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手段として、人々の日常生活において重要な役割を果たしている。しかし、癌などの病気で喉頭を摘出すると、声帯振動による音声生成が不可能になり、従来の発声手段を失う。このような場合の代用音声手法として、電気式人工喉頭や食道発声、シャント発声があるが、自然で聞き取りやすい音声を生成することが難しかったり、習得に訓練を必要としたり、器具の交換のための定期的な手術を必要とするといった課題がある。これに対して本研究では、新たな代用音声手法として、深層学習を活用して口唇の動きと音声波形の関係性を学習することで、口唇の動きから音声を合成するアプローチを提案する。この手法により、訓練や手術を必要とせずとも、自然な声でのコミュニケーションを可能にすることを目指す。

近年の動画音声合成では、LRS3[1] や VoxCeleb2[2] などの大規模データセットを活用して構築された、自己教師あり学習（Self Supervised Learning; SSL）モデルを FineTuning することの有効性が示されている。特に、近年の動画音声合成においては AVHuBERT[3] というモデルが動画からの特徴抽出器として採用される場合が多い。AVHuBERT は、動画と音声の対応関係を Masked Prediction という自己教師あり学習方法により捉えたモデルであり、Modality Dropout という工夫により、自己教師あり学習時には動画と音声の両方を入力とするものの、FineTuning 時には動画あるいは音声のみを入力とできる柔軟性を持ち合わせている。これに加え、従来の動画音声においては動画からの予測対象としてメルスペクトログラムが選択されることが多かったが、近年ではこれにテキストや、音声 SSL モデルである HuBERT を利用して得られた離散特徴量を合わせて予測対象とする、マルチタスク学習の有効性が示されている [4, 5]。

1.2　目的

本研究では、動画音声合成モデルによって得られる合成音声の品質が依然として低く、自然音声に迫る合成音の実現が困難である点を課題とする。この課題に対し、近年高い精度を達成した手法 [5] における、「AVHuBERT を利用したメルスペクトログラムと HuBERT 離散特徴量を予測対象とするマルチタスク学習手法」をベースラインとして採用し、この性能を上回る新たなモデルを提案することを目的とする。

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。

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

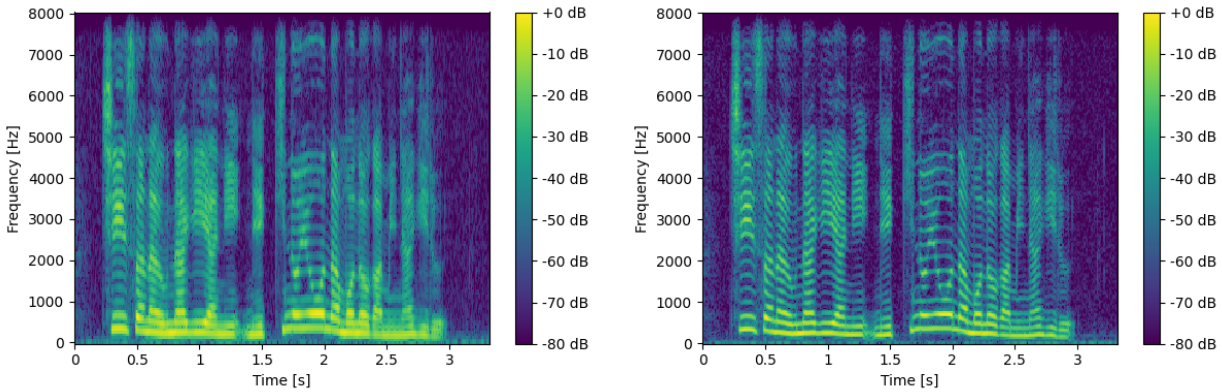
$$= |X[k]|e^{j\angle \mathrm{X[k]}} \tag{2.2}$$

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、

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

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

と表される。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



(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

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。

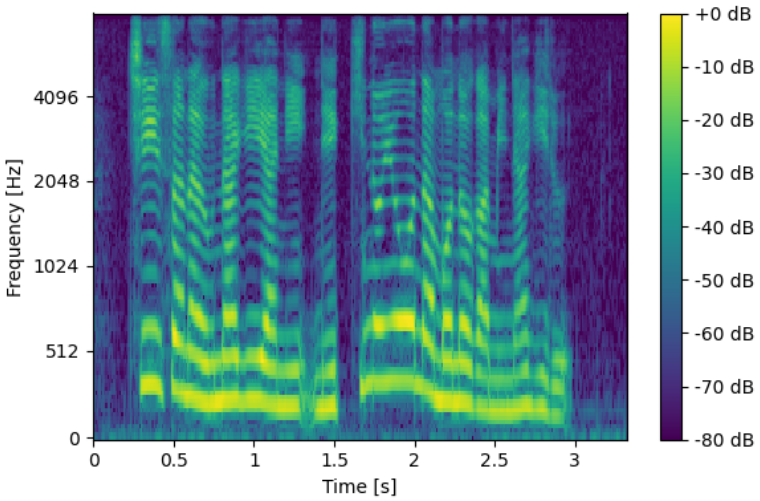


図 2.2: 「小さな鰻屋に、熱気のようなものがみなぎる」と発話した音声に対する対数メルスペクトログラム

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。

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

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、大量のパラメータによる表現力により、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、DNN の構成要素及び、構築した DNN の学習方法について説明する。

3.1　DNN の構成要素

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、$\boldsymbol{b} \in \mathbb{R}^m$ はバイアスベクトルである。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、所望の出力に次元を合わせるのに便利である。

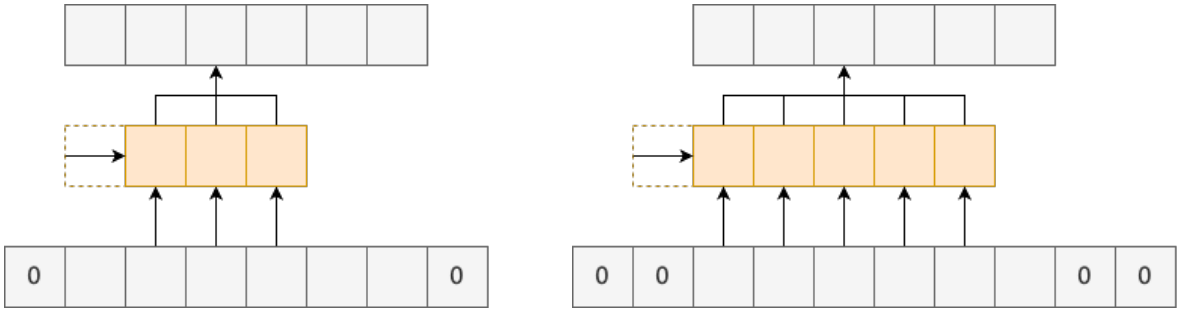
3.1.2　畳み込み層

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 $\boldsymbol{x} \in \mathbb{R}^{C_{\text{in}} \times T_{\text{in}}}$ に対し、出力 $\boldsymbol{y} \in \mathbb{R}^{C_{\text{out}} \times T_{\text{out}}}$ は、$\boldsymbol{y}_k[i]$ を出力テンソルの k 番目のチャンネルの i 番目の成分とすると、

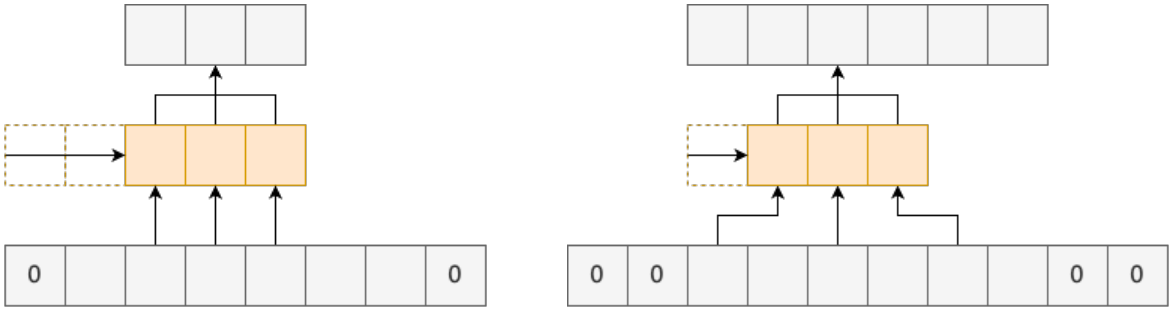
$$\boldsymbol{y}_k[i] = b_k + \sum_{c=0}^{C_{\text{in}}-1} \sum_{m=0}^{M-1} \boldsymbol{x}_c \left[i - \left\lfloor \frac{M}{2} \right\rfloor + m \right] \boldsymbol{W}_{k,c}[m] \tag{3.2}$$

で計算される。ここで、M がカーネルサイズ、$\boldsymbol{x}_c \left[i - \left\lfloor \frac{M}{2} \right\rfloor + m \right]$ が入力テンソルの c 番目のチャンネルの $\left[i - \left\lfloor \frac{M}{2} \right\rfloor + m \right]$ 番目の成分、$\boldsymbol{W}_{k,c}[m]$ が出力チャンネル k と入力チャンネル c に対応するカーネルの m 番目の成分である。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)



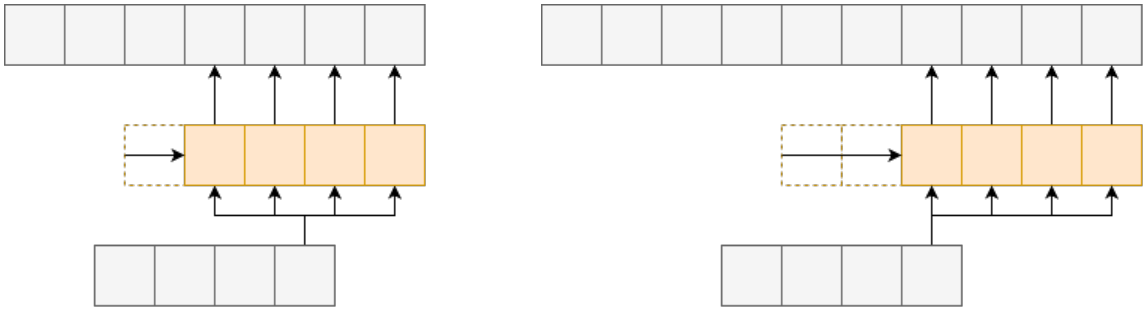
(c) (k, s, d) = (3, 2, 1)　(d) (k, s, d) = (3, 1, 2)

図 3.1: ある入出力チャンネル間における一次元畳み込み層の処理。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り、これによって考慮できる入力特徴量の範囲が定まる。二つ目は、ストライドであり、これによってカーネルのシフト幅を設定できる。三つ目は、ダイレーションであり、これは畳み込み演算における入力特徴量の間隔を表す。ダイレーションを大きくすることで、カーネルサイズが同じでも考慮できる入力特徴量の範囲を広げることが可能である。また、出力系列長 T_{out} を入力系列長 T_{in} の整数倍に保つには、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れば良い。図 3.1 に、ある入出力チャンネル間における一次元畳み込み層の処理を示す。

3.1.3　転置畳み込み層

転置畳み込み層は、畳み込み層の逆演算に対応する層であり、主に入力のアップサンプリングに使用される。図 3.2 に、ある入出力チャンネル間における一次元転置畳み込み層の処理を示す。転置畳み込み層では、i 番目の入力とカーネルの積を計算し、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、複数の入力から計算された出力が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。アップサンプリングを行いたい場合は、ストライドを



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

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

2 以上とすれば良い。図 3.2b にカーネルサイズを 4、ストライドを 2 とした場合を示す。この時、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保つためには、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、出力の両端の削除数である。上述の例ではパディングを 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出力に非線形性を与えるための関数である。これにより、DNN 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、活性化関数への入力を $x \in \mathbb{R}$ として、代表的なものを述べる。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、$|x - 0|$ が大きくなるほどさらに小さくなることがわかる。DNN の各重みは、損失関数の勾配を利用することで更新されるから、シグモイド関数以前の層の重みにおける勾配は、シグモイド関数の一階導関数の値が乗算された結果となる。従って、シグモイド関数は一階導関数の値が小さくなりがちであるから、それ以前の層における勾配もこれに伴い小さくなってしまうことで、重みの更新が進みづらくなる課題がある。これ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。tanh 関数の値域は $[-1,1]$ となっており、図 3.3b より $|x-0|$ が小さいところではシグモイド関数より一階導関数の値が大きくなっていることがわかる。しかし、$|x-0|$ が大きくなればシグモイド関数と同様に一回導関数の値が小さく、勾配消失のリスクを抱えていることがわかる。

