卒 業 論 文

機械学習を用いた音声の伝達特性からの声道断面積の自動抽出に関する研究

北海道科学大学 工学部
情報工学科
2-16-3-020
指導教員　松崎 博季
2020 年 (令和 2 年)2 月

目 次

第1章　緒言

音声発話機構の解明には発話器官と発話音声の関係を明らかにする事が欠かせない．そのための方法として，声道形状から音声を推定する手法と，音声から声道形状を逆推定する手法がある．前者の場合は，推定は容易だが声道形状を多量に取得することは困難である．それに対して後者の手法においては，音声データを多量に取得することは容易だが高い推定精度を出すことは困難であるという問題がある．しかしこの精度を向上させることができれば，多量のデータセット (音声と声道形状) が容易に得られるようになる．これまで逆推定には，初期値として任意の声道形状を，決められた拘束条件に従って変化させ対応するスペクトルを求め，これを所望のスペクトルが得られるまで繰り返す方法が提案されてきた [1,2]．これらは限られたデータのみを対象にしている，あるいは推定精度の面で問題が残っている．これらの手法とは異なる，近年深層学習で注目を浴びている機械学習を用いた手法 [3] が試みられるようになった．この手法では多量の音声データに対して声道形状を精度よく取得できることが期待されたが，従来手法よりも遥かに劣る推定精度しか得られていなかった．その原因として，ニューラルネットワーク (NN) の層が足りないことや，損失関数が問題に適していないなどの構造上の問題が考えられる．そこで本研究では文献 [3] の NN に対して上記問題の改善を施すことにより，推定精度向上を試みた結果について報告する．

第2章　声道形状の逆推定について

発声される音声の周波数特性 (共鳴特性) は発話器官が形作る形状，すなわち声道形状によって唯一に特徴付けられる．従って，声道形状を取得できれば，声道のどの部位が発声された音声を音響的に特徴づける主要因となっているのか，音声の個人性を特徴づけているのか，といったことを解析できるようになる．このため，音声研究において声道形状を取得することは必須とも言える事項であるため，様々な方法で声道形状を取得することが試みられている．主な取得法としては，コンピュータ断層撮影 (Computed Tomography, CT)，磁気共鳴映像法 (Magnetic Resonance Imaging，MRI) や，EMMA(Electromagnetic midsagittal articulater) がある．CT は鮮明に発話器官の形状を画像として取得できるが，X 線を使用するために人体が被爆するという問題がある．MRI は人体を被爆することはないが，時間分解能や空間分解能に起因する画像の不鮮明さ，さらに歯や骨など水分 (水素元素) をほとんど含まない部分が空気とほぼ同じ輝度値で取得されることによりこれらの部位を可視化できないなどの問題がある．EMMA はコイルを貼り付けた位置の時間変化を取得できるが，喉頭や咽頭などにコイルを貼り付けるのが困難で，これらの形状を取得するのが難しいという問題がある．いずれも機器の操作並びにデータ取得には専門のオペレーターが必要で，さらに CT や MRI での撮像費用は高額なため，多量の声道形状取得は容易ではない．上記のような特殊な機器を使わずに，発話された音声から声道形状を求めることができれば，声道形状の取得費用が安価になり多量のデータを得ることができるようになる．

これにより声道形状の微細構造が発話音声に与える影響を定量的にも明らかにすることができるようになり，音声生成機構の解明に有益である．音声から声道形状を求めることは，音声が生成される過程とは逆方向に推定することになるため逆推定と呼ばれ，これまで様々な手法が提案されてきた．例えば，Schroder や Mermelstein らは，声道断面積が一定の一様音響管における声道断面積関数の反対称な摂動を共鳴周波数の変動と対応付けるという，解析的なアプローチ [5,6] を提案した．また，白井らは，幾何学的な調音モデルによって声道形状を表し逆推定を行う手法 [7] を提案し，この研究では調音パラメータの時間変化が評価関数に取り入れられ，調音運動の軌道がより滑らかになるように求めることができるようになった．この他にも，前章で述べた調音位置などに生理学的な拘束条件を与える方法 [1] や，近年では機械学習を用いた手法も検討されている．これに関する概要は第 4 章で説明する．

第3章　機械学習について

機械学習とは，NN[1] を用いてコンピューターに学習をさせ，データの予測や分類などをすることである．本章では機械学習や NN に関する基本用語に関して概説する．

3.1　学習手法

機械学習の手法について以下に示す．

3.1.1　教師あり学習

機械学習には男性か女性かを判断するような分類問題と数値を予測する回帰問題があり，本研究は後者に属している．そしてこのような問題に取り組む際に用いられる学習手法は教師あり学習と呼ばれ，学習データに正解ラベルを付けて学習させる．

3.1.2　ミニバッチ学習

通常機械学習では多量のデータが用いられるため，それらすべてのデータに対して学習を行なっていると非常に時間を要してしまう．そのため多量のデータの中からランダムに選んだデータのみを対象に学習を行う手法がよく採用されており，この手法をミニバッチ学習という．

3.2　ニューラルネットワークで用いられる関数について

NN は主に入力層，中間層，出力層で構成され，各層で様々な計算が行われる．そのため本節では各層で行う計算に用いられる関数に関して説明する．

[1]NN とは人間の脳の神経回路網を人工的に数式モデルによって再現したものである

3.2.1　活性化関数

活性化関数とは，入力信号の総和を出力信号に変換する関数のことであり，主に入力層及び中間層で用いられる．また，NN では層を深くすることの利点を活かすために，活性化関数に非線形関数を用いなければならない [8]．文献 [3] では数ある活性化関数の中から，NN の活性関数としてよく用いられるシグモイド関数を使用している．シグモイド関数は式 (3.1) で表される．

$$h(x) = \frac{1}{1 + exp(-x)} \tag{3.1}$$

3.2.2　恒等関数

恒等関数とは NN の出力層で用いられる関数であり，前層から受け取ったデータをそのまま出力する関数である．機械学習は主に分類問題と回帰問題とに分けられるが，恒等関数は文献 [3] のように回帰問題に取り組む際に用いられる．

3.2.3　損失関数

損失関数とは，学習の性能の悪さを示す関数であり出力層から出力された値を評価する．そしてその損失関数の値が小さければ小さいほど，そのモデルは性能が良いということになる．そのため機械学習では，この損失関数の値ができるだけ小さくなるように，重みなどのパラメータを調節していくことが目的となる．代表的な損失関数として交差エントロピー誤差を式 3.2 に，2 乗和誤差を式 3.3 に示す．

$$E = -\sum_k t_k \log y_k \tag{3.2}$$

$$E = \frac{1}{2}\sum_k (y_k - t_k)^2 \tag{3.3}$$

3.3　機械学習において有用な手法

機械学習において精度向上や学習の高速化などに役立つ手法を以下に示す．

3.3.1　BatchNormalization

BatchNormaliztion は 2015 年に提案された新しい手法であるにも関わらず，機械学習において多くの優れた結果を出すことに成功している．BatchNormalization を導入することにより以下のような効果が期待できる [8].

- 学習を速く進行させることができる．
- 初期にそれほど依存しない．
- 過学習を抑制する．

過学習とは，訓練データのみに対応した学習を行なってしまいテストデータでの精度が低くなってしまうことである．過学習を起こすと，未知のデータに対しての対応力 (汎化性能) が損なわれてしまい，実用化に至れないといった問題が起きる．

3.3.2　パラメータ更新

パラメータの更新手法は数多く存在するが，その中でも新しい 2015 年に提案された Adam [9] という手法について説明する．Adam の特徴は他の更新手法である Momentum と AdaGrad の両方の利点を持つということである．まず Momentum の利点は物理法則を利用しているため，学習経路がボールが地面を転がるように滑らかに変化していき，効率的に最適化できる点である．次に AdaGrad の利点は学習係数の減衰という手法を用いていることで，最初は大きく学習し次第に小さく学習することで効率よく正しく学習が行える．以上のような利点を併せ持つ Adam は有効な手法であり，近年多くの研究者や技術者が用いている．

3.3.3　正規化

正規化とは，データを扱いやすくしたり特定の形式に当てはまるように，あるルールに基づいて変換することである．例えば決められた最小値と最大値に値が収まるように出力データの正規化を行う場合，与えられたデータを X，最大値を M，最小値を m とすると式 (3.4) で表される．

$$Y = \frac{X - X_{min}}{X_{max} - X_{min}}(M - m) + m \tag{3.4}$$

第4章　機械学習により声道形状を逆推定する先行研究 [3] について

文献 [3] は，機械学習を用いて声道伝達特性から声道断面積関数の推定を試みた研究である．本章ではこの研究の概要を示すとともに学習精度および声道断面積の出力結果を確認しその問題点について言及する．

4.1　概要

文献 [3] の概要について以下に示す．

4.1.1　学習に用いるデータ

入力データとして声道伝達特性[1] データを，その正解ラベルとして声道断面積関数[2] データを用いる．また，訓練データ (学習用のデータ) およびテストデータ (学習した結果を評価するデータ) は，それぞれ 10000 個作成する．その中から訓練データとして 7000 個を，テストデータとして 3000 個をランダムに選んで使用する．以下に声道断面積関数データおよび声道伝達特性データの作成方法を示す．

声道断面積関数データの作成方法

文献 [10] の図 69 などの声道断面積の模式図を参考に 7 つの調音位置[3] を定め，調音位置以外の声道断面積関数をスプライン関数 [11,12] を用いて，先に求めた調音位置が滑らかにつながるように作成されている．

---

[1] 音源フィルタ理論 [4] より声道伝達特性とは，喉頭から発せられた音源が声道フィルタにより調音され反映される共鳴特性のことである．

[2] 声道断面積関数とは声門から唇までの音波の伝送経路に対して垂直な面と声道の輪郭との交線により声道断面積を近似し，この交線を距離として表したものである．

[3] 声門，歯列，喉頭，咽頭，軽口蓋，口腔，口唇

声道伝達特性データの作成方法

電気的等価回路モデル [13] を用いることで声道断面積関数から声道伝達特性を求める．前述したような方法で求めた声道断面積関数の各セクション間を伝送線路による等価回路として扱った場合，全セクション間を継続接続音響管として扱うことができ，これにより声道断面積関数から声道伝達特性を求めることができる [4].

4.1.2　ニューラルネットワークの設定

学習に用いる NN には 3 層 (入力層・中間層・出力層)NN を採用している．これに関して NN の構造を図 4.1 に，NN の設定を表 4.1 に示す．入力層は Affine，BatchNormalization，Sigmoid という 3 つのレイヤにより構成され，過学習 (訓練データのみに対応した学習) を防ぎながら中間層にデータを渡す．中間層は，Affine レイヤのみで構成され，入力層から受け取ったデータを重みとよばれるデータの重要度を表す値により乗和計算を行う．出力層は，恒等関数により中間層から受け取ったデータをそのまま出力して，Cross_entropy レイヤにより学習の精度を評価する．また，ノード数については入力層，中間層，出力層の順にそれぞれ 1024 個，50 個，34 個である．

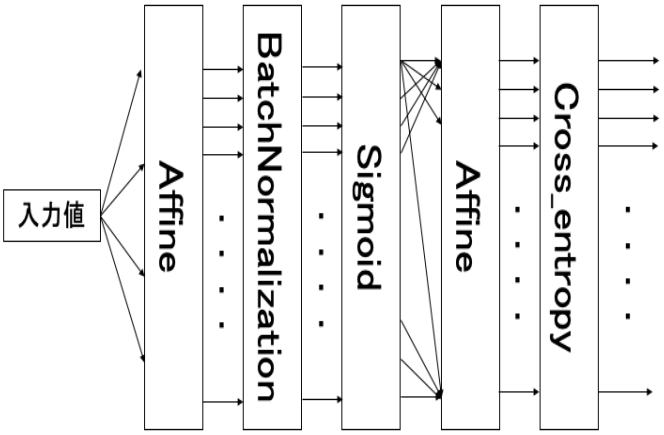

図 4.1: 学習に用いる NN の構造 [3]

表 4.1: NN の設定

データ数 (訓練/テスト)	学習回数	活性化関数	出力層	損失関数	BatchNormalization	パラメータ更新	正規化
7000/3000	10000	シグモイド	恒等関数	交差エントロピー誤差	あり	Adam	なし

4.1.3　声道断面積関数の出力方法

学習後に得た重みやバイアスなどのパラメータが設定された NN にテストデータの声道伝達特性を与え計算を行う．そして計算後の NN の出力が，推定された声道断面積関数となる．図 4.2 にテストデータの声道断面積関数と出力された声道断面積関数の比較を示す．これを見ると，テストデータの声道断面積関数の形をほとんど再現できていないことが分かる． また声道断面積には含まれないはずの負数が出力されており，正しい出力結果とは言えない．

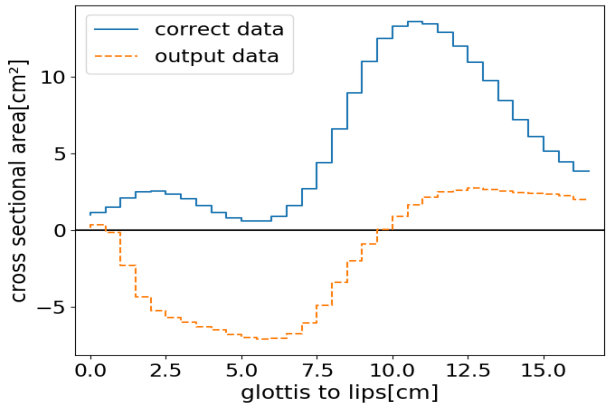

図 4.2: テストデータの声道断面積関数データと出力された声道断面積関数データ

4.1.4　出力結果の精度評価

出力より得た声道断面積関数と教師データとして使用した声道断面積関数の各セッションの誤差を測定する．求めた全 34 セクションの総誤差を平均した値が小さいほど高精度であるとする．なお図 4.2 における平均誤差は約 7.2cm^2 であり，文献 [3] で声道断面積の最大値を 12.67086cm^2 としていることを考えると，誤差が大きすぎることが分かる．次に文献 [3] と同じ条件で学習した結果を図 4.3 に示す．その精度は 20 %程度で十分な精度とは言えず，学習精度が向上していく様子も見られない．これらの要因を次節で考察する．

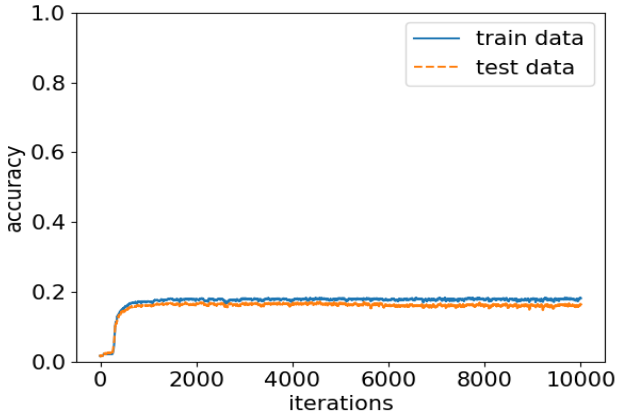

図 4.3: ニューラルネットワークの精度の推移

4.2　推定精度が低い要因について

主に NN の設定及び構造に着目し考察した．その結果以下のような要因により推定精度が低くなっていると考えられる．

4.2.1　ニューラルネットワークの層が不足している

NN において層の数やノード数は，表現力の向上に影響する．しかし，文献 [3] では NN を 3 層で構成しているため，入力層から中間層にかけてノード数が急激に減っている．これにより本問題を解決するだけの表現力に至らない，また特徴を上手く捉えられないまま結果が出力されているなどの問題が発生していると考えられる．

4.2.2　損失関数が問題に適していない

機械学習には男性か女性かを判断するような分類問題と数値を予測する回帰問題があり，どちらの問題に取り組むかによって採用する損失関数 (学習の精度を評価する関数) は変わってくる．今回の研究は回帰問題に当たるため，損失関数は 2 乗和誤差を採用するべき [8] であるが，文献 [3] では交差エントロピー誤差を用いている．これにより正しい結果が得られていないと考えられる．

4.2.3　出力データが正規化されていない

文献 [3] では，出力データをそのまま出力結果としており，その結果正しい出力が得られていなかった．そのため，出力データを正規化し適切な範囲に収めることが必要であると考えられる．

第5章　機械学習による逆推定手法の改善と改善後の学習結果

5.1　文献 [3] の推定手法の改善

前章で述べた文献 [3] の問題改善に伴う変更及び，それぞれの変更の有効性を検証した結果を示す．また，施した変更が有効であると判断された場合は，その変更を施したまま次の変更を施すこととする．なお，改善後の NN を構成するプログラムを付録 A.1 に，機械学習の実行をする際に使用するプログラムを付録 A.2 に，推定された声道断面積関数を表示するプログラムを表 A.3 に示す．

5.1.1　損失関数の 2 乗和誤差への変更

文献 [3] では損失関数に交差エントロピー誤差が用いられているが，本問題は回帰問題であるため，これを回帰問題に適した 2 乗和誤差 [8] に変更した．変更後の実行結果は図 5.1 および図 5.2 のようになった．実行結果より学習精度は 30%前後まで向上しており，さらに損失関数に関しても，徐々に値が 0 に近づいており正しい結果だと言えるためこの変更は有効だと言える．しかし，まだ十分な精度とは言えず改善が必要だと思われるため，さらなる変更を施すこととする．

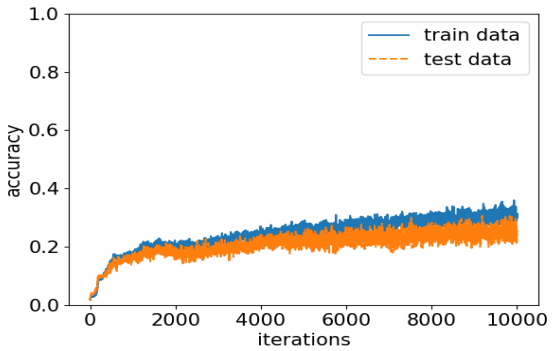

図 5.1: 学習精度の推移

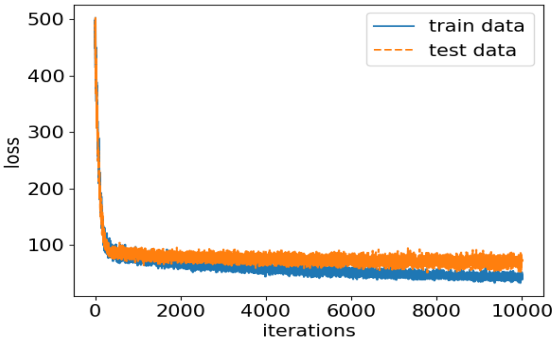

図 5.2: 学習の損失関数の推移

5.1.2　ニューラルネットワークの多層化

NN の表現力及び汎化性能向上のため，図 5.3 に示すような 7 層に多層化した．各層のノード数は入力層から順に，1024 個，800 個，512 個，256 個，128 個，64 個，34 個である．変更後の実行結果は図 5.4 および図 5.5 のようになった．学習精度は訓練データ (train data) が約 80 %まで，テストデータ (test data) が約 60%まで向上した．また損失関数に関しても先ほどより値の減少が見られるためこの変更は有効だったと言える．

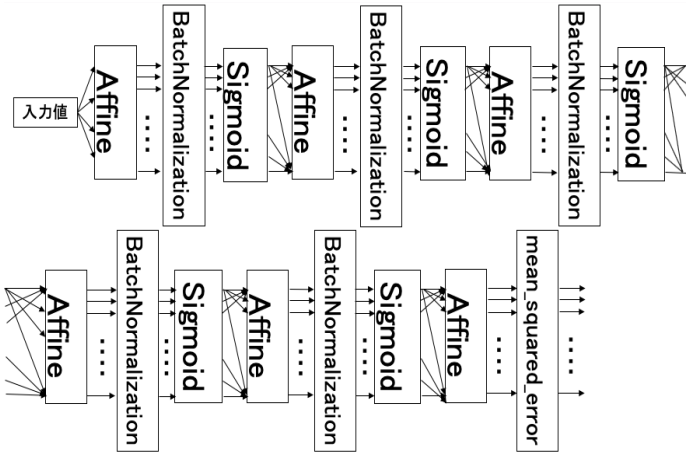

図 5.3: 学習に用いる NN の構造

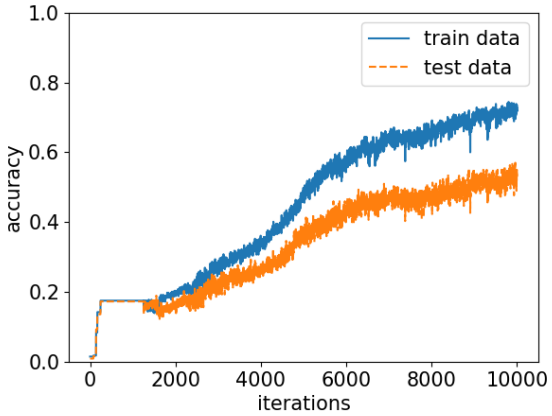

図 5.4: 学習精度の推移

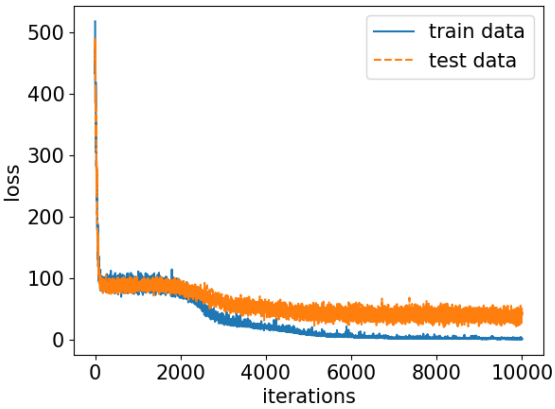

図 5.5: 学習の損失関数の推移

5.1.3　データの正規化

文献 [3] より，声道断面積関数の最小値は 0.45029cm^2，最大値は 12.67086cm^2 と定められているため，その範囲に収まるように，式 (3.4) を用いて出力データの正規化を行う．その結果，出力結果の確認に使用した全テストデータ 300 個の平均誤差は，改善前が約 5.15cm^2，改善後が約 3.86cm^2 となり，誤差が減少した．図 5.6 および図 5.7 は，改善後の誤差が最も小さい約 1.44cm^2 の場合と最も大きい約 6.59cm^2 の場合の正解，並びに改善前後の推定された声道断面積関数を表す．図 5.6 より改善後の結果 (output data(after)) が良くなっていることが分かる．しかし図 5.7 よりテストデータ (correct data) によらず出力される結果がほとんど同じになってしまうことが確認された．

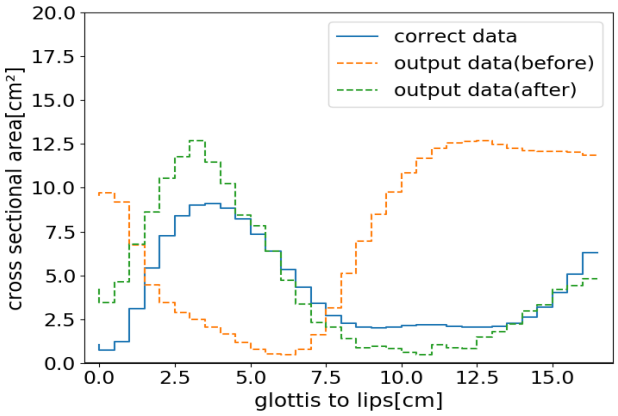

図 5.6: 出力された声道断面積関数の推定誤差が最も小さい場合

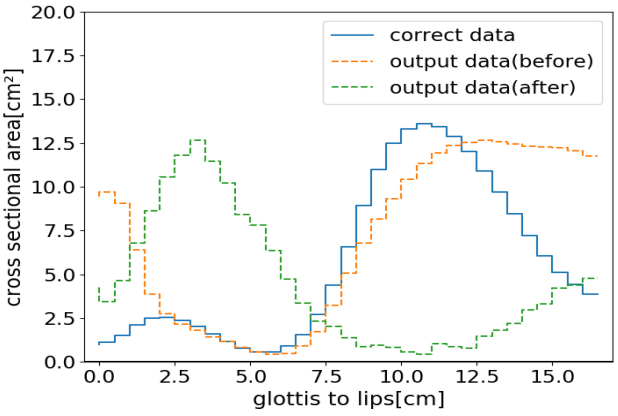

図 5.7: 出力された声道断面積関数の推定誤差が最も大きい場合

NN の変更は以上とし，改善後の最終的な NN の設定を表 5.1 に示す．

表 5.1: NN の設定

データ数 (訓練/テスト)	学習回数	活性化関数	出力層	損失関数	BatchNormalization	パラメータ更新	正規化
700/300	10000	シグモイド	恒等関数	2 乗和誤差	あり	Adam	あり

5.2　考察

上述の結果から考えられる問題点を考察し以下に示す．

5.2.1　学習精度について

5.1.1 項および 5.1.2 項の結果から学習精度は向上し，改善が見られた．しかし，図 5.4 より訓練データとテストデータの精度に 20%程度の差があり，過学習を起こしていると考えられる．そしてそれを防ぐための方法としてはデータ数の増加が挙げられる．つまり様々なパターンのデータを用意することでデータに多様性が生まれ，汎化性能の向上が期待できる．そこで，データ数を 10000 個に増やして実験したところ極端に学習精度が下がってしまった．この原因としては，使用している NN がデータ数 10000 個の多様性に対応できる構造になっていないか，あるいはデータを作成した際に予期しない値が発生していたなどの根本的な原因も考えられるため，これに関しての追加調査が必要である．

5.2.2　声道断面積の出力結果について

5.1.3 項の結果から，出力された声道断面積関数と正解の声道断面積関数との各セクションの誤差の平均値が小さくなっているため改善したと言える．しかし，図 5.6 および図 5.7 より，テストデータを変えても出力される結果にほとんど変化が見られず，汎化性能が無いことが分かる．これに関しては，NN 構造の見直しによる改善を図る．例えば，層やノード数が NN の表現力や汎化性能に関係しているため，それらの数の見直しなどが挙げられる．

第6章　おわりに

本研究では，先行研究 [3] をもとに機械学習による声道断面積の自動抽出の精度向上を計り，その結果改善することに成功した．しかし，まだ前章で述べたような問題点が存在したため，今後はまず NN の層数やノード数の増減による結果への影響を検証をすることで，より良い構造を導き出すことを試みる．それでもなお汎化性能に改善が見られない場合は，入力データと出力データを画像データとして捉え，画像における機械学習に長けた Conventional Neural Network(CNN) を用いるなどの別手法も検討する．

第7章　参考文献

[1] 西墻憲一, 堂建武, 本多清志, 発話機構モデルによる声道形状推定法を用いた音韻と発話様式の分析, 信学技報 SP2000-149, 2000

[2] 鏑木時彦, 感度関数を用いた音声スペクトルからの声道形状推定に関する検討, 電子情報通信信学会技術研究報告 SP2014-98, 2014

[3] 佐藤正隆, 機械械学習を用いた音声の伝達特性からの声道断面積の自動抽出, 北海道科学大学学位論文, 2019

[4] 鏑木時彦, 正木信夫, 元木邦俊, 松崎博季, 北村達也, 音声生成の計算モデルと可視化, コロナ社, 2010

[5] Mori, S., Zhang, J., Prciples of diffusion tensor imaging and its application to basic neuroscience research, Neuron, 2006

[6] Baser, J, P., Fiber-tractgraphy via diffusion tensor MRI(DT-MRI), Proc.6th Annual Meeting ISMRM, 1998

[7] Mady, K., Sader, R., Zimmermann, A., Hoole, P., Beer, A., Zeilhofer, H. and Hanning, C., Assesment of consonant articulation in glossectomee speech by dynmic MRI, Proc. ICSLP2002, 2002

[8] 斎藤秦毅, ゼロから作る Deep Learning - Python で学ぶディープラーニングの理論と実装, O'REILLY, 2016

[9] Kingma, Diederik and Ba., Jimmy, Adam: A Method for Stochastic Optimization, arXiv:1412.6980 [cs], 2014

[10] 千葉勉, 梶山正登, 母音ーその性質と構造ー, 岩波書店, 2003

[11] PolynomialSplineFunction(Commons Math 2.2 API), https://commons.apache.org/proper/commons-math/javadocs/api-2.2/org/apache/commons/math/analysis/polynomials/PolynomialSplineFunction.html

[12] SplineInterpolator(Commons Math 3.0 API), https://commons.apache.org/proper/commons-math/javadocs/api-3.0/org/apache/commons/math3/analysis/interpolation/SplineInterpolator.html

[13] 神山直久, 音声生成過程における音響特徴量抽出と実体的声道モデルに関する研究, 北海道大学博士論文, 1993.

付 録A　本研究で用いているプログラムの説明

本研究ではプログラミング言語 Python を用いてソースコードの作成を行なった．開発環境には emacs，実行環境には Mac のターミナルを使用した．また，現在では NN の構築を簡単に行うことができるライブラリ (TensorFlow，Chainer など) が複数存在するが，本研究では NN に関する知識・理解を深めるためにそういったライブラリを使わずに文献 [8] を参考に NN の構築を行なった．

A.1　7層NNを構成するプログラム

7 層 NN を構成するプログラムを表 A.1 に示す．

表 A.1: 7 層 NN を構成するプログラム一覧

ファイル名	付録	プログラムについての説明
SevenLayerNet.py	B.1	7 層 NN を構成するメインプログラム
layers.py	B.2	誤差逆伝播対応の各レイヤプログラム
function.py	B.3	各関数の計算を行うプログラム
gradient.py	B.4	微分計算を行うプログラム

A.2　学習の実行プログラム

機械学習を行う際に用いるプログラムを表 A.2 に示す．

表 A.2: 機械学習を構成するプログラム一覧

ファイル名	付録	プログラムについての説明
SevenLayerNet.py	B.1	7 層 NN を構成するプログラム
dataset.py	C.1	学習データ読み込みプログラム
optimizer.py	C.2	Adam を用いたパラメータ更新プログラム
akemi.py	C.3	機械学習実行プログラム

A.3　断面積関数のグラフ化する際に用いるプログラム

断面積関数をグラフ化する際に用いるプログラムを表 A.3 に示す.

表 A.3: 断面積関数のグラフ化に関するプログラム一覧

ファイル名	付録	プログラムについての説明
SevenLayerNet.py	B.1	7 層 NN を構成するメインプログラム
dataset.py	C.1	データを読み込むためのプログラム
neuralnet_area.py	D.1	断面積関数をグラフとして出力するプログラム

付 録B　NNのプログラム

B.1　7層NNプログラム (SevenLayerNet.py)

7層 NN を定義しているプログラムである.

```
"""
7層NN プログラム
"""
import sys, os
sys.path.append(os.pardir) # 親ディレクトリのファイルをインポートするための設定
import numpy as np
from common.layers import * #各レイヤで用いる誤差逆伝播対応のプログラム
from common.functions import * #活性化関数や恒等関数の計算を行うプログラム
from common.gradient import numerical_gradient #微分計算を行うプログラム
from collections import OrderedDict

class SevenLayerNet:

    def __init__(self, input_size, hidden_size, output_size, weight_init_std = 0.01):
        # 重み(W○ )及びバイアス( b○ )の初期化
        self.params = {}
        # 1 層目(入力層)
        self.params['W1'] = np.random.randn(input_size, 800) / np.sqrt(input_size)
        self.params['b1'] = np.zeros(800)
        # 2 層目(中間層)
        self.params['W2'] = np.random.randn(800, 512) / np.sqrt(800)
        self.params['b2'] = np.zeros(512)
        # 3 層目(中間層)
        self.params['W3'] = np.random.randn(512, 256) / np.sqrt(512)
        self.params['b3'] = np.zeros(256)
        # 4 層目(中間層)
        self.params['W4'] = np.random.randn(256, 128) / np.sqrt(256)
        self.params['b4'] = np.zeros(128)
        # 5 層目(中間層)
        self.params['W5'] = np.random.randn(128,64) / np.sqrt(128)
```

```
31          self.params['b5'] = np.zeros(64)
32          #６層目（中間層）
33          self.params['W6'] = np.random.randn(64, output_size) / np.sqrt(64)
34          self.params['b6'] = np.zeros(output_size)
35
36          #レイヤの生成
37          self.layers = OrderedDict()
38          #１層目（入力層）
39          self.layers['Affine'] = Affine(self.params['W1'], self.params['b1'])
40          self.layers['Sigmoid'] = Sigmoid()
41          self.layers['BatchNormalization'] = BatchNormalization(1, 0)
42          #２層目（中間層）
43          self.layers['Affine'] = Affine(self.params['W2'], self.params['b2'])
44          self.layers['Sigmoid'] = Sigmoid()
45          self.layers['BatchNormalization'] = BatchNormalization(1,0)
46          #３層目（中間層）
47          self.layers['Affine'] = Affine(self.params['W3'], self.params['b3'])
48          self.layers['Sigmoid'] = Sigmoid()
49          self.layers['BatchNormalization'] = BatchNormalization(1,0)
50          #４層目（中間層）
51          self.layers['Affine'] = Affine(self.params['W4'], self.params['b4'])
52          self.layers['Sigmoid'] = Sigmoid()
53          self.layers['BatchNormalization'] = BatchNormalization(1,0)
54          #５層目（中間層）
55          self.layers['Affine'] = Affine(self.params['W5'], self.params['b5'])
56          self.layers['Sigmoid'] = