i}$ とすると、

$$L = \lambda_{mel} * L_{mel} + \lambda_{ssl^d} * L_{ssl^d} + \lambda_{ssl^i} * L_{ssl^i} \tag{4.1}$$

となる。最適化手法には AdamW [32] を利用し、$\beta_1 = 0.9$、$\beta_2 = 0.98$、$\lambda = 0.01$ とした。学習率に対するスケジューラには、Cosine Annealing with Warmup を利用した。開始時の学習率は 1.0×10^{-6} として、最大エポック数の 10%に至るまでは学習率を 1.0×10^{-3} まで線形に増加させ、その後のエポックでは cosine 関数に基づいて 10×10^{-6} まで減少させた。バッチサイズはメモリの都合上 4 としたが、学習の安定化のため、Gradient Accumulation によって各イテレーションにおける勾配を累積させ、8 イテレーションに一回重みを更新するようにした。モデルに入力する動画の秒数は 10 秒を上限とし、それを超える場合はランダムにトリミング、それに満たない場合はゼロパディングした。ゼロパディングした部分は損失の計算からは除外した。勾配のノルムは 3.0 を上限としてクリッピングすることで、過度に大きくなることを防止した。最大エポック数は 50 とし、10 エポック連続して検証データに対する損失が小さくなら

ない場合には、学習を中断するようにした（Early Stopping）。また、学習終了時には検証データに対する損失が最も小さかったエポックにおけるチェックポイントを保存し、これをテストデータに対する評価に用いた。

第二段階について、損失関数はメルスペクトログラムの MAE Loss と HuBERT 離散特徴量の Cross Entropy Loss の重み付け和とした。これは式 (4.1) において、$\lambda_{ssl^i} = 0.0$ と固定した場合に相当する。最適化手法には AdamW を利用し、$\beta_1 = 0.9$、$\beta_2 = 0.98$、$\lambda = 0.01$ とした。学習率に対するスケジューラには、Cosine Annealing with Warmup を利用した。開始時の学習率は 1.0×10^{-6} として、最大エポック数の 10%に至るまでは学習率を 5.0×10^{-4} まで線形に増加させ、その後のエポックでは cosine 関数に基づいて 10×10^{-6} まで減少させた。学習率の最大値は第一段階の 1/2 となっているが、これは値を半減させることによって学習を安定させることができたためである。その他のパラメータは、第一段階における値と同じである。

第三段階について、Cosine Annealing with Warmup における最大学習率のみ 1.0×10^{-3} としたが、それ以外は第二段階と同様である。

Multi-input Vocoder の学習では、Hi-Fi-Captain と JVS を用いた。はじめに Hi-Fi-Captain のみを用いて学習させ、その後学習済みモデルを JVS によって再学習した。損失関数は HiFi-GAN と同様である。二つのデータセットを用いた理由について、Hi-Fi-Captain は男女一人ずつの文章数が豊富なデータセットであるため、高品質なモデルを構築可能であった。しかし、学習できる話者数が少ない分、学習外話者に対する合成音声の品質が低かった。そのため、一人当たりの文章数は 100 文章程度と少ないながらも、100 人分の話者からなる JVS を利用して再学習することによって、学習外話者に対する合成音声の品質を向上させた。最適化手法には AdamW を利用し、$\beta_1 = 0.8$、$\beta_2 = 0.99$、$\lambda = 1.0 \times 10^{-5}$ とした。学習率は 2.0×10^{-4} から開始し、1 エポック経過するごとに 0.99 かけて徐々に減衰させた。バッチサイズは 16 とし、ここでは Gradient Accumulation は利用しなかった。モデルへの入力は 1 秒を上限とし、それを超える場合はランダムにトリミング、それに満たない場合はゼロパディングした。勾配のノルムは 3.0 を上限としてクリッピングすることで、過度に大きくなることを防止した。最大エポック数は 30 とし、ここでは Early Stopping は適用しなかった。また、学習終了時には検証データに対する損失（メルスペクトログラムに対する L1 Loss）が最も小さかったエポックにおけるチェックポイントを保存し、これをテストデータに対する評価に用いた。また、Multi-input Vocoder の提案された先行研究 [9] においては、学習時にあえてメルスペクトログラムにノイズをかけることによって、合成音声に対する汎化性能を向上させる学習方法が提案されている。本研究では、動画から推定されるメルスペクトログラムと HuBERT 離散特徴量の推定精度向上に焦点を当てたため、Multi-input Vocoder の学習は原音声から計算される特徴量そのもので行い、ボコーダ自体の汎化性能向上による精度改善は追求しなかった。

実装に用いた深層学習ライブラリは PyTorch および PyTorch Lightning である。GPU には NVIDIA RTX A4000 を利用し、計算の高速化のため Automatic Mixed Precision を適用した。

4.2.4 比較手法

比較手法は、以下の五つである。

1. メルスペクトログラムと HuBERT 離散特徴量のマルチタスク学習手法（ベースライン）
2. 提案手法のネットワーク B で、事前学習済み重みを読み込まずに HuBERT Transformer 層を用いる手法
3. 提案手法のネットワーク C で、事前学習済み重みを読み込まずに HuBERT Transformer 層を用いる手法
4. 提案手法のネットワーク B で、事前学習済み重みを読み込んで HuBERT Transformer 層を用いる手法
5. 提案手法のネットワーク C で、事前学習済み重みを読み込んで HuBERT Transformer 層を用いる手法

手法 1 が先行研究において有効性が確認された手法であり、今回の実験においてベースラインとなる。これに対する改善案として、手法 2 から手法 5 が提案手法である。手法 2 と手法 3 及び手法 4 と手法 5 の違いは事前学習済み重みを読み込むか否かであり、これらを比較することで初期値を事前学習済み重みとして転移学習することの有効性を調べた。

4.2.5 客観評価

合成音声の客観評価には、二種類の指標を用いた。一つ目は、音声認識の結果から算出した単語誤り率（Word Error Rate; WER）である。WER の計算方法について、まず、正解文字列 s_1 と音声認識モデルによる予測文字列 s_2 に対し、レーベンシュタイン距離によってその差分を測る。レーベンシュタイン距離は、二つの文字列を一致させるために必要なトークンの挿入数 I、削除数 D、置換数 R の和の最小値として定義される。WER は、レーベンシュタイン距離を測ることによって得られた I, D, R を利用し、

$$\mathrm{WER}(s_1, s_2) = \frac{I + D + R}{|s_1|} \tag{4.2}$$

で与えられる。ここで、$|s_1|$ は正解文字列 s_1 のトークン数を表す。実際には、音声認識モデルに Whisper [46] を利用し、出力される漢字仮名交じり文に対して MeCab を用いて分かち書きを行った上で、jiwer というライブラリを用いて算出した。Whisper は Large モデルを利用し、MeCab の辞書には unidic を利用した。WER の値は 0%から 100%であり、この値が低いほど音声認識の誤りが少ないため、より聞き取りやすい音声であると判断した。

二つ目は、話者 Embedding から計算したコサイン類似度である。モデルへの入力値を計算するのに用いた話者識別モデルを同様に利用し、サンプルごとに評価対象音声の話者 Embedding と原音声の話者 Embedding のペアでコサイン類似度を計算した。今回構築するモデルは 4 人の話者に対応するモデルとなるため、原音声に似た声質の合成音声が得られているかをこの指標で評価した。値は 0 から 1 であり、高いほど原音声と類似した合成音声だと判断できる。

4.3 結果

4.3.1 客観評価

まず、損失関数 (4.1) の重み係数 λ_{ssl^d} を変化させた時の、客観評価指標の全テストデータに渡る平均値を表 4.3 に示す。各手法ごとに 0.0001 から 1.0 まで 10 倍刻みで 5 段階検討し、各手法の客観指標ごとに最も優れた値を下線で示している。最良エポックは検証データに対する損失が最小となったエポックであり、テストデータの合成にはこのエポックにおけるチェックポイントを利用した。また、L_{mel}、L_{ssl^d}、L は最良エポックにおける検証データに対する損失の平均値である。また、これ以降の比較のために、最適だと考えられる λ_{ssl^d} の値を選択しており、選択された行を太字で表している。

手法 1 では、λ_{ssl^d} の値が 0.0001 の時に WER が最も低く、0.01 の時に話者類似度が最も高くなった。現状 WER の高さが特に課題であり、話者類似度は 0.004 とわずかな違いでもあるため、今回は WER が最小であることを優先して 0.0001 が最適であると判断した。λ_{ssl^d} の値による評価指標の変化について、WER は λ_{ssl^d} の値を大きくするのに伴って単調に増加していることがわかる。一方、話者類似度は λ_{ssl^d} の値が 0.1 以上となった時に、0.01 以下であった場合と比較して顕著に低下することがわかる。次に、図 4.5 に手法 1 における学習曲線の結果を示す。横軸がエポック数、縦軸が損失の値を表す。損失の値は各エポックにおける平均値である。実線は検証データに対する損失、点線は学習データに対する損失を表しており、線の色は λ_{ssl^d} の違いを表す。また、丸いマーカーは表 4.4 に示した最良エポック時における損失の値を表す。学習曲線より、λ_{ssl^d} の値を変化させることによって、特に L_{ssl^d} の傾向が変化していることがわかる。具体的には、λ_{ssl^d} の値を 0.0001 から 1.0 へと増加させるのに伴って、学習初期における損失の下がり方が急峻になっており、達する最小値自体が小さくなっていることがわかる。また、λ_{ssl^d} の値が 0.1 以上の場合、検証データに対する L_{ssl^d} は早いうちから増加傾向に転じている。これに伴い、今回は検証データに対する L の値を監視し、Early Stopping の適用と最良エポックの決定を行なったため、L_{mel} が下がり切らない状態で学習が中断される結果となった。客観評価指標において、特に λ_{ssl^d} の値が 0.1 以上となるとき、0.01 以下の場合と比較して話者類似度の低下や、WER の上昇傾向が見られていたが、学習曲線の挙動より、L_{mel} を下げきれなくなっていたことが原因として考えられる。最適な λ_{ssl^d} の値は客観評価指標から 0.0001 としたが、0.01 以下ではそれと概ね同程度の品質であったことを考えると、手法 1 では検証データに対する L_{mel} の値を十分小さくすることのできる λ_{ssl^d} が適していると考えられる。

手法 2 では、λ_{ssl^d} の値が 0.1 の時に WER が最も低く、0.0001 の時に話者類似度が最も高くなった。ここでは λ_{ssl^d} の値が 0.1 の時に、ベースラインで選択された最適なケースと比較して WER が 9.1%低下しており、話者類似度についても 0.003 高くなっていることから、0.1 が最適だと判断した。λ_{ssl^d} の値による評価指標の変化について、λ_{ssl^d} を 0.1 以上とすることで、0.01 以下の場合と比較して WER が低下する傾向が見られた。しかし、1.0 まで大きくすると 0.1 の場合よりも WER が大きくなっており、単調に減少していないこともわかる。話者類似度については、λ_{ssl^d} の値を 0.1 としたときに、0.01 以下の場合と比較して値が少し低下し、さらに 1.0 ま

で大きくすることで 0.1 以下の場合と比較して顕著に低下していることがわかる。次に、図 4.7 に手法 2 における学習曲線の結果を示す。手法 1 と同様に、λ_{ssl^d} の値を 0.0001 から 1.0 へと増加させるのに伴って、学習初期における L_{ssl^d} の下がり方が急峻になっており、達する最小値自体が小さくなっていることがわかる。また、λ_{ssl^d} が 1.0 の場合に、L_{mel} を下げきれなくなる傾向が見られる。加えて、手法 2 では λ_{ssl^d} が 0.1 の場合における L_{ssl^d} の増加が緩やかであり、L_{mel} も十分下げられていることがわかる（ただし、表 4.3 の L_{ssl^d} より、手法 1 の λ_{ssl^d} が 0.1 の場合と比較して、損失の値自体は大きい）。さらに、λ_{ssl^d} が 0.1 の場合、L_{mel} が達する最小値自体が、λ_{ssl^d} が 0.01 以下の場合と比較して小さくなっていることがわかる（具体的な値は表 4.3 の L_{mel} の値を参照）。これは手法 1 では見られなかった新たな傾向であった。最適な λ_{ssl^d} の値は客観評価指標から 0.1 としたが、学習曲線の挙動より、この時他の値の場合と比較して L_{mel} と L_{ssl^d} の両方をバランスよく下げられていたことがわかった。よって、手法 2 においては L_{mel} と L_{ssl^d} の両方をバランスよく下げられるような、程よい大きさの重みが適していると考えられる。

手法 3 における最適値の選択理由は手法 2 と同様であり、λ_{ssl^d} の値による評価指標の変化についても同様であった。次に、図 4.8 に手法 3 における学習曲線の結果を示す。傾向は手法 2 と概ね同様であるが、手法 3 では λ_{ssl^d} の値が 0.1 の場合における L_{ssl^d} の増加が早く、学習が早期に中断されている点は異なる。しかし、この時 L_{mel} も十分下げられており、L_{mel} が達する最小値自体が、λ_{ssl^d} が 0.01 以下の場合と比較して小さくなっていることがわかる（具体的な値は表 4.3 の L_{mel} の値を参照）。以上より、細かな違いはあるものの手法 3 は手法 2 と同様の挙動を示し、λ_{ssl^d} は L_{mel} と L_{ssl^d} の両方をバランスよく下げられるような、程よい大きさの重みが適していると考えられる。

手法 4 及び手法 5 についても、最適値の選択理由は手法 2 と同様である。しかし、これらは λ_{ssl^d} の値を 0.1 としたときの話者類似度の低下の度合いが手法 2、手法 3 と比較して大きい点で、傾向が異なっていた。次に、図 4.9 に手法 4、図 4.10 に手法 5 における学習曲線の結果を示す。傾向については、L_{ssl^d} の増加の程度について細かな違いはあるが、概ね手法 2 と同様であり、L_{mel} と L_{ssl^d} の両方をバランスよく下げられるような、程よい大きさの重みが適していると考えられる。また、手法 4 と手法 5 において、λ_{ssl^d} が 0.1 の場合における話者類似度は、手法 2 および手法 3 と比較して低くなっていたが、L_{mel} や L_{ssl^d} の値からは一貫した傾向が見られない。これより、平均値からこの違いを分析することは難しいが、実際の細かな誤差の出方が異なっていて、これが影響を与えているのではないかと考える。

次に、最適なチューニングをした場合における、手法ごとの客観評価指標の全テストデータに渡る平均値を表 4.4 に示す。分析合成は、原音声から計算した特徴量を入力として、Multi-input Vocoder で逆変換した合成音声であり、本実験下において合成音声により達成され得る上限値を表す。手法 1 から手法 5 については、表 4.3 において太字としたもの、すなわち最適なチューニングだと判断されたものを選択している。また、分析合成、原音声を除いた合成音声（手法 1 から手法 5）の中で、最も優れた値を下線で示している。これより、WER、話者類似度ともに手法 3 が最も優れていることが分かる。特に、今回のベースラインである手法 1 と比較すると、

WERが9.5%低下し、話者類似度は0.01高くなっていることから、提案手法がベースラインよりもより聞き取りやすく、話者性を反映した音声を合成できたと考えられる。また、HuBERTの事前学習済み重みを初期値とすることの効果について、手法2と手法4を比較すると、手法2の方がWERが1.2%低く、話者類似度が0.017高いことがわかる。よって、本研究ではHuBERTの事前学習済み重みを初期値とした転移学習が有効である可能性を検討したが、むしろ事前学習済み重みを初期値とせず、ランダム初期化したモデルの方が優れていたと考えられる。これは仮説に反した結果であったが、事前学習済み重みによって与えられる初期値が、より良い局所解への収束には繋がらなかったことが原因だと考えられる。一方、アンサンブル手法の有効性について、手法2と手法3を比較すると、アンサンブル手法を導入することによってWERが0.4%低下し、話者類似度が0.007高くなっていることから、わずかではあるが改善していることがわかる。しかし、手法4と手法5を比較すると、WERは変化せず、話者類似度は0.017低下していることから、特に話者類似度の観点で悪化したことがわかる。よって、アンサンブル手法は必ずしも改善につながるわけではなく、改善するとしても顕著な変化はもたらさないと考えられる。

次に、最適なチューニングをした場合における手法ごとに、発話文章ごとのWERの比較を行った結果を図4.11に示す。ここで、縦軸は発話文章を表し、横軸はベースラインである手法1とその他手法の間でWERの差を計算し、発話文章ごとに平均した値を表す。負の値を取っているとき、手法1に対してより低いWERを達成したと解釈できる。図より、表4.4における平均値のみを見れば、手法2から手法5の全てが手法1に対してより低いWERを達成していたが、発話文章ごとに見れば、手法1に対してWERが高くなっているサンプルもあることがわかる。例えば、「ATR503_j11」では、手法2から手法5の全てがベースラインよりも20%前後高いWERとなっていることがわかる。また、全体的な傾向として手法ごとにWERはばらけており、いかなる場合においても最良となるようなモデルは構築できていないことがわかる。これより、現状のいかなるモデルも発話文章に対する汎化性能が不十分であり、さらなる改善が必要だと考えられる。

次に、話者ごとのWERの比較を行った結果を図4.12に示す。ここで縦軸は話者を表し、横軸はベースラインである手法1とその他手法の間でWERの差を計算し、話者ごとに平均した値を表す。負の値を取っているとき、手法1に対してより低いWERを達成したと解釈できる。これより、手法2から手法5は平均的にいかなる話者に対しても手法1より低いWERを達成していることがわかるが、一方でその改善の度合いは話者によって異なることがわかる。特に、「F02_kablab」はその他三人の話者と比較して改善の度合いが小さい。また、例えば「F01_kablab」では手法2と手法3がより有効である一方、「F02_kablab」では手法4と手法5がより有効となっており、有効な手法が話者によって異なることもわかる。これより、話者に対する性能の依存があると考えられ、さらなる汎化性能の向上が必要だと言える。

次に、話者ごとの話者類似度の比較を行った結果を図4.13に示す。ここで縦軸は話者を表し、横軸はベースラインである手法1とその他手法の間で話者類似度の差を計算し、話者ごとに平均した値を表す。正の値を取っているとき、手法1に対してより高い話者類似度を達成したと

解釈できる。これを見ると、手法 4 と手法 5 はいかなる話者に対しても話者類似度を悪化させたことがわかる。手法 4 と手法 5 は事前学習済み重みを初期値としたモデルであったため、この影響が出ている可能性がある。今回検討した HuBERT は音声の言語情報を考慮する点で強みがあり、実際音声認識ではその転移学習の有効性が確認されている。しかし、話者性を考慮するという点で特に有効でなく、話者性を軽視するような局所解への収束に繋がった可能性が考えられる。ただ、表 4.3 より手法 4 と手法 5 においても λ_{ssl^d} が 0.01 以下であれば、比較的高い話者類似度を達成していたため、損失の重み付けにも依存していると言える。また、手法 3 はいかなる話者に対しても類似度を向上させており、この点で優れていたと言える。手法 2 では「F02_kablab」のみ話者類似度が悪化しており、手法 3 と比較して話者に対する汎化性能が低かったと考えられる。

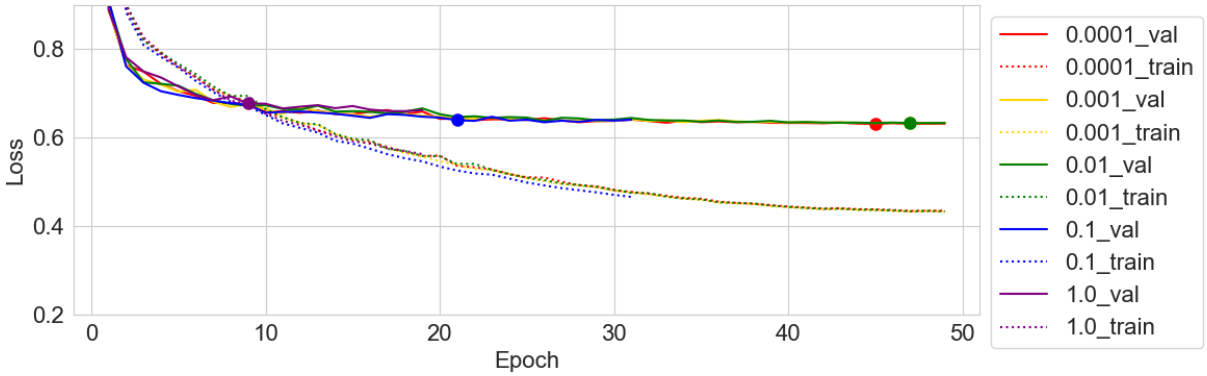
以上のことから、提案手法である手法 2 から手法 5 はいずれも手法 1 に対して平均的に WER を低下させ、特に手法 2 及び手法 3 は話者類似度についても若干の改善を達成したことから、提案手法によるベースラインからの改善が達成できたと考える。また、HuBERT の事前学習済み重みを用いることは仮説に反して有効ではなく、アンサンブル手法についてもその効果は顕著なものではなかった。さらに、特に手法 2 から手法 5 は損失関数の重み係数である λ_{ssl^d} による性能の変化が著しく、ここで最適値を選択できなければベースラインである手法 1 から改善しないこともわかった。よって、提案手法におけるベースラインからの改善に寄与した可能性のあるポイントは、以下でまとめられる。

1. HuBERT 中間特徴量を入力とし、メルスペクトログラムと HuBERT 離散特徴量を推定するネットワークを導入したこと
2. 上記のネットワーク構造を HuBERT Transformer としたこと
3. ネットワークをランダム初期化した上で学習させたこと
4. 最適な λ_{ssl^d} の値を発見できたこと

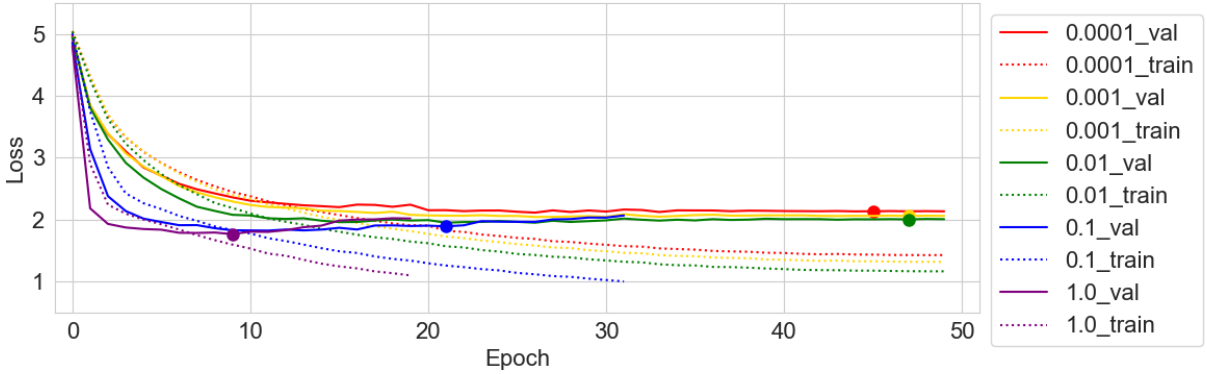
本実験からは、HuBERT Transformer をランダム初期化する必要性と、λ_{ssl^d} の値のチューニングを行う必要性が明らかとなった。しかし、入力特徴量やネットワーク構造については HuBERT を用いることを前提とした実験条件であったことから、検討できていない。実験結果から、もはや HuBERT に依存する必要性は無くなっているため、入力が HuBERT 中間特徴量でなくても良いし、ネットワークも任意に選択できる。入力については、例えば最終予測値であるメルスペクトログラムや HuBERT 離散特徴量を採用しても実装は可能であるが、これによって性能が変化するのであれば、HuBERT 中間特徴量が良い入力特徴量であると考えられる。ネットワークについては、Transformer 自体が多くの場合有効であるため現状で適切なものとなっている可能性もあるが、検討の余地はある。こういった点が、今後の検討課題として挙げられる。

表 4.3: 損失関数の重み係数 λ_{ssl^d} による客観評価指標の比較

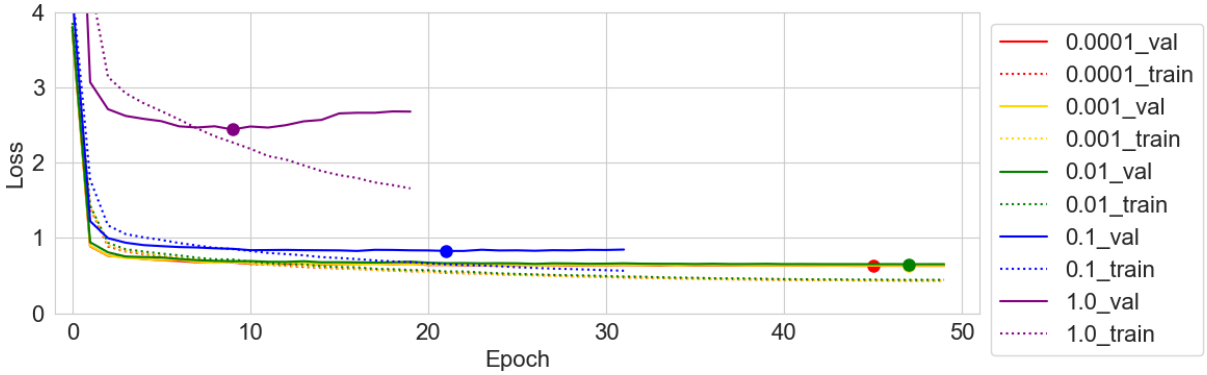
手法	λ_{ssl^d}	WER [%]	話者類似度	最良エポック	L_{mel}	L_{ssl^d}	L_{ssl^i}
1	0.0001	57.3	0.833	45	0.631	2.134	
1	0.001	54.6	0.836	47	0.633	2.057	
1	0.01	56.2	0.837	47	0.633	2.003	
1	0.1	58.2	0.784	21	0.639	1.890	
1	1	59.0	0.685	9	0.677	1.764	
1-proposed	0.0001	55.4	0.841	47	0.631	2.172	0.234
1-proposed	0.001	55.1	0.842	45	0.628	2.069	0.233
1-proposed	0.01	53.4	0.843	43	0.626	1.981	0.233
1-proposed	0.1	54.6	0.809	25	0.634	1.904	0.236
1-proposed	1	58.7	0.698	11	0.669	1.771	0.245
2	0.0001	60.6	0.852	35	0.631	2.500	
2	0.001	56.7	0.829	18	0.631	2.470	
2	0.01	54.4	0.847	25	0.627	2.110	
2	0.1	45.3	0.840	24	0.620	1.929	
2	1	45.5	0.712	12	0.652	1.788	
3	0.0001	58.5	0.860	49	0.631	2.547	
3	0.001	55.0	0.850	22	0.633	2.391	
3	0.01	56.0	0.848	21	0.628	2.049	
3	0.1	45.8	0.848	15	0.620	1.992	
3	1	46.9	0.763	13	0.644	1.842	
4	0.0001	60.1	0.841	22	0.633	2.585	
4	0.001	57.1	0.839	21	0.632	2.527	
4	0.01	56.8	0.860	27	0.628	2.115	
4	0.1	44.2	0.778	10	0.625	1.865	
4	1	48.1	0.685	7	0.655	1.758	
5	0.0001	57.8	0.861	28	0.635	2.582	
5	0.001	57.2	0.865	45	0.630	2.300	
5	0.01	55.7	0.870	45	0.624	2.084	
5	0.1	45.5	0.849	27	0.619	1.940	
5	1	46.2	0.753	15	0.637	1.781	



(a) メルスペクトログラムの MAE Loss（式 (4.1) の L_{mel}）

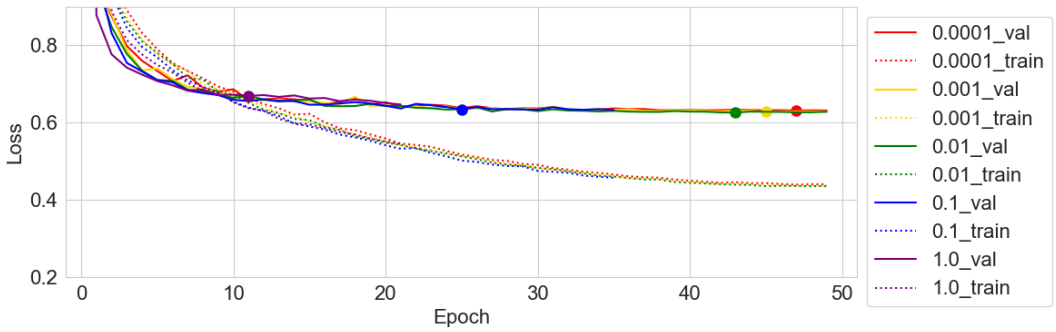


(b) HuBERT 離散特徴量の Cross Entropy Loss（式 (4.1) の L_{ssl^d}）

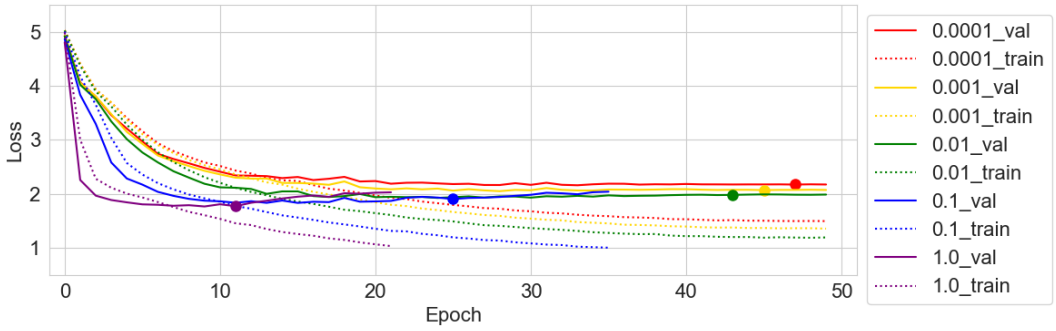


(c) 損失の合計値（式 (4.1) の L）

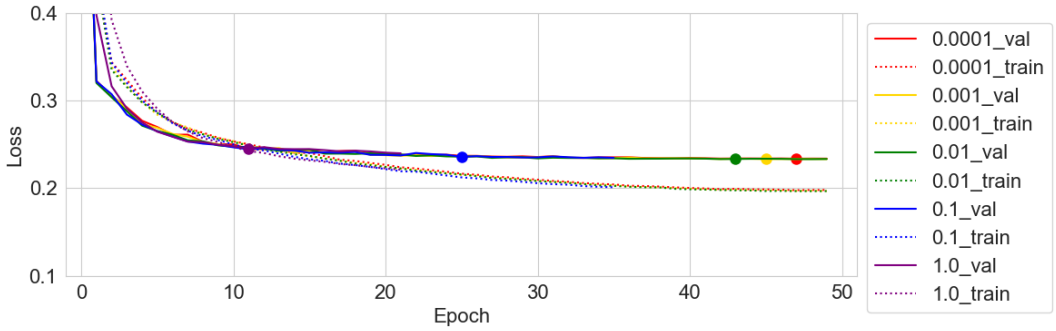
図 4.5: 手法 1 における学習曲線



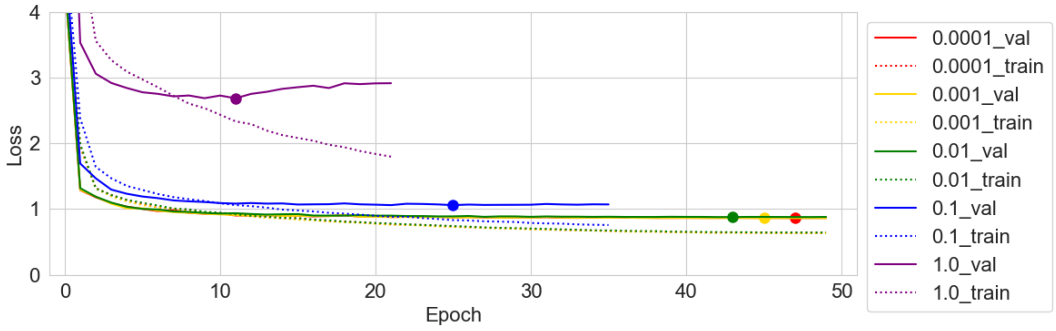
(a) メルスペクトログラムの MAE Loss（式 (4.1) の L_{mel}）



(b) HuBERT 離散特徴量の Cross Entropy Loss（式 (4.1) の L_{ssl^d}）

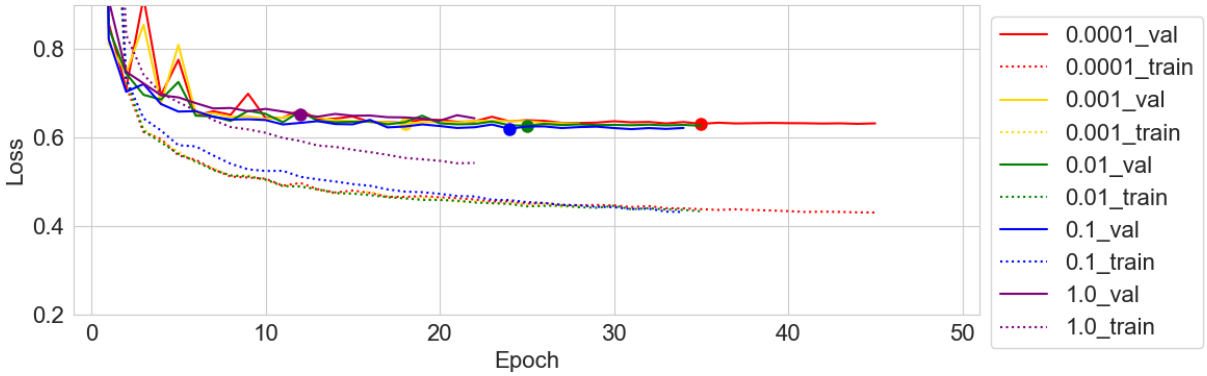


(c) HuBERT 中間特徴量の MAE Loss（式 (4.1) の L_{ssl^d}）

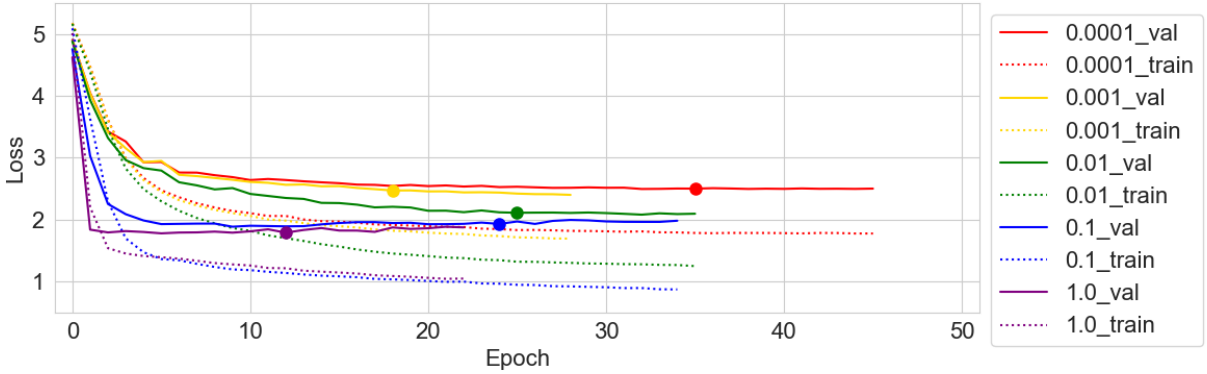


(d) 損失の合計値（式 (4.1) の L）

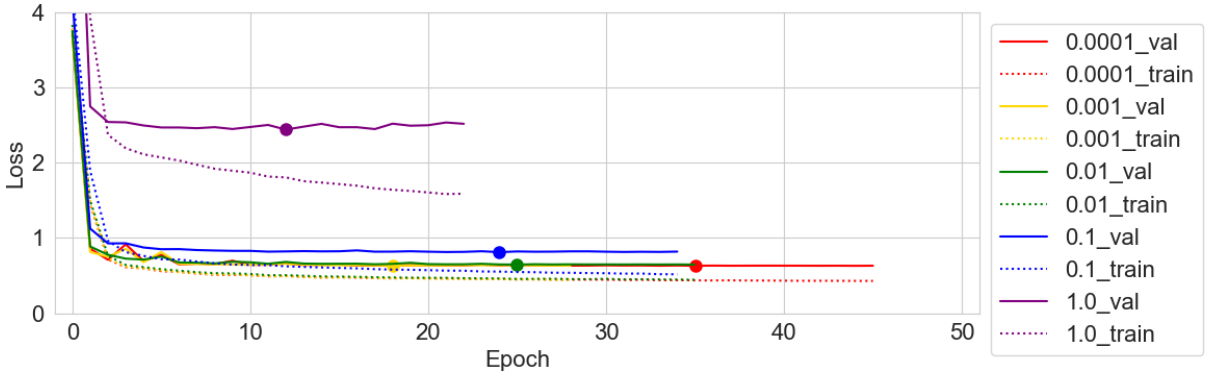
図 4.6: 手法 1-proposed における学習曲線



(a) メルスペクトログラムの MAE Loss（式 (4.1) の L_{mel}）

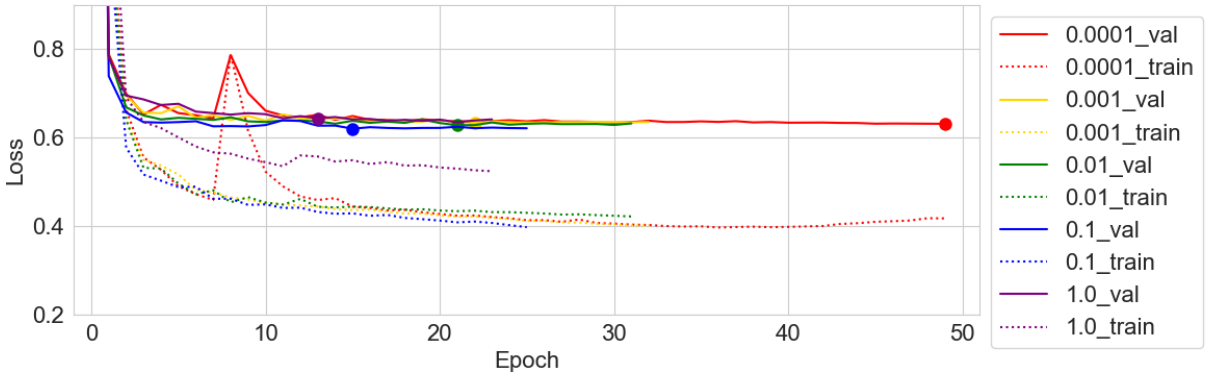


(b) HuBERT 離散特徴量の Cross Entropy Loss（式 (4.1) の L_{ssl^d}）

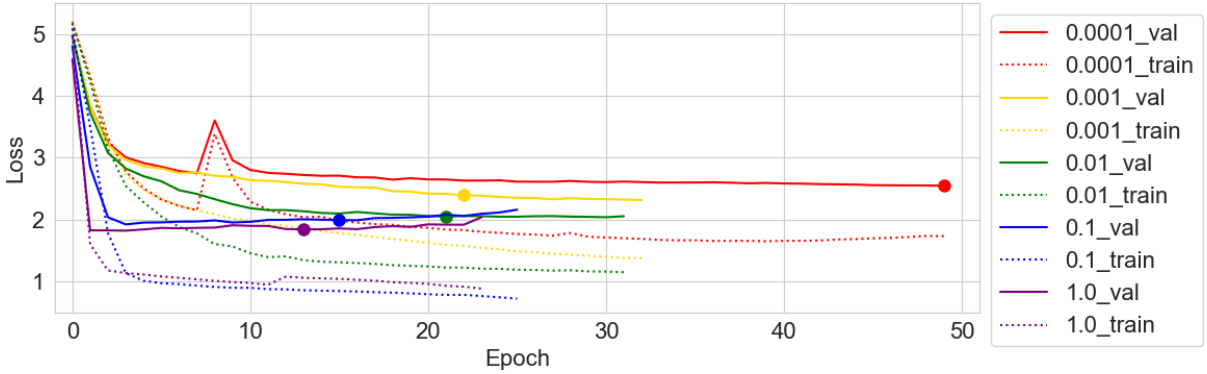


(c) 損失の合計値（式 (4.1) の L）

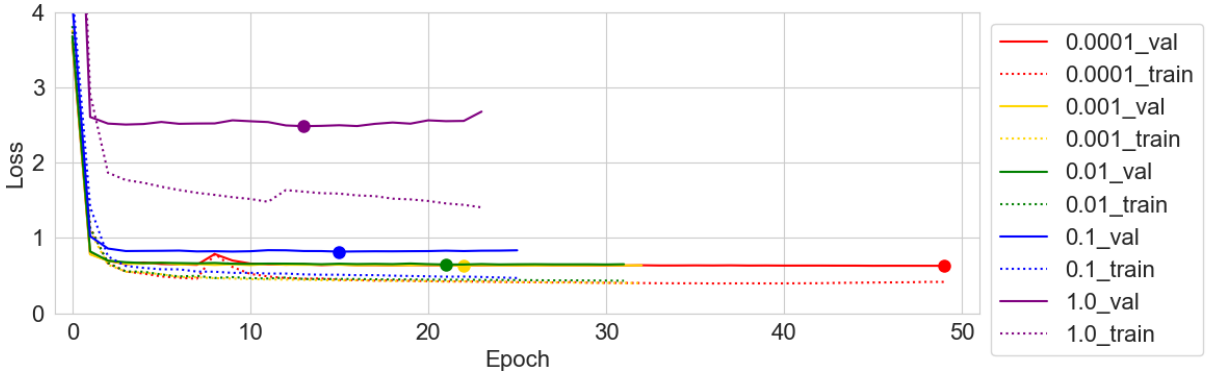
図 4.7: 手法 2 における学習曲線



(a) メルスペクトログラムの MAE Loss（式 (4.1) の L_{mel}）

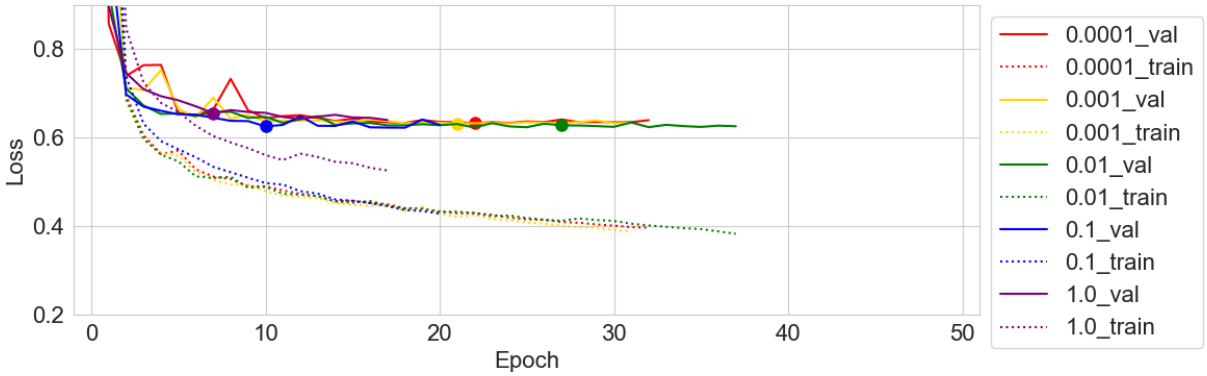


(b) HuBERT 離散特徴量の Cross Entropy Loss（式 (4.1) の L_{ssl^d}）

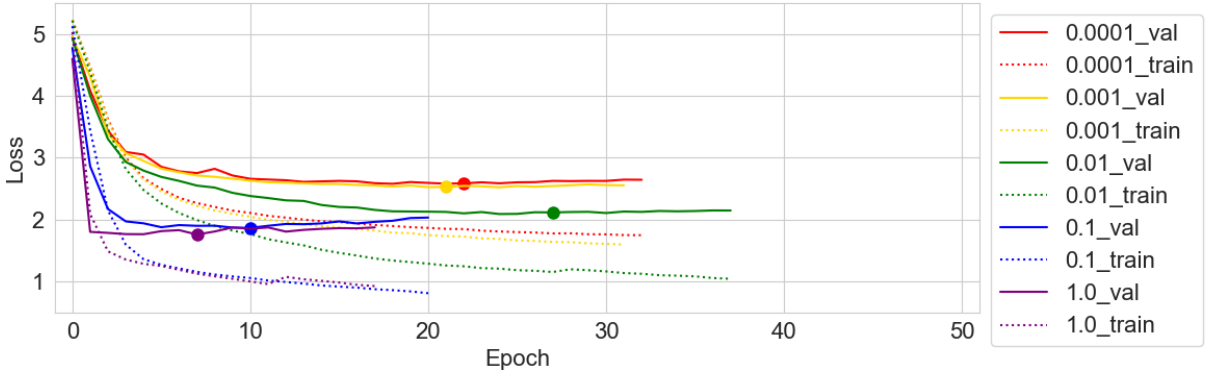


(c) 損失の合計値（式 (4.1) の L）

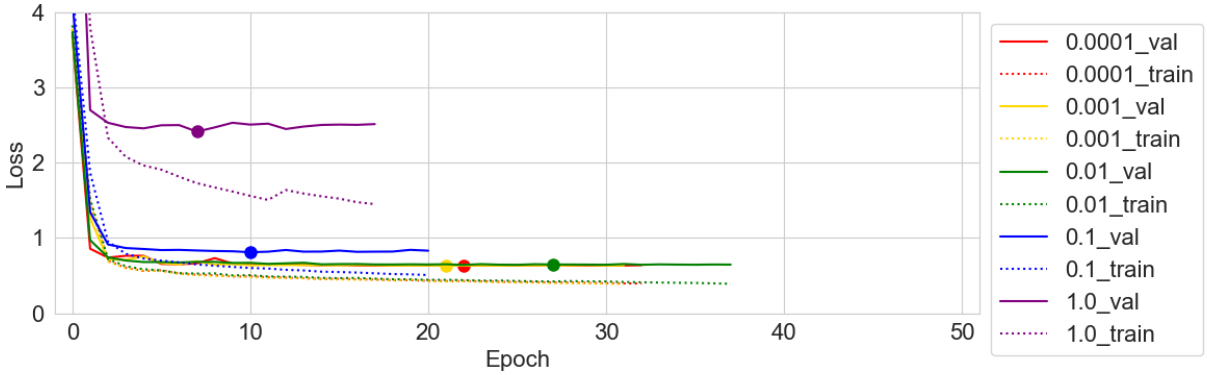
図 4.8: 手法 3 における学習曲線



(a) メルスペクトログラムの MAE Loss（式 (4.1) の L_{mel}）

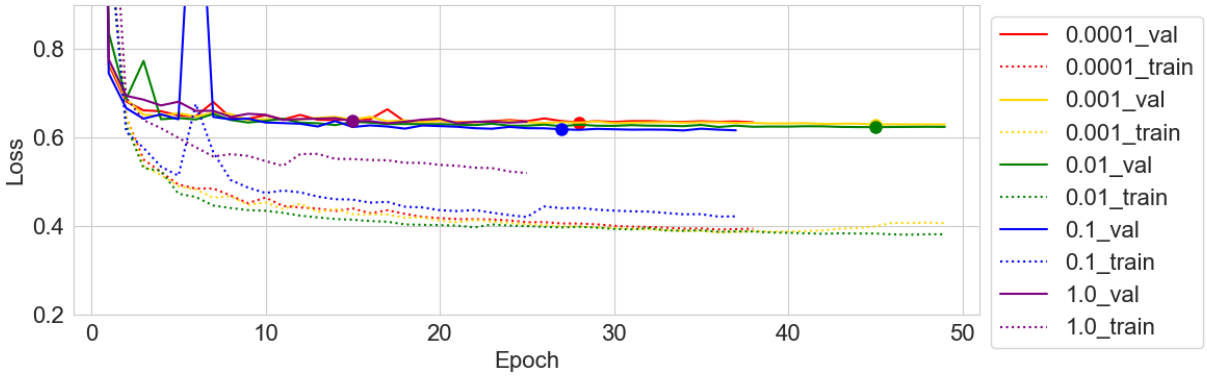


(b) HuBERT 離散特徴量の Cross Entropy Loss（式 (4.1) の L_{ssl^d}）

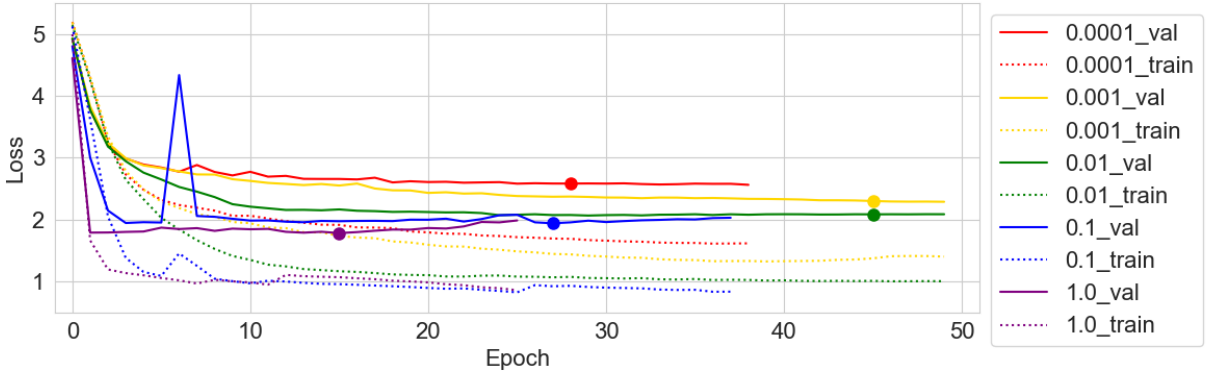


(c) 損失の合計値（式 (4.1) の L）

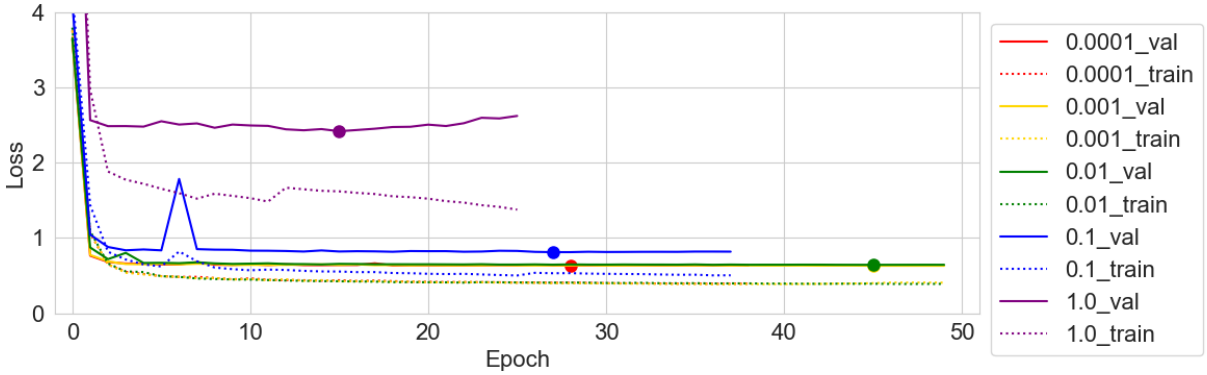
図 4.9: 手法 4 における学習曲線



(a) メルスペクトログラムの MAE Loss（式 (4.1) の L_{mel}）



(b) HuBERT 離散特徴量の Cross Entropy Loss（式 (4.1) の L_{ssl^d}）



(c) 損失の合計値（式 (4.1) の L）

図 4.10: 手法 5 における学習曲線

表 4.4: 最適なチューニングをした場合における手法ごとの比較

手法	詳細	WER [%]	話者類似度
1	ベースライン	55.3	0.841
2	ランダム初期化 Transformer	46.2	0.844
3	ランダム初期化 Transformer + アンサンブル	<u>45.8</u>	<u>0.851</u>
4	事前学習あり Transformer	47.4	0.827
5	事前学習あり Transformer + アンサンブル	47.4	0.810
6	分析合成	4.8	0.944
7	原音声	4.5	1.000

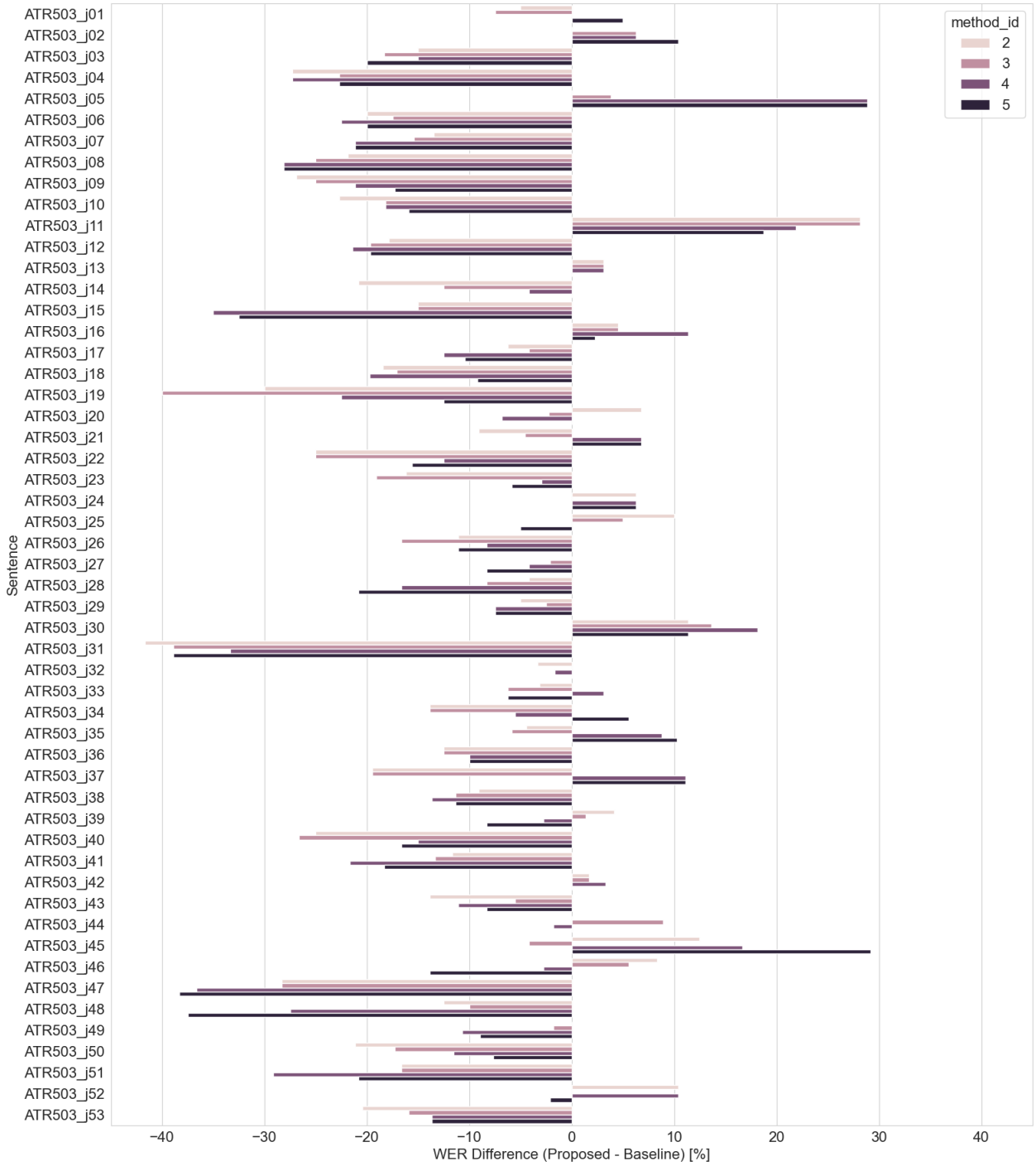


図 4.11: テストデータにおける発話文章ごとの WER の比較

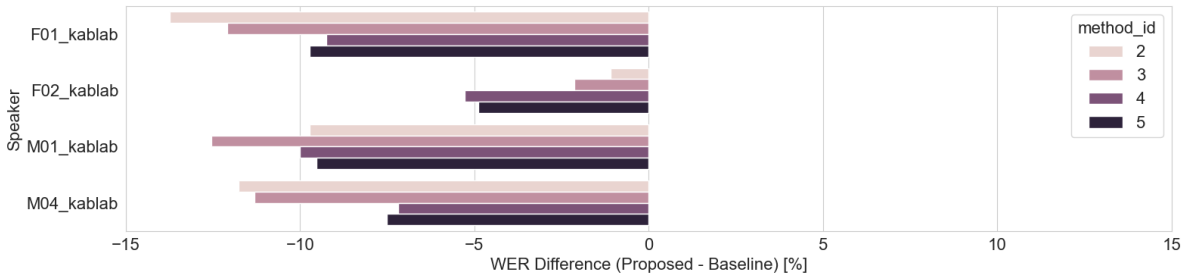


図 4.12: テストデータにおける話者ごとの WER の比較

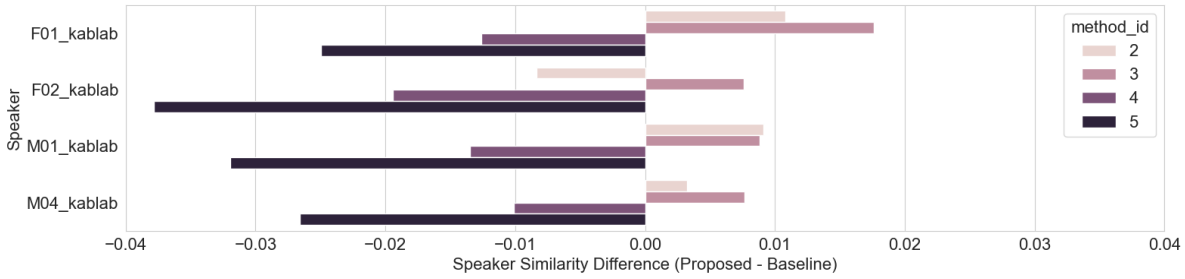


図 4.13: テストデータにおける話者ごとの話者類似度の比較

4.3.2 主観評価

4.4 まとめ

5　結論

謝辞

参考文献

[1] Triantafyllos Afouras, Joon Son Chung, and Andrew Zisserman. Lrs3-ted: a large-scale dataset for visual speech recognition. *arXiv preprint arXiv:1809.00496*, 2018.

[2] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. Voxceleb2: Deep speaker recognition. *arXiv preprint arXiv:1806.05622*, 2018.

[3] Bowen Shi, Wei-Ning Hsu, Kushal Lakhotia, and Abdelrahman Mohamed. Learning audio-visual speech representation by masked multimodal cluster prediction. *arXiv preprint arXiv:2201.02184*, 2022.

[4] Qiushi Zhu, Long Zhou, Ziqiang Zhang, Shujie Liu, Binxing Jiao, Jie Zhang, Lirong Dai, Daxin Jiang, Jinyu Li, and Furu Wei. Vatlm: Visual-audio-text pre-training with unified masked prediction for speech representation learning. *IEEE Transactions on Multimedia*, 2023.

[5] Alexandros Haliassos, Pingchuan Ma, Rodrigo Mira, Stavros Petridis, and Maja Pantic. Jointly learning visual and auditory speech representations from raw data. *arXiv preprint arXiv:2212.06246*, 2022.

[6] Jiachen Lian, Alexei Baevski, Wei-Ning Hsu, and Michael Auli. Av-data2vec: Self-supervised learning of audio-visual speech representations with contextualized target representations. In *2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 1–8. IEEE, 2023.

[7] Alexandros Haliassos, Andreas Zinonos, Rodrigo Mira, Stavros Petridis, and Maja Pantic. Braven: Improving self-supervised pre-training for visual and auditory speech recognition. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 11431–11435. IEEE, 2024.

[8] Minsu Kim, Joanna Hong, and Yong Man Ro. Lip-to-speech synthesis in the wild with multi-task learning. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2023.

[9] Jeongsoo Choi, Minsu Kim, and Yong Man Ro. Intelligible lip-to-speech synthesis with speech units. *arXiv preprint arXiv:2305.19603*, 2023.

[10] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM transactions on audio, speech, and language processing*, Vol. 29, pp. 3451–3460, 2021.

[11] Wei-Ning Hsu, Tal Remez, Bowen Shi, Jacob Donley, and Yossi Adi. Revise: Self-supervised speech resynthesis with visual input for universal and generalized speech regeneration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18795–18805, 2023.

[12] Neha Sahipjohn, Neil Shah, Vishal Tambrahalli, and Vineet Gandhi. Robustl2s: Speaker-specific lip-to-speech synthesis exploiting self-supervised representations. In *2023 Asia Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pp. 1492–1499. IEEE, 2023.

[13] Jeong Hun Yeo, Minsu Kim, Jeongsoo Choi, Dae Hoe Kim, and Yong Man Ro. Akvsr: Audio knowledge empowered visual speech recognition by compressing audio knowledge of a pretrained model. *IEEE Transactions on Multimedia*, 2024.

[14] Xize Cheng, Tao Jin, Linjun Li, Wang Lin, Xinyu Duan, and Zhou Zhao. Opensr: Open-modality speech recognition via maintaining multi-modality alignment. *arXiv preprint arXiv:2306.06410*, 2023.

[15] Yasser Abdelaziz Dahou Djilali, Sanath Narayan, Haithem Boussaid, Ebtessam Almazrouei, and Merouane Debbah. Lip2vec: Efficient and robust visual speech recognition via latent-to-latent visual to audio representation mapping. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 13790–13801, 2023.

[16] Xubo Liu, Egor Lakomkin, Konstantinos Vougioukas, Pingchuan Ma, Honglie Chen, Ruiming Xie, Morrie Doulaty, Niko Moritz, Jachym Kolar, Stavros Petridis, et al. Synthvsr: Scaling up visual speech recognition with synthetic supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18806–18815, 2023.

[17] Pingchuan Ma, Alexandros Haliassos, Adriana Fernandez-Lopez, Honglie Chen, Stavros Petridis, and Maja Pantic. Auto-avsr: Audio-visual speech recognition with automatic labels. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2023.

[18] Oscar Chang, Hank Liao, Dmitriy Serdyuk, Ankit Shahy, and Olivier Siohan. Conformer is all you need for visual speech recognition. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 10136–10140. IEEE, 2024.

[19] Andreas Zinonos, Alexandros Haliassos, Pingchuan Ma, Stavros Petridis, and Maja Pantic. Learning cross-lingual visual speech representations. In *ICASSP 2023-2023 IEEE*

International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1–5. IEEE, 2023.

[20] Ariel Ephrat, Inbar Mosseri, Oran Lang, Tali Dekel, Kevin Wilson, Avinatan Hassidim, William T Freeman, and Michael Rubinstein. Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation. *arXiv preprint arXiv:1804.03619*, 2018.

[21] Elizabeth Salesky, Matthew Wiesner, Jacob Bremerman, Roldano Cattoni, Matteo Negri, Marco Turchi, Douglas W Oard, and Matt Post. The multilingual tedx corpus for speech recognition and translation. *arXiv preprint arXiv:2102.01757*, 2021.

[22] Ya Zhao, Rui Xu, and Mingli Song. A cascade sequence-to-sequence model for chinese mandarin lip reading. In *Proceedings of the 1st ACM International Conference on Multimedia in Asia*, pp. 1–6, 2019.

[23] Minsu Kim, Jeong Hun Yeo, Jeongsoo Choi, and Yong Man Ro. Lip reading for low-resource languages by learning and combining general speech knowledge and language-specific knowledge. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15359–15371, 2023.

[24] Jeong Hun Yeo, Minsu Kim, Shinji Watanabe, and Yong Man Ro. Visual speech recognition for low-resource languages with automatic labels from whisper model. *arXiv preprint arXiv:2309.08535*, 2023.

[25] 田口史郎. ”深層学習を用いたデータ駆動型調音・音声間変換に関する研究”. 九州大学大学院芸術工学府芸術工学専攻 博士論文, 2021.

[26] 江崎蓮. ”深層学習を用いた口唇動画・音声変換に関する調査”. 九州大学大学院芸術工学府芸術工学専攻 修士論文, 2022.

[27] Ji-Hoon Kim, Jaehun Kim, and Joon Son Chung. Let there be sound: Reconstructing high quality speech from silent videos. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38, pp. 2759–2767, 2024.

[28] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[29] Kyunghyun Cho. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[30] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[31] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[32] I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[33] Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[34] Jimmy Lei Ba. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[35] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, Vol. 29, , 2016.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, Vol. 15, No. 1, pp. 1929–1958, 2014.

[37] Li Wan, Quan Wang, Alan Papir, and Ignacio Lopez Moreno. Generalized end-to-end loss for speaker verification. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4879–4883. IEEE, 2018.

[38] Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in neural information processing systems*, Vol. 33, pp. 17022–17033, 2020.

[39] Yoshinori Sagisaka, Kazuya Takeda, M Abel, Shigeru Katagiri, Tetsuo Umeda, and Hisao Kuwabara. A large-scale japanese speech database. In *ICSLP*, pp. 1089–1092, 1990.

[40] T Okamoto, Y Shiga, and H Kawai. Hi-fi-captain: High-fidelity and high-capacity conversational speech synthesis corpus developed by nict, 2023.

[41] Shinnosuke Takamichi, Kentaro Mitsui, Yuki Saito, Tomoki Koriyama, Naoko Tanji, and Hiroshi Saruwatari. Jvs corpus: free japanese multi-speaker voice corpus. *arXiv preprint arXiv:1908.06248*, 2019.

[42] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem?(and a dataset of 230,000 3d facial landmarks). In *Proceedings of the IEEE international conference on computer vision*, pp. 1021–1030, 2017.

[43] Yukiya Hono, Kentaro Mitsui, and Kei Sawada. rinna/japanese-hubert-base.

[44] Kei Sawada, Tianyu Zhao, Makoto Shing, Kentaro Mitsui, Akio Kaga, Yukiya Hono, Toshiaki Wakatsuki, and Koh Mitsuda. Release of pre-trained models for the Japanese language. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 13898–13905, 5 2024. Available at: https://arxiv.org/abs/2404.01657.

[45] Ankita Pasad, Bowen Shi, and Karen Livescu. Comparative layer-wise analysis of self-supervised speech models. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2023.

[46] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pp. 28492–28518. PMLR, 2023.