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して振る舞うが、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が起こりづらくなっている。ReLU は現在、標準的な活性化関数として広く用いられている。しかし、ReLU への入力が 0 以下の値を取るとき、ReLU 出力に対する入力の勾配は 0 になるから ReLU 以前の層の重みに対する勾配も全て 0 になってしまい、重みが更新されないことで学習が遅くなる可能性がある。

四つ目は、LeakyReLU[6] 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。ReLU と比較すると、0 以下の入力に対しても 0 でない値を出力し、微分係数も 0 にならない点が異なっている。これにより、任意の入力に対して重みの更新を可能にすることで、ReLU の問題点を解消した。

五つ目は、PReLU[7] である。これは、LeakyReLU と似た活性化関数であるが、LeakyReLU のパラメータ a を学習可能にすることで、その他の層と合わせてデータに対する最適化が可能となったことが特徴である。

六つ目は、GELU[8] である。GELU は

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

で与えられる。ここで、

$$\Phi(x) = P(X \leq x),\ X \sim \mathcal{N}(0,1) \tag{3.13}$$

である。GELU の一階導関数は、

$$\frac{d}{dx}\mathrm{GELU}(x) = \Phi(x) + \frac{x}{\sqrt{2\pi}}\exp(-\frac{x^2}{2}) \tag{3.14}$$

となる。GELU は、ReLU が入力に対して 0 あるいは 1 を確定的にかける活性化関数と捉えた上で、これを入力に依存した確率的な挙動に変更したものである。実際、$m \sim \mathrm{Bernoulli}(\Phi(x))$ とすると、

$$\begin{aligned}\mathrm{GELU}(x) &= x\Phi(x) &(3.15)\\ &= 1x \cdot \Phi(x) + 0x \cdot (1 - \Phi(x)) &(3.16)\\ &= \mathbb{E}[mx] &(3.17)\end{aligned}$$

と見ることができる。

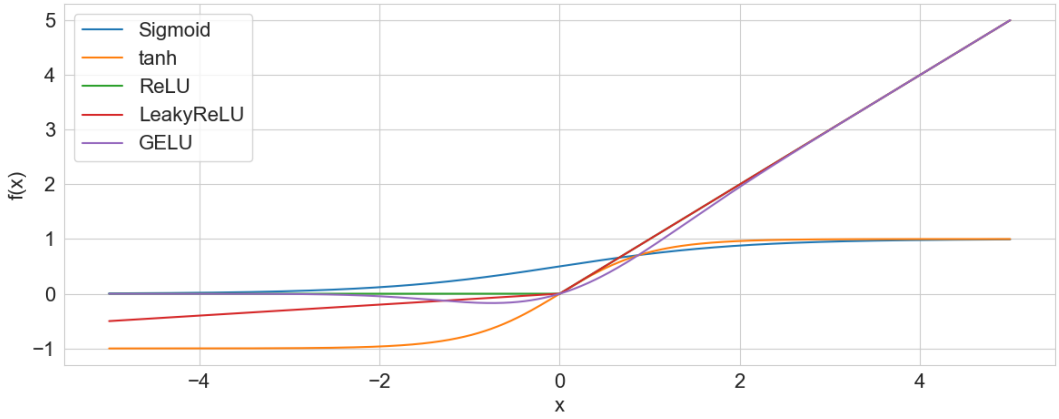
3.1.5 再帰型ニューラルネットワーク

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。

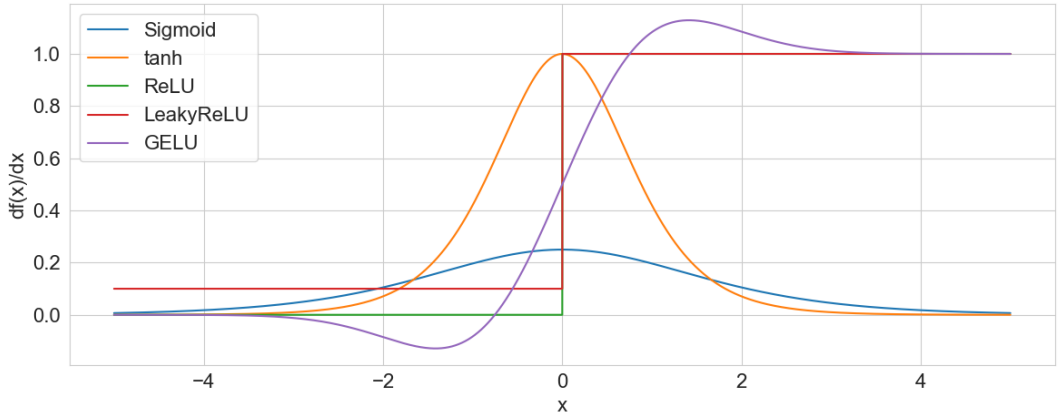
近年よく用いられる RNN として、長・短期記憶（Long Short-Time Memory; LSTM）[9] がある。LSTM では入力ゲート、忘却ゲート、出力ゲートの 3 つを持ち、これらゲートによってネットワーク内部の情報の取捨選択を行うことで、長い系列データからの学習を可能にした。LSTM のネットワーク内部で行われる計算を以下に示す。

$$\begin{aligned}\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_f) &(3.18)\\ \boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_i) &(3.19)\\ \tilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_h[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_h) &(3.20)\\ \boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t &(3.21)\\ \boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_o) &(3.22)\\ \boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) &(3.23)\end{aligned}$$

ここで、$\boldsymbol{x}_t \in \mathbb{R}^n$ は時刻 t の入力、$\boldsymbol{f}_t \in [0,1]^m$ は忘却ゲートの出力、$\boldsymbol{i}_t \in [0,1]^m$ は入力ゲートの出力、$\boldsymbol{c}_t \in [-1,1]^m$ が時刻 t におけるセルの状態、$\boldsymbol{o}_t \in [0,1]^m$ が出力ゲートの出力、$\boldsymbol{h}_t \in [-1,1]^m$ が時刻 t における隠れ状態を表す。また、$[\cdot,\cdot]$ は次元方向の結合演算子、$\odot$ は要素積を表す。



(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

$\boldsymbol{W}_f, \boldsymbol{W}_i, \boldsymbol{W}_h, \boldsymbol{W}_o \in \mathbb{R}^{m\times(n+m)}$ は学習可能な重み行列、$\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_h, \boldsymbol{b}_o \in \mathbb{R}^m$ は学習可能な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）[10] 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。

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_z) \tag{3.24}$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_r) \tag{3.25}$$

$$\tilde{\boldsymbol{h}_t} = \tanh(\boldsymbol{W}_h[\boldsymbol{x}_t, \boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}] + \boldsymbol{b}_h) \tag{3.26}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}_t} \tag{3.27}$$

ここで、$\boldsymbol{x}_t \in \mathbb{R}^n$ が時刻 t における入力、$\boldsymbol{z}_t \in [0,1]^m$ が更新ゲートの出力、$\boldsymbol{r}_t \in [0,1]^m$ がリセットゲートの出力、$\boldsymbol{h}_t \in [-1,1]^m$ が時刻 t における隠れ状態を表す。また、$\boldsymbol{W}_z, \boldsymbol{W}_r, \boldsymbol{W}_h \in \mathbb{R}^{m\times(n+m)}$ は学習可能な重み行列、$\boldsymbol{b}_z, \boldsymbol{b}_r, \boldsymbol{b}_h \in \mathbb{R}^m$ は学習可能な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[11] 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 $\boldsymbol{X} \in \mathbb{R}^{T\times d_{\text{model}}}$ 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、クエリ $\boldsymbol{Q} \in \mathbb{R}^{T\times d_k}$、キー $\boldsymbol{K} \in \mathbb{R}^{T\times d_k}$、バリュー $\boldsymbol{V} \in \mathbb{R}^{T\times d_v}$ の計算を行う。これは、

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \tag{3.28}$$

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K \tag{3.29}$$

$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V \tag{3.30}$$

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、$\boldsymbol{W_Q} \in \mathbb{R}^{d_{\text{model}}\times d_k}$ はクエリに対する重み行列、$\boldsymbol{W_K} \in \mathbb{R}^{d_{\text{model}}\times d_k}$ はキーに対する重み行列、$\boldsymbol{W_V} \in \mathbb{R}^{d_{\text{model}}\times d_v}$ はバリューに対する重み行列である。次に、クエリ $\boldsymbol{Q}$ とキー $\boldsymbol{K}$ を元にアテンション重みを求め、バリューに対する行列積を計算する。

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{3.31}$$

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 $\boldsymbol{Q}_i \in \mathbb{R}^{d_k}$ と、キーベクトル $\boldsymbol{K}_j \in \mathbb{R}^{d_k}$ の内積であり、$\boldsymbol{Q}_i$ と $\boldsymbol{K}_i$ の関連度を表していると

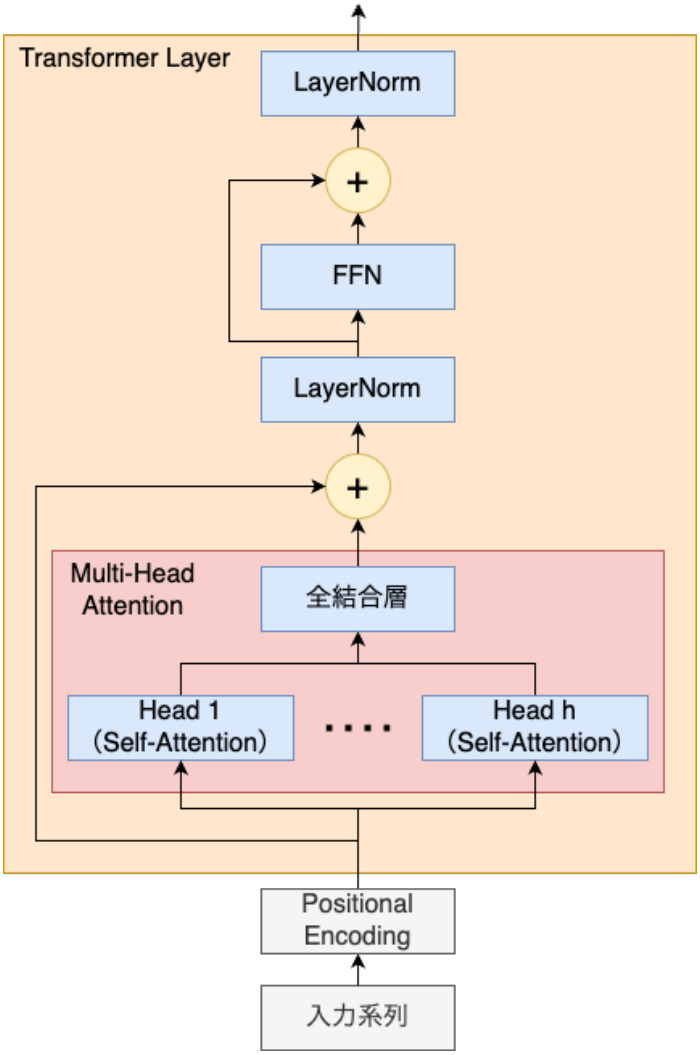


図 3.4: Transformer 層の構造

解釈できる。また、softmax 関数が行方向に適用されることで、i 番目のクエリベクトル $\boldsymbol{Q}_i$ から $\boldsymbol{K}_j$ $(j = 1, \ldots, T)$ 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 $\mathrm{softmax}\left(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d_k}\right)$ をバリュー $\boldsymbol{V}$ にかけることにより、各バリューベクトル $\boldsymbol{V}_i \in \mathbb{R}^{d_v}$ は、$\boldsymbol{Q}_i$ からキーへの注意度による重み付け和となる。

Transformer では、Self-Attention 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 が用いられる。ヘッド数を H とすると、各ヘッド h におけるクエリ $\boldsymbol{Q}^h \in \mathbb{R}^{T \times \frac{d_k}{H}}$、キー $\boldsymbol{K}^h \in \mathbb{R}^{T \times \frac{d_k}{H}}$、バリュー $\boldsymbol{V}^h \in \mathbb{R}^{T \times \frac{d_v}{H}}$ の計算は、

$$\boldsymbol{Q}^h = \boldsymbol{X}\boldsymbol{W}_Q^h \tag{3.32}$$

$$\boldsymbol{K}^h = \boldsymbol{X}\boldsymbol{W}_K^h \tag{3.33}$$

$$\boldsymbol{V}^h = \boldsymbol{X}\boldsymbol{W}_V^h \tag{3.34}$$

で与えられる。ここで、$\boldsymbol{W}_Q^h \in \mathbb{R}^{d_{\text{model}} \times \frac{d_k}{H}}$ は h 番目のヘッドのクエリに対する重み行列、$\boldsymbol{W}_K^h \in \mathbb{R}^{d_{\text{model}} \times \frac{d_k}{H}}$ は h 番目のヘッドのキーに対する重み行列、$\boldsymbol{W}_V^h \in \mathbb{R}^{d_{\text{model}} \times \frac{d_v}{H}}$ は h 番目のヘッドのバ

リューに対する重み行列である。この後の処理も同様に、

$$\mathrm{Attention}(\boldsymbol{Q}^h, \boldsymbol{K}^h, \boldsymbol{V}^h) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}^h(\boldsymbol{K}^h)^\top}{\sqrt{d_k}/H}\right)\boldsymbol{V}^h \tag{3.35}$$

で行われる。その後、すべてのヘッドからの出力を結合し、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、

$$\mathrm{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{Concat}(\mathrm{head}^1, \ldots, \mathrm{head}^H)\boldsymbol{W}_o \tag{3.36}$$

と表される。ここで、$\mathrm{head}^h \in \mathbb{R}^{T\times\frac{d_v}{H}}$ が各ヘッドから出力された Attention スコア、$\mathrm{Concat}(\mathrm{head}^1, \ldots, \mathrm{head}^H) \in \mathbb{R}^{T\times d_v}$ が全ヘッドの Attention スコアを結合した特徴量、$\boldsymbol{W}_o \in \mathbb{R}^{d_v\times d_{\mathrm{model}}}$ 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 $\boldsymbol{Y} \in \mathbb{R}^{T\times d_{\mathrm{model}}}$ とすると、

$$\boldsymbol{Y} = \mathrm{LayerNorm}\left(\mathrm{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) + \boldsymbol{X}\right) \tag{3.37}$$

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、

$$\mathrm{FFN}(\boldsymbol{Y}) = \mathrm{ReLU}(\boldsymbol{Y}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{3.38}$$

で与えられる。ここで、$\boldsymbol{W}_1 \in \mathbb{R}^{d_{\mathrm{model}}\times d_{\mathrm{ff}}}$、$\boldsymbol{W}_2 \in \mathbb{R}^{d_1\times d_{\mathrm{model}}}$ は学習可能な重み行列、$\boldsymbol{b}_1 \in \mathbb{R}^{d_{\mathrm{ff}}}$、$\boldsymbol{b}_2 \in \mathbb{R}^{d_{\mathrm{model}}}$ 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、

$$\mathrm{PE}_{\mathrm{pos},2d} = \sin\left(\frac{\mathrm{pos}}{10000^{2d/d_{\mathrm{model}}}}\right) \tag{3.39}$$

$$\mathrm{PE}_{\mathrm{pos},2d+1} = \cos\left(\frac{\mathrm{pos}}{10000^{2d/d_{\mathrm{model}}}}\right) \tag{3.40}$$

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

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

$$L_{MAE}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \frac{1}{TD} \sum_{t=1}^{T} \sum_{d=1}^{D} |\boldsymbol{Y}_{t,d} - \hat{\boldsymbol{Y}}_{t,d}| \tag{3.41}$$

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 $\boldsymbol{y} \in [0,1]^C$、DNN による推定結果を $\hat{\boldsymbol{y}} \in [0,1]^C$ とすると、Cross Entropy Loss は

$$L_{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{c=1}^{C} \boldsymbol{y}_c \log(\hat{\boldsymbol{y}}_c) \tag{3.42}$$

で与えられる。ここで、$\boldsymbol{y}, \hat{\boldsymbol{y}}$ はどちらもクラスに対する確率分布であり、

$$\sum_{c=1}^{C} \boldsymbol{y}_c = 1, \quad \sum_{c=1}^{C} \hat{\boldsymbol{y}}_c = 1 \tag{3.43}$$

を満たす。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 $\boldsymbol{Y} \in [0,1]^{T \times C}$、DNN による推定結果を $\hat{\boldsymbol{Y}} \in [0,1]^{T \times C}$ とすると、Cross Entropy Loss は

$$L_{CE}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{c=1}^{C} \boldsymbol{Y}_{t,c} \log(\hat{\boldsymbol{Y}}_{t,c}) \tag{3.44}$$

で与えられる。ここで、$\boldsymbol{Y}_t, \hat{\boldsymbol{Y}}_t \in [0,1]^C$ $(t = 1, \ldots, T)$ はそれぞれクラスに対する確率分布である。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。

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla L(\boldsymbol{\theta}) \tag{3.45}$$

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る。

まず、$\boldsymbol{W}_{i,j}^{k} \in \mathbb{R}$ 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、$\boldsymbol{b}_{i}^{k} \in \mathbb{R}$ 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 $\boldsymbol{a}_{i}^{k} \in \mathbb{R}$ は、

$$\boldsymbol{a}_{i}^{k} = \boldsymbol{b}_{i}^{k} + \sum_{j=1}^{N_{k-1}} \boldsymbol{W}_{j,i}^{k} \boldsymbol{o}_{j}^{k-1} \tag{3.46}$$

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、$\boldsymbol{o}_{j}^{k-1} \in \mathbb{R}$ は $k-1$ 層目の j 番目のノードにおける出力 $\boldsymbol{a}_{j}^{k-1}$ に活性化関数 f を適用した結果を表す。すなわち、

$$\boldsymbol{o}_{j}^{k-1} = f(\boldsymbol{a}_{j}^{k-1}) \tag{3.47}$$

で与えられる。ここで、式 (3.46) に対し、$\boldsymbol{W}_{0,i}^{k} = \boldsymbol{b}_{i}^{k}$ とし、$\boldsymbol{o}_{0}^{k-1} = 1$ とすれば、

$$\boldsymbol{a}_{i}^{k} = \boldsymbol{b}_{i}^{k} + \sum_{j=1}^{N_{k-1}} \boldsymbol{W}_{j,i}^{k} \boldsymbol{o}_{j}^{k-1} = \sum_{j=0}^{N_{k-1}} \boldsymbol{W}_{j,i}^{k} \boldsymbol{o}_{j}^{k-1} \tag{3.48}$$

と整理できる。

次に、DNN の学習用データセットを $\{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$ 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 $\boldsymbol{x}_n$ 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、損失関数の値を $L(\boldsymbol{y_n}, \hat{\boldsymbol{y}}_n; \boldsymbol{\theta})$ 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 $\boldsymbol{W}_{i,j}^k$ についての勾配は、

$$\frac{\partial L}{\partial \boldsymbol{W}_{i,j}^k} = \frac{\partial}{\partial \boldsymbol{W}_{i,j}^k}\left(\frac{1}{N_{\text{batch}}}\sum_{n=1}^{N_{\text{batch}}} L(\boldsymbol{y_n}, \hat{\boldsymbol{y}}_n; \boldsymbol{\theta})\right) \tag{3.49}$$

$$= \frac{1}{N_{\text{batch}}}\sum_{n=1}^{N_{\text{batch}}} \frac{\partial}{\partial \boldsymbol{W}_{i,j}^k} L(\boldsymbol{y_n}, \hat{\boldsymbol{y}}_n; \boldsymbol{\theta}) \tag{3.50}$$

$$= \frac{1}{N_{\text{batch}}}\sum_{n=1}^{N_{\text{batch}}} \frac{\partial L_n}{\partial \boldsymbol{W}_{i,j}^k} \tag{3.51}$$

となる。ここで、式 (3.51) では $L(\boldsymbol{y_n}, \hat{\boldsymbol{y}}_n; \boldsymbol{\theta}) = L_n$ とおいた。式 (3.51) 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、$\partial L_n / \partial \boldsymbol{W}_{i,j}^k$ は、

$$\frac{\partial L_n}{\partial \boldsymbol{W}_{i,j}^k} = \frac{\partial L_n}{\partial \boldsymbol{a}_j^k}\frac{\partial \boldsymbol{a}_j^k}{\partial \boldsymbol{W}_{i,j}^k} \tag{3.52}$$

となる。ここで、式 (3.48) より

$$\frac{\partial \boldsymbol{a}_j^k}{\partial \boldsymbol{W}_{i,j}^k} = \frac{\partial}{\partial \boldsymbol{W}_{i,j}^k}\left(\sum_{l=0}^{N_{k-1}} \boldsymbol{W}_{l,j}^k \boldsymbol{o}_l^{k-1}\right) = \boldsymbol{o}_i^{k-1} \tag{3.53}$$

であり、

$$\delta_j^k = \frac{\partial L_n}{\partial \boldsymbol{a}_j^k} \tag{3.54}$$

とおけば、式 (3.52) は

$$\frac{\partial L_n}{\partial \boldsymbol{W}_{i,j}^k} = \delta_j^k \boldsymbol{o}_i^{k-1} \tag{3.55}$$

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

$$\frac{\partial L_n}{\partial \boldsymbol{W}_{i,j}^L} = \delta_j^L \boldsymbol{o}_i^{L-1} \tag{3.56}$$

$$= \frac{\partial L_n}{\partial \boldsymbol{a}_j^L}\boldsymbol{o}_i^{L-1} \tag{3.57}$$

$$= \frac{\partial}{\partial \boldsymbol{a}_j^L} L(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n; \boldsymbol{\theta}) \cdot \boldsymbol{o}_i^{L-1} \tag{3.58}$$

$$= \frac{\partial}{\partial \boldsymbol{a}_j^L} L(\boldsymbol{y}_n, f(\boldsymbol{a}^L)_n; \boldsymbol{\theta}) \cdot \boldsymbol{o}_i^{L-1} \tag{3.59}$$

$$= \frac{\partial}{\partial f(\boldsymbol{a}^L)_n} L(\boldsymbol{y}_n, f(\boldsymbol{a}^L)_n; \boldsymbol{\theta}) \cdot \frac{\partial}{\partial \boldsymbol{a}_j^L} f(\boldsymbol{a}^L)_n \cdot \boldsymbol{o}_i^{L-1} \tag{3.60}$$

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

$$\frac{\partial L_n}{\partial \boldsymbol{W}_{i,j}^{k}} = \delta_j^k \boldsymbol{o}_i^{k-1} \tag{3.61}$$

$$= \frac{\partial L_n}{\partial \boldsymbol{a}_j^k} \boldsymbol{o}_i^{k-1} \tag{3.62}$$

$$= \boldsymbol{o}_i^{k-1} \sum_{l=0}^{N_{k+1}} \frac{\partial L_n}{\partial \boldsymbol{a}_l^{k+1}} \frac{\partial \boldsymbol{a}_l^{k+1}}{\partial \boldsymbol{a}_j^k} \tag{3.63}$$

$$= \boldsymbol{o}_i^{k-1} \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} \frac{\partial \boldsymbol{a}_l^{k+1}}{\partial \boldsymbol{a}_j^k} \tag{3.64}$$

$$= \boldsymbol{o}_i^{k-1} \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} \frac{\partial}{\partial \boldsymbol{a}_j^k} \left(\sum_{m=0}^{N_k} \boldsymbol{W}_{m,l}^{k+1} \boldsymbol{o}_m^k \right) \tag{3.65}$$

$$= \boldsymbol{o}_i^{k-1} \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} \frac{\partial}{\partial \boldsymbol{a}_j^k} \left(\sum_{m=0}^{N_k} \boldsymbol{W}_{m,l}^{k+1} f(\boldsymbol{a}_m^k) \right) \tag{3.66}$$

$$= \boldsymbol{o}_i^{k-1} \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} \boldsymbol{W}_{j,l}^{k+1} f'(\boldsymbol{a}_j^k) \tag{3.67}$$

$$= \boldsymbol{o}_i^{k-1} f'(\boldsymbol{a}_j^k) \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} \boldsymbol{W}_{j,l}^{k+1} \tag{3.68}$$

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 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、具体的な方法を四つ述べる。

一つ目は、L2 正則化である。L2 正則化は、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、L2 正則化を適用した場合における損失関数の値 $L_w(\boldsymbol{\theta})$ は

$$L_w(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 \tag{3.69}$$

で与えられる。重みの L2 ノルムの二乗が損失関数に加算されることで、重みの大きさが過大になることを防ぐ。正則化の程度は係数 λ で調整する。

二つ目は、Weight Decay である。Weight Decay は重みの絶対値が過大になることを防ぐ手法であり、重みの更新式を

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla L(\boldsymbol{\theta}) - \lambda' \boldsymbol{\theta} \tag{3.70}$$

とすることで与えられる。これは、特に通常の勾配降下法において L2 正則化を用いたとき、式 (3.69) における係数 λ を $\lambda = \lambda'/\eta$ とした場合に等しい [12]。実際、

$$\nabla L_w(\boldsymbol{\theta}) = \nabla L(\boldsymbol{\theta}) + \nabla \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 \tag{3.71}$$

$$= \nabla L(\boldsymbol{\theta}) + \frac{\lambda'}{2\eta} \cdot 2\boldsymbol{\theta} \tag{3.72}$$

$$= \nabla L(\boldsymbol{\theta}) + \frac{\lambda'}{\eta} \boldsymbol{\theta} \tag{3.73}$$

となり、このとき重みの更新式は式 (3.70) に一致する。

三つ目は、Dropout[13] 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。ここで、推論時は全ノードの出力を用いるため、学習時と挙動が異なる。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 $p \in [0, 1]$ とすると、学習時の Dropout 出力 $\boldsymbol{h}_{\text{train}} \in \mathbb{R}^D$ および推論時の Dropout 出力 $\boldsymbol{h}_{\text{infer}} \in \mathbb{R}^D$ は、

$$\boldsymbol{h}_{\text{train}} = \frac{\boldsymbol{h} \odot \boldsymbol{m}}{1 - p} \tag{3.74}$$

$$\boldsymbol{h}_{\text{infer}} = \boldsymbol{h} \tag{3.75}$$

で与えられる。ここで、$\boldsymbol{m} \in \{0, 1\}^D$ は各要素 $\boldsymbol{m}_d$ が確率 $1 - p$ で 1、確率 p で 0 をとる確率変数であり、$\odot$ は要素積を表す。式 (3.74) より、学習時は Dropout 出力を $1/(1 - p)$ 倍していることがわかる。この理由は、学習時の出力の期待値と推論時の出力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 $\boldsymbol{h}_{\text{train}}$ の期待値をとれば、

$$\mathbb{E}[\boldsymbol{h}_{\text{train}}] = \mathbb{E}\left[\frac{\boldsymbol{h} \odot \boldsymbol{m}}{1 - p}\right] \tag{3.76}$$

$$= \frac{\boldsymbol{h}}{1 - p} \mathbb{E}[\boldsymbol{m}] \tag{3.77}$$

$$= \frac{\boldsymbol{h}}{1 - p} (1 - p) \tag{3.78}$$

$$= \boldsymbol{h} = \boldsymbol{f}_{\text{infer}} \tag{3.79}$$

となる。Dropout は学習時に一部のニューロンを落とすことで、実質的に異なるネットワークを学習させていると解釈できる。よって、学習時の出力の期待値に推論時の出力を一致させることは、異なるネットワークの出力の平均をとり、アンサンブルモデルとして推論時の出力を得ていると考えられる。

四つ目は、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。

3.2.5 最適化手法

3.2.2 節において、DNN の重みとバイアス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、通常の勾配降下法に代わり、近年よく用いられる最適化手法として Adam[14] 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 $L(\boldsymbol{\theta})$ の勾配の一次モーメント、$\boldsymbol{m}_t$ が勾配の一次モーメントの指数移動平均、$\boldsymbol{v}_t$ が勾配の二次モーメントの指数移動平均である。$\hat{\boldsymbol{m}}_t$ および $\hat{\boldsymbol{v}}_t$ はそれぞれ、$\boldsymbol{m}_t$ および $\boldsymbol{v}_t$ 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、λ は L2 正則化の程度を調整するパラメータである。ここで、Adam では正則化として L2 正則化を採用した

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 $\boldsymbol{\theta}_t$

が、Weight Decay を採用した最適化手法として AdamW[12] がある。AdamW の計算過程をアルゴリズム 2 に示す。正則化が L2 正則化から Weight Decay に変わった点以外は同じである。

3.2.6 学習率のスケジューリング

3.2.5 節では、DNN の学習における学習率の重要性および、Adam や AdamW といった、学習過程で適応的に重みおよびバイアス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、学習率のスケジューリングは、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る。これは、より安定した学習を促すための調整や、より早く学習を収束させるのに役立つ手段である。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

$$\eta_t = \eta_0 \gamma^{\lfloor t/\text{step\_size} \rfloor} \tag{3.80}$$

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**
  $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_{t-1})$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_{t^2}$
  $\tilde{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1-\beta_1^t)$
  $\tilde{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1-\beta_2^t)$
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta(\tilde{\boldsymbol{m}}_t / (\sqrt{\tilde{\boldsymbol{v}}_t} + \epsilon) - \lambda \boldsymbol{\theta}_{t-1})$
end for
Return $\boldsymbol{\theta}_t$

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

$$\eta_t = \eta_0 \cdot \exp(-\gamma t) \tag{3.81}$$

で与えるスケジューラである。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ため、StepLRScheduler と比較して、学習率を連続的に変化させられる。

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

$$\eta_t = \begin{cases} \eta_{\min} + \left(\frac{t}{\text{warmup\_steps}}\right)(\eta_{\max} - \eta_{\min}) & \text{if } t < \text{warmup\_steps} \\ \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{(t-\text{warmup\_steps})\pi}{t_{\max} - \text{warmup\_steps}}\right)\right) & \text{if } t \geq \text{warmup\_steps} \end{cases} \tag{3.82}$$

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 warmup_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に学習率が低い状態から開始して、徐々に学習率を大きくすることで解の十分な探索を可能にし、その後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。

3.2.7 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）[15] が有効である。バッチ正規化は、

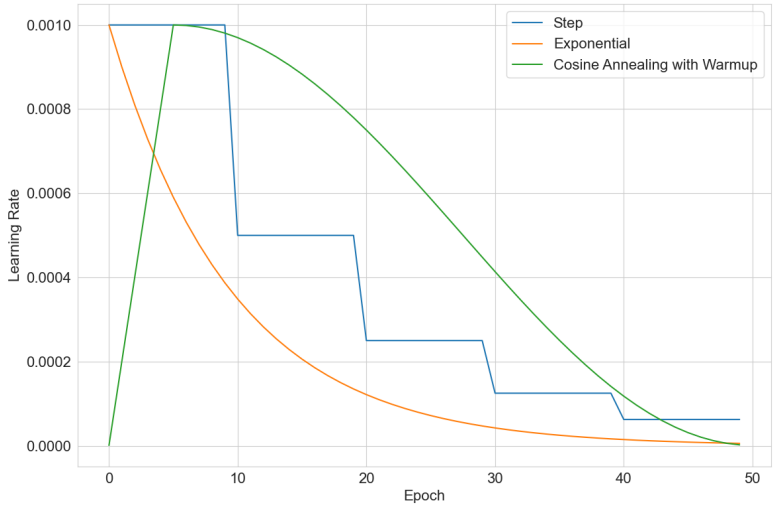


図 3.5: スケジューラによる学習率の変化

ミニバッチ学習を行う際にそのミニバッチ内における期待値と分散を次元ごとに計算し、特徴量を次元ごとに標準化する。バッチサイズを N、バッチ正規化への入力を $\boldsymbol{x}_i \in \mathbb{R}^D\ (1 \leq i \leq N)$ とすると、出力 $\boldsymbol{y}_i \in \mathbb{R}^D\ (1 \leq i \leq N)$ は

$$\boldsymbol{\mu}_{\text{batch}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i \tag{3.83}$$

$$\boldsymbol{\sigma}^2_{\text{batch}} = \frac{1}{N}\sum_{i=1}^{N} (\boldsymbol{x}_i - \boldsymbol{\mu}_{\text{batch}})^2 \tag{3.84}$$

$$\hat{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \boldsymbol{\mu}_{\text{batch}}}{\sqrt{\boldsymbol{\sigma}^2_{\text{batch}} + \epsilon}} \tag{3.85}$$

$$\boldsymbol{y}_i = \boldsymbol{\gamma}\hat{\boldsymbol{x}}_i + \boldsymbol{\beta} \tag{3.86}$$

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期待値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。$\boldsymbol{\gamma}$ の乗算と $\boldsymbol{\beta}$ の加算によって表現力を向上させており、実際 $\boldsymbol{\gamma} = \boldsymbol{\mu}_{\text{batch}}, \boldsymbol{\beta} = \boldsymbol{\sigma}_{\text{batch}}$ とすれば、バッチ正規化適用前の結果である $\boldsymbol{x}_i$ を $\hat{\boldsymbol{x}}_i$ から再度得ることも可能である。また、同じサンプルでも組み合わせられるミニバッチのサンプルによって統計量が変わり、この多少のノイズが正則化としての効果ももたらす。さらに、学習時は、サンプルの標準化に用いる統計量とは別に、期待値の移動平均と不偏分散の移動平均を計算しておく。推論時は学習終了時に得られたこれら移動平均の値を一貫して用いることで、確率的な挙動を持たないよう工夫されている。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一方、統計量の計算にミニバッチ内のサンプルを利用するため、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。また、テキストや音声といった時系列データを用いる場合、ミニバッチを構成するためには系列長を揃える必要があるためゼロパディングを行う。特に RNN においてバッチ正規化を用

いる場合、t ステップ目の出力にバッチ正規化を適用して $t+1$ ステップ目の入力に備える必要があるが、ゼロパディングによって人為的に系列量を揃えているから、ミニバッチ内の t ステップ目における統計量が実際のデータの分布からかけ離れたものになる可能性がある。こういった課題に対し、ミニバッチ内の各サンプル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）[16] が提案された。サンプルごとに次元方向に平均と分散を求める場合、レイヤー正規化の出力 $\boldsymbol{y}_i$ は

$$\mu_{\text{layer},i} = \frac{1}{D}\sum_{d=1}^{D} \boldsymbol{x}_{i,d} \tag{3.87}$$

$$\sigma^2_{\text{layer},i} = \frac{1}{D}\sum_{d=1}^{D} (\boldsymbol{x}_{i,d} - \mu_{layer})^2 \tag{3.88}$$

$$\hat{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \mu_{\text{layer},i}}{\sqrt{\sigma^2_{\text{layer},i} + \epsilon}} \tag{3.89}$$

$$\boldsymbol{y}_i = \boldsymbol{\gamma}\hat{\boldsymbol{x}}_i + \boldsymbol{\beta} \tag{3.90}$$

で与えられる。ここで、$\mu_{\text{layer},i}, \sigma_{\text{layer},i} \in \mathbb{R}$ は各サンプル i の平均および分散である。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、DNN 内部の特徴量を標準化することで、学習を安定させるための手法であった。一方、DNN 内のある層の重みを再パラメータ化することで学習を安定させる手法として、重み正規化（Weight Normalization）[17] がある。これは、ある層の重みベクトル $\boldsymbol{\theta}$ を、

$$\boldsymbol{\theta} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} g \tag{3.91}$$

のように単位ベクトル $\boldsymbol{v}/\|\boldsymbol{v}\|$（ベクトルの向き）とスカラー g（ベクトルの大きさ）に再パラメータ化するものである。学習時は重みの更新を $\boldsymbol{v}$ と g それぞれ別々に行う。重み正規化は、バッチ正規化やレイヤー正規化と同様に学習の安定化に役立つが、計算に入力特徴量の系列長が依存しない。そのため、例えば音声波形など系列長が非常に長くなりがちなデータを扱う場合、計算コストを下げながら同様の効果を狙える手段だと考えられる。実際、本研究にて音声波形を直接用いるモデルのベースとなった手法である HiFi-GAN[18] の一部レイヤーでは、重み正規化が採用されている。

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、学習が不安定になる可能性がある。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、

$$\boldsymbol{g} \leftarrow \boldsymbol{g} \cdot \frac{c}{\max\{\|\boldsymbol{g}\|_2, c\}} \tag{3.92}$$

で与えられる。

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

$$\boldsymbol{g}_{\text{accum}} \leftarrow \boldsymbol{g}_{\text{accum}} + \boldsymbol{g}_t \tag{3.93}$$

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\boldsymbol{g}_{\text{accum}}}{k} \tag{3.94}$$

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 $1/k$ 倍して平均をとることで、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。ここで、自己教師あり学習とは、教師ラベルのないデータからデータ自体の特徴を学習するために、データ自体から教師ラベルを作成して教師あり学習を行う方法である。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が、データ自体から教師ラベルを作成するという点で、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。ここでは特に、本研究で用いた自己教師あり学習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、データの一部をマスクした上でモデルに入力し、マスクされた領域をモデルに予測させることで学習を行う。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 $\boldsymbol{X}^{\text{masked}} \in \mathbb{R}^{T \times D}$ の各時刻 t における値 $\boldsymbol{X}_t^{\text{masked}}$ は、

$$\boldsymbol{X}_t^{\text{masked}} = \begin{cases} \boldsymbol{X}_t & \text{if } t \notin \mathcal{M} \\ \boldsymbol{m} & \text{if } t \in \mathcal{M} \end{cases} \tag{3.95}$$

である。$\boldsymbol{m} \in \mathbb{R}^D$ はマスクベクトルである。ここで、音声自体の特徴を学習する HuBERT[19] や、動画と音声の対応関係を学習する AVHuBERT[3] といった自己教師あり学習モデルでは、

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することで教師ラベルを作成する。クラスタ数を C としたクラスタリングによって得られる教師ラベルを $\boldsymbol{X}^{\text{discretized}} \in \{0,1\}^{T\times C}$ とする。各時刻 t における教師ラベル $\boldsymbol{X}_t^{\text{discretized}}$ は、正解クラスが 1、それ以外が 0 となった One-hot ベクトルである。この時、モデルの予測値 $\hat{\boldsymbol{X}} \in [0,1]^{T\times C}$ に対する損失は、

$$L_{CE}(\boldsymbol{X}^{\text{discretized}}, \hat{\boldsymbol{X}}) = -\frac{1}{|\mathcal{M}|}\sum_{t\in\mathcal{M}}\sum_{c=1}^{C}\boldsymbol{X}_{t,c}^{\text{discretized}}\log(\hat{\boldsymbol{X}}_{t,c}) \tag{3.96}$$

で与えられる。これは、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モデルと比較して高い精度を達成できることが示されている。

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 [20] 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。

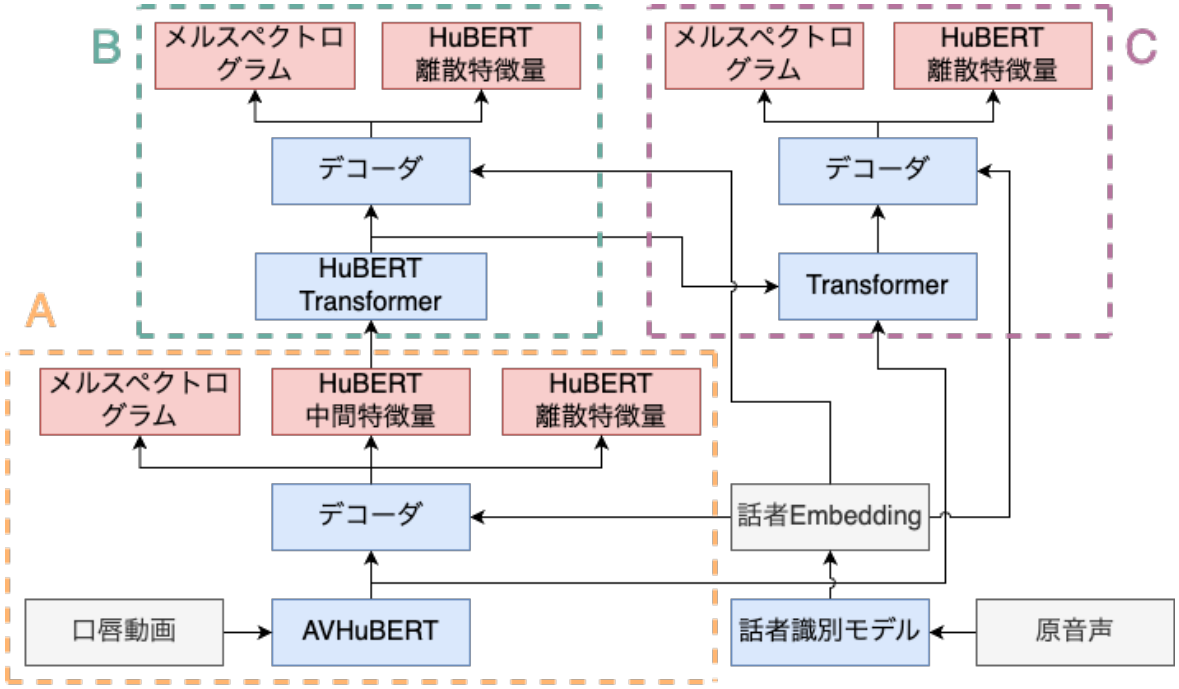


図 4.1: 提案するネットワークの構造

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 [5] 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

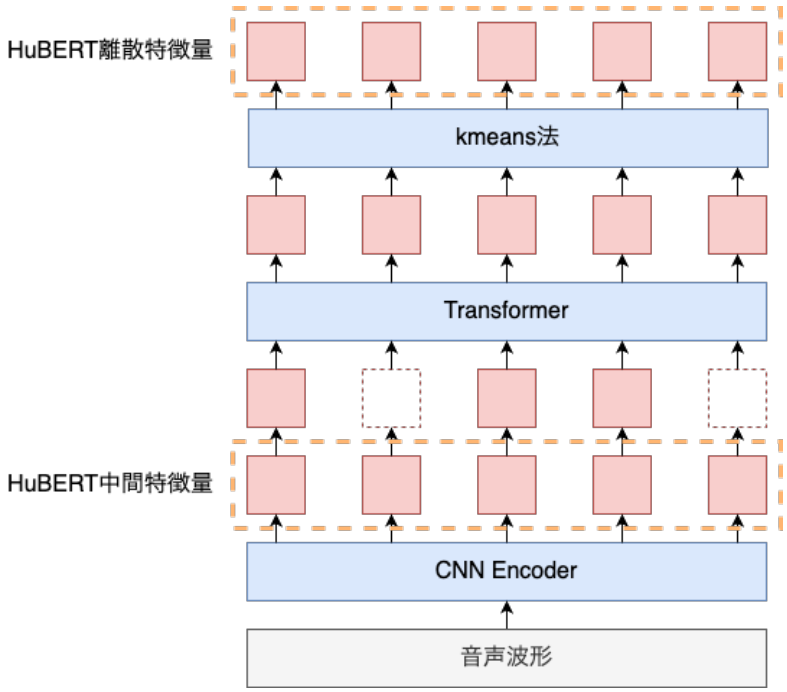


図 4.2: HuBERT 中間特徴量と HuBERT 離散特徴量の取得位置

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 [5] 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[18] 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、

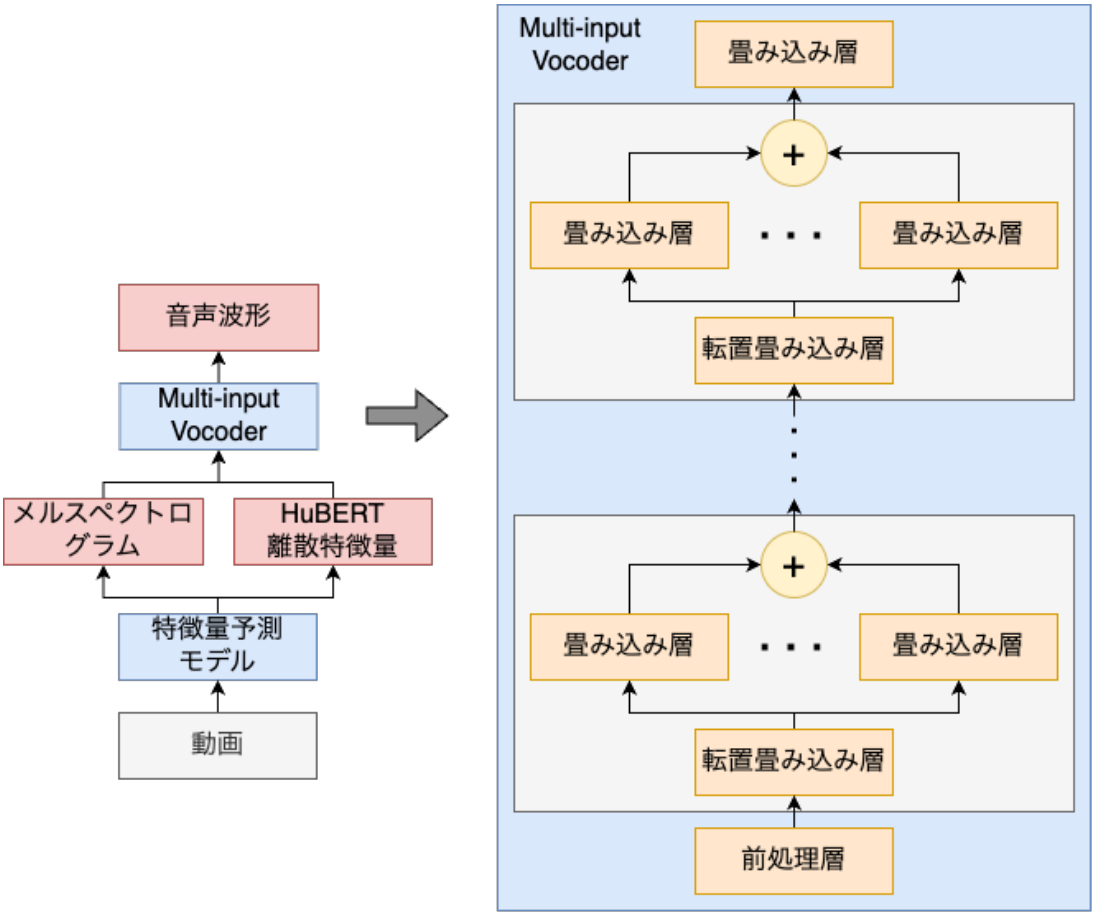


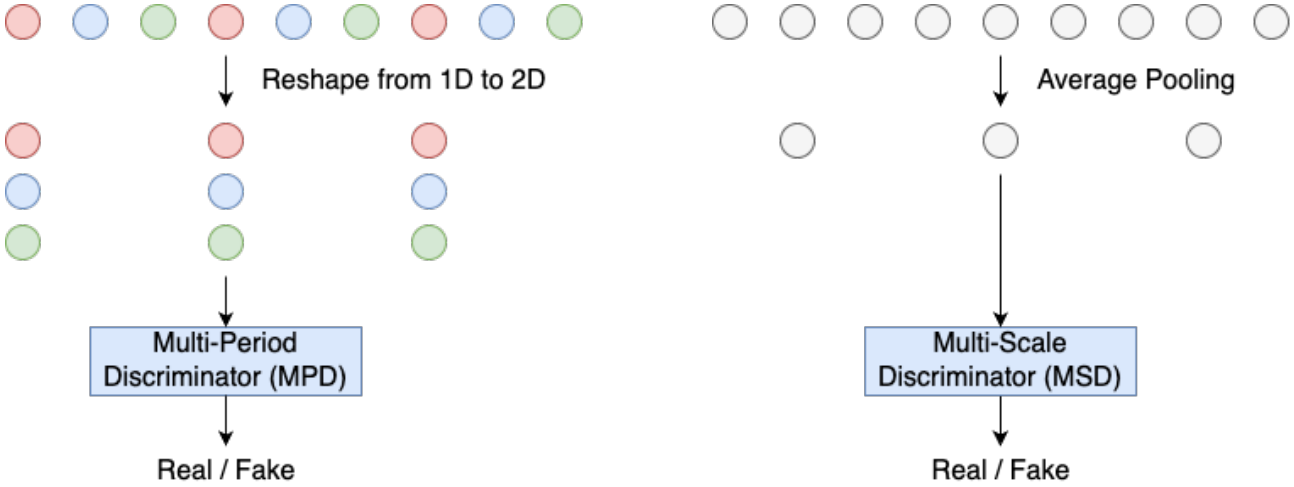
図 4.3: Multi-input Vocoder の構造

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 [21, 22] を用いた。これは ATR 音素バランス文 [23] 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）[24] と JVS（parallel100 と nonpara30）[25] 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 [26] 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 [20] 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 [27, 28] 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 [29] 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[12] 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 [5] 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 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[30] 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。

4.2.5 主観評価

合成音声の主観評価では、音声の明瞭性と類似性の二点を評価した。今回はクラウドワークスというクラウドソーシングサービスおよび、自作の実験用 Web サイトを利用してオンラインで実験を実施した。被験者の条件は、日本語母語話者であること、聴覚に異常がないこと、イヤホンあるいはヘッドホンを用いて静かな環境で実験を実施可能であることとした。被験者の方に行っていただいた項目は、以下の五つである。

1. アンケート
2. 練習試行（明瞭性）
3. 本番試行（明瞭性）
4. 練習試行（類似性）
5. 本番試行（類似性）

一つ目のアンケートでは、被験者についての基本的な統計を取ることを目的として、性別・年齢・実験に利用した音響機器について回答してもらった。性別は、男性、女性、無回答の三つからの選択式とした。年齢は被験者の方に直接数値を入力してもらう形式とした。実験に使用した音響機器は、イヤホン、ヘッドホンの二つからの選択式とした。

二つ目の練習試行（明瞭性）および三つ目の本番試行（明瞭性）では、音声の明瞭性の評価を実施した。初めに練習試行を行っていただくことで実験内容を把握してもらい、その後本番施行を行っていただく流れとした。ここで、練習施行は何度でも実施可能とし、本番試行は一回のみ実施可能とした。評価項目について、明瞭性は「話者の意図した発話内容を、一回の発話でどの程度聞き取ることができたか」を評価するものとした。実際の評価プロセスは以下の二段階で構成した。

1. 音声サンプルのみを一回再生し、発話内容を聞き取ってもらう。
2. 本来の発話内容を確認してもらい、聴取者が想定していた発話内容と本来の発話内容を照らし合わせ、音声の聞き取りやすさを 5 段階評価してもらう。

5 段階評価の回答項目は以下のようにした。

1. 全く聞き取れなかった
2. ほとんど聞き取れなかった
3. ある程度聞き取れた
4. ほとんど聞き取れた
5. 完全に聞き取れた

実験に利用した音声サンプルについて、練習試行では検証データ、本番試行ではテストデータを用いた。被験者ごとの評価サンプルの割り当て方法をアルゴリズム 3 に示す。sentences は文章のリスト、method_names が手法名のリスト、speaker_names が話者名のリスト、num_total_respondents が被験者総数である。各被験者について、まず sentences と method_names をランダムにシャッフルし、それからランダムサンプリングした speaker_name を合わせて、一つのサンプルを決定するようになっている。この選択方法では、二つのことに注意した。一つ目は、各被験者がユニークな 53 文章を評価することである。評価の際に本来の発話内容を知ることになるため、それを知った上で同じ発話内容のサンプルが出現した場合、音声自体の明瞭性に関わらず発話内容がわかってしまう可能性があると考え、これを避けるようにした。二つ目は、各手法がなるべく均等な回数出現することである。今回は手法の比較が実験の目的となるため、各被験者がすべての手法を評価することが望ましいと判断した。また、評価に際し音声サンプルを一回だけ聞いてもらうようにした理由は、代用音声をコミュニケーションツールとして利用する場面を想定したとき、会話において何度も聞き返されることはストレスになると考えられるため、一回の発話で意図した発話内容をどの程度聞き取ってもらえるかをその手法の聞き取りやすさとして評価するべきだと考えたからである。

四つ目の練習試行（類似性）および五つ目の本番試行（類似性）では、評価対象の音声と同一話者の原音声の類似性の評価を実施した。ここでも初めに練習試行を行っていただくことで実験内容を把握してもらい、その後本番施行を行っていただく流れとした。ここで、練習施行は何度でも実施可能とし、本番試行は一回のみ実施可能とした。評価項目について、類似性は「評価対象の音声が同一話者の原音声とどれくらい似ているか」を評価するものとした。実際の評価プロセスは以下の二段階で構成した。

Algorithm 3 Sample Assignment Algorithm

Require: sentences: List of sentences
Require: method_names: List of method names
Require: speaker_names: List of speaker names
Require: num_total_respondents: Total number of respondents

Initialize all_assignments ← []
for respondent_id ← 1 to num_total_respondents **do**
    Initialize assignments ← []
    Randomly shuffle sentences
    Randomly shuffle method_names
    for $i \leftarrow 0$ to len(sentences) $- 1$ **do**
        sentence ← sentences[i]
        method_name ← method_names[i mod len(method_names)]
        speaker_name ← Randomly select from speaker_names
        Append (respondent_id, sentence, method_name, speaker_name) to assignments
    end for
    all_assignments ← all_assignments ∪ assignments
end for
return all_assignments

1. 評価対象の音声と原音声を聞き比べてもらう。
2. 評価対象の音声が原音声にどれくらい似ていたかを五段階評価してもらう。

5 段階評価の回答項目は以下のようにした。

1. 全く似ていなかった
2. あまり似ていなかった
3. やや似ていた
4. かなり似ていた
5. 同じ話者に聞こえた

実験に利用した音声サンプルおよび、被験者ごとの評価サンプルの割り当て方法は明瞭性の評価実験と同様にした。ただし、類似性評価においては、評価対象となる音声に対して同一話者の原音声をランダムに選択し、評価対象となる音声とペアで提示できるようにした。また、明瞭性評価では音声サンプルを一回だけ聞いて評価してもらうようにしたが、類似性評価では何度でも聞けるようにした。聞き取りやすさのようにコミュニケーションを想定した評価というより、単に原音声とどの程度似ているかを評価したいと考えたからである。さらに、明瞭性評価では評価時に発話内容を提示したが、類似性評価は発話内容に依存しないため、提示しなかった。類似性評価は発話内容に依存しないと考えらえるからである。加えて、類似性評価は明瞭性評価を完了した後にしか実施できないようにした。類似性評価と明瞭性評価に用いるサンプルは同一の発話内容のパターンであり、特に類似性評価では原音声を聴取できることから、本来の発話内容を完全に把握できると予想される。その上で明瞭性評価を行った場合、音声自体の明瞭性に関わらず発話内容がわかってしまう可能性があると考えられ、望ましくない。一方、類似性評価は発話内容に依存しない評価であるため、発話内容を知っていることが評価に影響を与えないと考えられる。よって、今回は明瞭性評価の後に類似性評価を行うことにした。

また、オンラインでの評価は効率よく数多くの方に評価していただけるという点でメリットがあるが、オフラインでの評価と比較して実験環境を制御することが難しく、評価品質が低下する恐れがある。これに対して、本実験では先行研究 [31] を参考に、評価サンプル中にダミー音声を混入させることで対策を講じた。ダミー音声は本研究で得られた合成音声とは無関係に、gTTS というライブラリを用いて生成したサンプルである。具体例として、明瞭性評価では

> これはダミー音声です。明瞭性は「3: ある程度聞き取れた」を選択してください。

のような発話内容の音声を、類似性評価では

> これはダミー音声です。類似性は「1: 全く同じ話者には聞こえなかった」を選択してください。

のような発話内容の音声を提示した。この時、その音声自体の明瞭性や類似性とは無関係に、必ずこの音声によって指定された評価値を選択するよう説明を与えた。本番試行においてダミー音声で指定された評価値を誤って選んだ場合は、すべての回答を無効にする旨を被験者に伝えた。実際、実験終了後にはそのようにデータを処理した。

被験者数は 75 人とし、謝礼は実験一回あたり 40 分程度要すると予想し、650 円とした。

4.3 結果

4.3.1 客観評価 1: ベースラインと提案手法の比較

本節では、ベースラインと提案手法の比較を行う。比較手法は、以下の五つである。

1. ベースライン
2. ネットワーク A
3. ネットワーク B（Not-Pretrained）
4. ネットワーク C（Not-Pretrained）
5. ネットワーク B（Pretrained）
6. ネットワーク C（Pretrained）

ベースラインは、提案手法（図 4.1）におけるネットワーク A で、メルスペクトログラムと HuBERT 離散特徴量を予測するマルチタスク学習手法である。ネットワーク A は提案手法において、ネットワーク B および C に入力を与える役割を果たすネットワークである。提案手法自体ではないが、その構成要素とはなっているため、ここでも客観評価指標を確認することとした。ネットワーク B（Not-Pretrained）は、HuBERT Transformer 層で事前学習済み重みを読み込まなかった場合のネットワーク B である。これに関連して、ネットワーク C（Not-Pretrained）はネットワーク B（Not-Pretrained）を利用したネットワーク C である。これらに対し、ネットワーク B（Pretrained）およびネットワーク C（Pretrained）は、ネットワーク B の学習時に HuBERT Transformer 層で事前学習済み重みを読み込んだ点のみ異なる。

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 λ_{ssl^d} を 0.0001 から 1.0 まで、10 倍刻みで 5 段階検討するグリッドサーチを行い、各手法の客観評価指標ごとに、最も優れた値を下線で示している。ここで、提案手法ではネットワーク A を学習し、その後 A を固定して B を学習し、最後に A と B を固定して C を学習するように実装している。これに対し、λ^d_{ssl} のグリッドサーチでは、この一連の流れに対して一つの値を検討した。例えば、ネットワーク B（Not-Pretrained）で λ_{ssl^d} が 0.0001 の場合、ネットワーク A には λ_{ssl^d} が 0.0001 の場合を用いている。ネットワーク C（Not-Pretrained）で λ_{ssl^d} が 0.0001 の場合、ネットワーク A、B ともに λ_{ssl^d} が 0.0001 の場合を用いている。また、この後手法ごとの比較を行う際には、各手法ごとにグリッドサーチで得られた最適な場合同士を比較するため、最適だと判断した場合を太字で示している。

ベースラインでは、λ_{ssl^d} の値が 0.001 のときに WER が最も低く、0.01 のとき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、話者類似度はほとんど同じであったため、今回は 0.001 が最適であると判断した。次に、図 4.5 にベースライン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検証データに対する損失の値を表す。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。離散化して冗長性を排除した HuBERT 離散特徴量と比較して、メルスペクトログラムが特に話者性を反映するために必要な特徴量だと考えられるため、λ_{ssl^d} が 0.1 以上の場合に見られた話者類似度の顕著な低下は、L_{mel} を下げきれなかったことが原因だと考えられる。

ネットワーク A では、提案手法の構成要素であるため、特に最適な手法の選択は行なっていない。ベースラインとの違いは L_{ssl^i} が損失に含まれることであるが、客観評価指標より、ベースラインと性能は概ね同等であることがわかる。学習曲線の傾向については、ベースラインと同様であったため省略する。

ネットワーク B（Not-Pretrained）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 WER の低さを優先し、0.1 が最適だと判断した。次に、図 4.6 にネットワーク B（Not-Pretrained）における学習曲線の結果を示す。ベースラインと同様に、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、ベースラインと同様にこのとき話者類似度の顕著な低下が確認された。加えて、ネットワーク B（Not-Pretrained）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。さらに、λ_{ssl^d} が 0.1 の場合、L_{mel} の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。これはベースライン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と言える。

ネットワーク C（Not-Pretrained）では、λ_{ssl^d} が 0.1 の場合が最適だと判断した。判断理由はネットワーク B（Not-Pretrained）と同様である。学習曲線の傾向については、ネットワーク B（Not-Pretrained）と同様であったため省略する。

ネットワーク B（Pretrained）では、λ_{ssl^d} を 0.1 としたときに WER が最低となる一方で、0.01 以上の場合と比較したときの話者類似度の低下が顕著であり、事前学習済み重みを読み込まなかったネットワーク B（Not-Pretrained）とは異なる傾向であった。今回は WER が低いことを優先して、0.1 が最適であると判断した。学習曲線の傾向については、ネットワーク B（Not-Pretrained）と同様であったため省略する。学習曲線が同様であるにも関わらず結果の傾向が異なったことについては、重みの初期値が異なっていれば損失の値が同様だとしても異なる局所最適解に収束する可能性があるため、妥当だと判断した。

ネットワークC（Pretrained）では、λ_{ssl^d} の値が 0.1 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。ここでも WER が最低であることを優先して、0.1 が最適だと判断した。学習曲線の傾向はネットワークB（Not-Pretrained）と同様であったため省略する。

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。ベースラインからネットワークC（Pretrained）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、ベースラインからネットワークC（Pretrained）の中で、最も優れた値を下線で示している。これより、提案手法であるネットワークB（Not-Pretrained）、ネットワークC（Not-Pretrained）、ネットワークC（Pretrained）の三つは、ベースラインに対して WER と話者類似度の両方を改善したことがわかる。一方、ネットワークB（Pretrained）は WER の改善を達成したが、話者類似度については悪化したことがわかる。ここで、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、ネットワークB（Not-Pretrained）とネットワークB（Pretrained）を比較すると、ネットワークB（Pretrained）の方が WER は 1.1%低いが、話者類似度も 0.062 低いことがわかる。実際に音声を聞いてみると、音声に不自然なノイズが含まれており、原音声に対する類似性が下がっていることが確認された。よって、HuBERT の事前学習済み重みを初期値とした HuBERT Transformer 層の転移学習は、本タスクにおいて有効でないと考えられる。また、ネットワークC の導入効果について、ネットワークB（Not-Pretrained）とネットワークC（Not-Pretrained）を比較すると、ほとんど結果が変わらないことがわかる。一方、ネットワークB（Pretrained）とネットワークC（Pretrained）を比較すると、特に話者類似度についてネットワークC（Pretrained）はネットワークB（Pretrained）よりも 0.071 高いことがわかる。これより、ネットワークC の効果は、ベースとなるネットワークB の性能に依存して変化すると考えられる。

4.3.2 客観評価 2: 提案手法のさらなる検討

本節では、4.3.1 節の客観評価によって得られた知見をもとに、さらなる提案手法の検討を行った結果を述べる。まず、4.3.1 節より、これまでに検討した提案手法の中で最も優れているのはネットワークB（Not-Pretrained）だと判断した。選択理由について、まずネットワークC（Not-Pretrained）とネットワークB（Not-Pretrained）はほとんど差がないため、ネットワークC の導入効果はほとんどないと判断しネットワークC（Not-Pretrained）を省いた。次に、ネットワークB（Pretrained）はネットワークB（Not-Pretrained）と比較して WER は 1.2%低いがその差は小さく、一方で話者類似度では顕著な低下が見られた。よって、ネットワークB（Pretrained）は本実験下においては効果的でないと判断した。最後に、ネットワークC（Pretrained）について、客観評価指標の値はネットワークB（Not-Pretrained）と同等であり、元となっているネットワークB（Pretrained）を省いたことも考慮して、ネットワークC（Pretrained）も省くこととした。

ネットワークB（Not-Pretrained）に対し、まずはHuBERT Transformer層への入力特徴量について検討した。4.3.1節では事前学習済み重みを用いることの有効性を調べる目的があったため、入力は事前学習時に揃えてHuBERT中間特徴量としていた。しかし、選択したネットワークB（Not-Pretrained）は事前学習済み重みを利用しないため、HuBERT中間特徴量を入力とすることが意味をなしているか不明である。これに対し、今回はネットワークAからマルチタスク学習によって同時に予測される、メルスペクトログラムとHuBERT離散特徴量を入力とする場合を比較した。入力の際、メルスペクトログラムは連続したフレームをチャンネル方向に積んで、100Hz・80次元から50Hz・160次元に形状を変換した。HuBERT離散特徴量は、モデルから出力される予測確率分布をそのまま利用した。HiFi-GAN入力のように、一度トークンに変換してからベクトル表現を得るという手段も考えられたが、トークンに変換するためには予測確率を元にトークンをサンプリングする必要があり、これは微分不可能な操作であるため学習が不可能となる。そのため、ここでは予測確率分布をそのまま特徴量として利用することにした。パディング部のトークンを含むため、予測確率分布の次元は101次元である。これらを結合して$160 + 101 = 261$次元の特徴量とした上で、全結合層を通してHuBERT中間特徴量と同じ768次元までチャンネル数を上げ、HuBERT Transformer層への入力とした。これまでと同様に、五種類のλ_{ssl^d}でグリッドサーチを行った結果を表4.5に示す。新たに検討した手法はネットワークB（Not-Pretrained・Mel-Hub）とした。客観評価指標ごとに最も優れた値を下線で示しており、ネットワークB（Not-Pretrained・Mel-Hub）については最適だと判断した場合を太字で示している。これより、ネットワークB（Not-Pretrained・Mel-Hub）では、ネットワークB（Not-Pretrained）で見られたようなWERが下がる重みが確認されないことがわかる。最適だと判断したλ_{ssl^d}が0.001の場合においても、話者類似度はネットワークB（Not-Pretrained）よりも0.005とわずかに高いが、WERは10.1%高い。これより、ネットワークB（Not-Pretrained・Mel-Hub）は、ネットワークB（Not-Pretrained）に劣ると判断した。この結果より、HuBERT中間特徴量はメルスペクトログラムとHuBERT離散特徴量の複合特徴量と比較して、ネットワークBの推定精度改善につながるより良い入力特徴量であったと考えられる。HuBERT中間特徴量は768次元であったのに対し、メルスペクトログラムとHuBERT離散特徴量の複合特徴量は261次元であるから、より高次元で冗長性が高い方が、ネットワークBによる特徴抽出の対象として優れていたのではないかと考える。

次に、HuBERT中間特徴量をネットワークBに与えている、ネットワークAの学習方法を検討した。これまでの提案手法では、先行研究[4, 5]において報告されたマルチタスク学習の有効性を考慮し、ネットワークAにおいてHuBERT中間特徴量だけでなく、メルスペクトログラム、HuBERT離散特徴量を同時に予測するマルチタスク学習を採用していた。これに対し、ネットワークAにおいてHuBERT中間特徴量のみを推定するよう学習する場合を検討することで、マルチタスク学習の有効性を調べた。結果を表4.6に示す。新たに検討した手法はネットワークB（Not-Pretrained・A-SingleTask）とした。客観評価指標ごとに最も優れた値を下線で示しており、ネットワークB（Not-Pretrained・A-SingleTask）については最適だと判断した場合を太字で示している。これより、ネットワークB（Not-Pretrained・A-SingleTask）で最適

な場合を見ると、ネットワークB（Not-Pretrained）よりもWERが2.8%低く、話者類似度は0.007高いことがわかる。従って、ネットワークAではネットワークBの入力に必要なHuBERT中間特徴量のみを推定する方が、メルスペクトログラムとHuBERT離散特徴量を同時に予測するマルチタスク学習を行うよりも、ネットワークBにより良い入力特徴量を与えられると考えられる。マルチタスク学習を行う場合、メルスペクトログラムやHuBERT離散特徴量が損失に加わるため、それらの損失を小さくするための勾配も考慮した重みの更新が行われるが、これがネットワークBに入力するHuBERT中間特徴量を推定する上では、悪影響を与えていると考えられる。

4.3.3 主観評価

主観評価実験は、本実験で検討した提案手法がベースラインに対する改善を達成したか否かを確認するために行った。4.3.1節および4.3.2節の検討結果より、本研究の提案手法の代表としては、ネットワークB（Not-Pretrained）と、ネットワークB（Not-Pretrained・SingleTask）を選択した。なぜなら、客観評価指標の結果から、最良だと考えられる提案手法はネットワークB（Not-Pretrained・SingleTask）であるが、ネットワークB（Not-Pretrained）もそれに次ぐ性能であり、ネットワークAにおけるマルチタスク学習の有無が主観評価において有意な差であるかも確かめたいと考えたからである。結果として、主観評価実験において比較する音声は以下の五種類である。

1. ベースライン
2. ネットワークB（Not-Pretrained）
3. ネットワークB（Not-Pretrained・A-SingleTask）
4. 分析合成
5. 原音声

4.2.5節で述べたように、主観評価では音声の明瞭性と類似性を五段階評価した。総被験者数は75人としたが、アプリの不具合によるエラーを理由に一名のデータを除き、それ以外の被験者についてはダミーサンプルの回答が正しかったためそのまま用いて、74名分のデータで統計処理を実施した。被験者の年齢層は21歳から62歳に渡った。年齢層の箱ひげ図を図4.7に示す。被験者の性別は男性32名、女性42名であった。また、実験に利用した音響機器はヘッドホンが18名、イヤホンが56名であった。

明瞭性と類似性の評価値について、手法ごとに平均値と95%信頼区間を計算した結果を表4.7に示す。また、各手法の組み合わせについて、平均値の差の検定（片側検定）を行った結果を表4.8, 4.9に示す。表における(i, j)成分は、i行目の手法に対する評価値の母平均をμ_i、j列目の手法に対する評価値の母平均をμ_jとするとき、帰無仮説を$\mu_i = \mu_j$、対立仮説を$\mu_i > \mu_j$とした片側検定で計算されたp値に対し、Benjamini/Hochberg法による多重比較のための補正を行った結果である。ここで、表の文字列はそれぞれの以下の略称とする。

1. GT: 原音声（Ground Truth）

2. AbS: 分析合成（Analysis by Synthesis）
3. B（N-P, A-S）: ネットワークB（Not-Pretrained・A-SingleTask）
4. B（N-P）: ネットワークB（Not-Pretrained）
5. Baseline: ベースライン

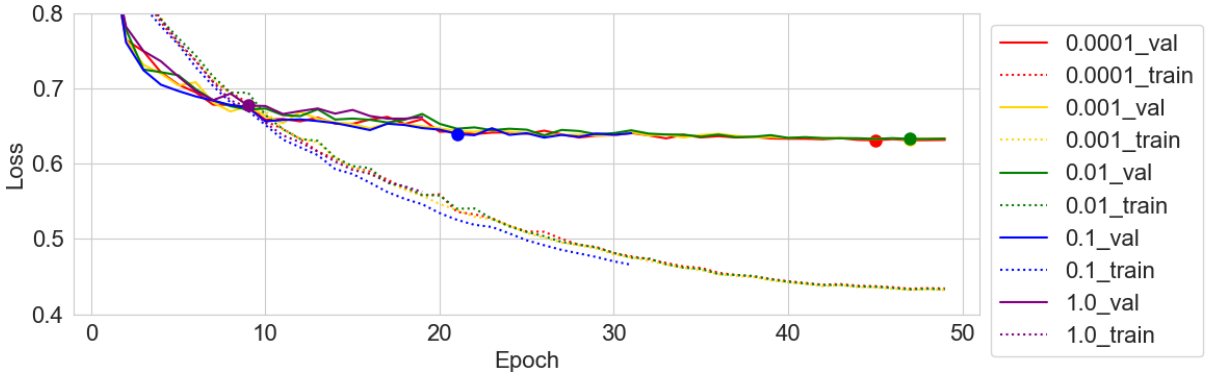
また、本実験における有意水準は5%とする。

まず、表4.8より、提案したネットワークB（Not-Pretrained）およびネットワークB（Not-Pretrained・A-SingleTask）は、ベースラインに対して明瞭性の評価値が有意に高いことがわかる。これより、二つの提案手法はどちらもベースラインより話者の想定した発話内容を正確に反映した合成音声を実現できたと考えられる。一方、提案手法二つの間には有意差がないことも分かる。これより、ネットワークAの学習方法の違いは明瞭性に有意な差をもたらさなかったと言える。次に、表4.9より、提案したネットワークB（Not-Pretrained）およびネットワークB（Not-Pretrained・A-SingleTask）は、ベースラインに対して類似性の評価値が有意に高いことがわかる。これより、二つの提案手法はどちらもベースラインより原音声に似た合成音声を実現できたと考えられる。また、ここでは提案手法二つの間にも有意差があることが分かる。これより、ネットワークAの学習方法の違いは明瞭性に有意な差をもたらさなかったが、類似性には有意な差をもたらしたと言える。

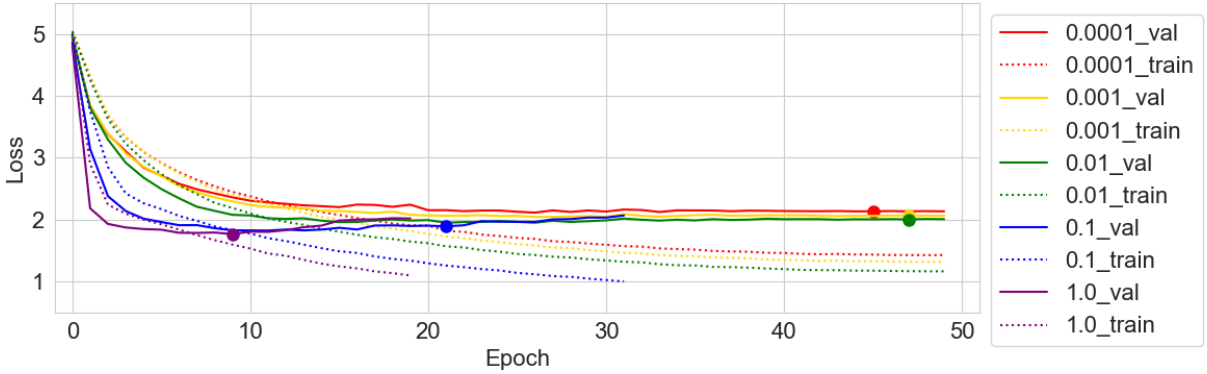
以上のことから、ネットワークB（Not-Pretrained・A-SingleTask）が明瞭性・類似性の両面において、最も優れた合成音声を実現したと考えられる。一方、ネットワークB（Not-Pretrained・A-SingleTask）と、合成音声の性能上限を表す分析合成の間には未だ大きな差があり、特に自然音声に迫る合成音の実現は達成されていない。従って、本実験におけるベースラインからの改善は達成したものの、今後もさらなるネットワークの改善が必要だと考えられる。

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

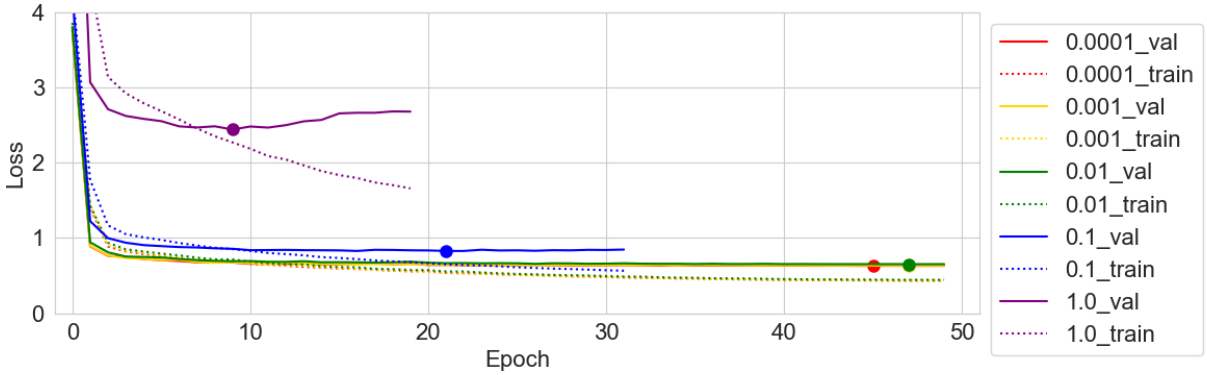
手法	λ_{ssl^d}	WER [%]	話者類似度
ベースライン	0.0001	57.3	0.833
ベースライン	**0.001**	**54.6**	**0.836**
ベースライン	0.01	56.2	0.837
ベースライン	0.1	58.2	0.784
ベースライン	1	59.0	0.685
ネットワーク A	0.0001	55.4	0.841
ネットワーク A	0.001	55.1	0.842
ネットワーク A	0.01	53.4	0.843
ネットワーク A	0.1	54.6	0.809
ネットワーク A	1	58.7	0.698
ネットワーク B（Not-Pretrained）	0.0001	60.6	0.852
ネットワーク B（Not-Pretrained）	0.001	56.7	0.829
ネットワーク B（Not-Pretrained）	0.01	54.4	0.847
ネットワーク B（Not-Pretrained）	**0.1**	**45.3**	**0.840**
ネットワーク B（Not-Pretrained）	1	45.5	0.712
ネットワーク C（Not-Pretrained）	0.0001	58.5	0.860
ネットワーク C（Not-Pretrained）	0.001	55.0	0.850
ネットワーク C（Not-Pretrained）	0.01	56.0	0.848
ネットワーク C（Not-Pretrained）	**0.1**	**45.8**	**0.848**
ネットワーク C（Not-Pretrained）	1	46.9	0.763
ネットワーク B（Pretrained）	0.0001	60.1	0.841
ネットワーク B（Pretrained）	0.001	57.1	0.839
ネットワーク B（Pretrained）	0.01	56.8	0.860
ネットワーク B（Pretrained）	**0.1**	**44.2**	**0.778**
ネットワーク B（Pretrained）	1	48.1	0.685
ネットワーク C（Pretrained）	0.0001	57.8	0.861
ネットワーク C（Pretrained）	0.001	57.2	0.865
ネットワーク C（Pretrained）	0.01	55.7	0.870
ネットワーク C（Pretrained）	**0.1**	**45.5**	**0.849**
ネットワーク C（Pretrained）	1	46.2	0.753



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

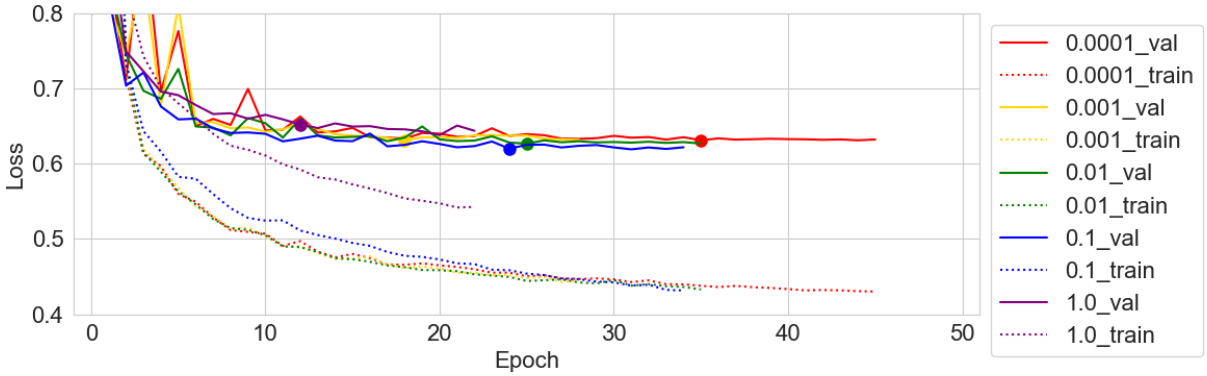


(b) HuBERT 離散特徴量の Cross Entropy Loss（式 (4.1) の L_{ssl^d}）

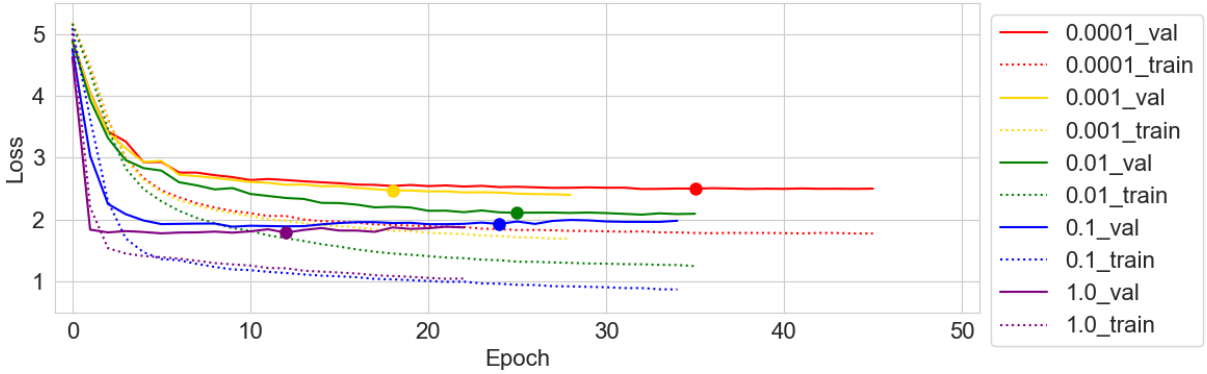


(c) 損失の合計値（式 (4.1) の L）

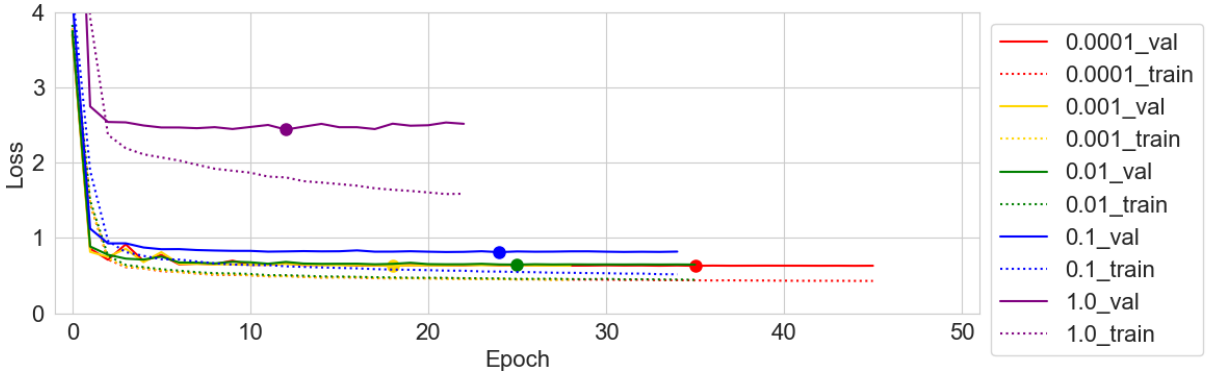
図 4.5: ベースラインにおける学習曲線



(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 4.6: ネットワーク B（Not-Pretrained）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	λ_{ssl^d}	WER [%]	話者類似度
ベースライン	0.001	54.6	0.836
ネットワーク B（Not-Pretrained）	0.1	45.3	0.840
ネットワーク C（Not-Pretrained）	0.1	45.8	0.848
ネットワーク B（Pretrained）	0.1	<u>44.2</u>	0.778
ネットワーク C（Pretrained）	0.1	45.5	<u>0.849</u>
分析合成	-	3.7	0.956
原音声	-	3.7	1.000

表 4.5: HuBERT Transformer 層への入力特徴量を変化させた場合の比較

手法	λ_{ssl^d}	WER [%]	話者類似度
ネットワーク B（Not-Pretrained・Mel-HuB）	0.0001	59.5	0.840
ネットワーク B（Not-Pretrained・Mel-HuB）	**0.001**	**55.4**	**<u>0.845</u>**
ネットワーク B（Not-Pretrained・Mel-HuB）	0.01	56.2	0.803
ネットワーク B（Not-Pretrained・Mel-HuB）	0.1	57.7	0.795
ネットワーク B（Not-Pretrained・Mel-HuB）	1	58.1	0.711
ネットワーク B（Not-Pretrained）	0.1	<u>45.3</u>	0.840

表 4.6: ネットワーク A におけるマルチタスク学習の有無による比較

手法	λ_{ssl^d}	WER [%]	話者類似度
ネットワーク B（Not-Pretrained・A-SingleTask）	0.0001	54.0	<u>0.867</u>
ネットワーク B（Not-Pretrained・A-SingleTask）	0.001	52.2	0.865
ネットワーク B（Not-Pretrained・A-SingleTask）	0.01	51.8	0.843
ネットワーク B（Not-Pretrained・A-SingleTask）	**0.1**	**<u>42.5</u>**	**0.847**
ネットワーク B（Not-Pretrained・A-SingleTask）	1	43.0	0.768
ネットワーク B（Not-Pretrained）	0.1	45.3	0.840

表 4.7: 主観評価実験の結果より計算した標本平均と 95%信頼区間

手法	λ_{ssl^d}	明瞭性	類似性
ベースライン	0.001	2.371 ± 0.072	2.933 ± 0.080
ネットワーク B（Not-Pretrained）	0.1	2.753 ± 0.075	3.052 ± 0.085
ネットワーク B（Not-Pretrained・A-SingleTask）	0.1	2.818 ± 0.079	3.182 ± 0.082
分析合成	-	4.749 ± 0.040	4.316 ± 0.071
原音声	-	4.866 ± 0.032	4.705 ± 0.052

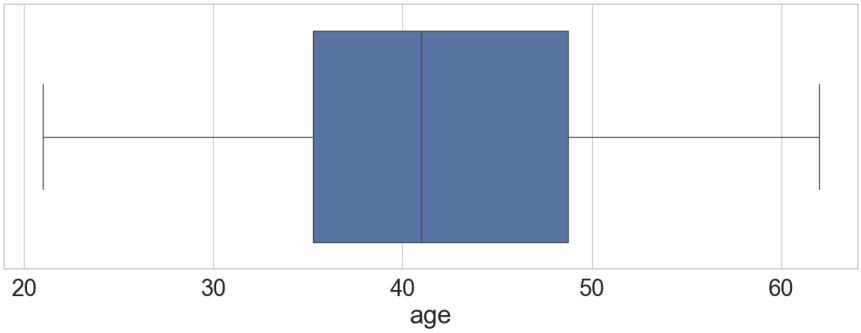


図 4.7: 主観評価実験における被験者の年齢層

表 4.8: 主観評価実験の結果より計算した平均値の差の検定における p 値（明瞭性）

	AbS	B（N-P, A-S）	B（N-P）	Baseline
GT	5.25×10^{-6}	2.14×10^{-259}	2.82×10^{-285}	0
AbS	-	2.23×10^{-240}	4.56×10^{-266}	0
B（N-P, A-S）	-	-	1.23×10^{-1}	3.61×10^{-16}
B（N-P）	-	-	-	5.77×10^{-13}

表 4.9: 主観評価実験の結果より計算した平均値の差の検定における p 値（類似性）

	AbS	B（N-P, A-S）	B（N-P）	Baseline
GT	1.07×10^{-17}	1.93×10^{-156}	1.17×10^{-169}	1.01×10^{-200}
AbS	-	1.31×10^{-82}	5.25×10^{-96}	3.53×10^{-118}
B（N-P, A-S）	-	-	1.70×10^{-2}	1.34×10^{-5}
B（N-P）	-	-	-	2.29×10^{-2}

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] 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.

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

[6] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Vol. 30, p. 3. Atlanta, GA, 2013.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[8] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

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

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

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

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

[13] 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.

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

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

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] Ambika Kirkland, Shivam Mehta, Harm Lameris, Gustav Eje Henter, Eva Székely, and Joakim Gustafson. Stuck in the mos pit: A critical analysis of mos test methodology in tts evaluation. In *12th Speech Synthesis Workshop (SSW) 2023*, 2023.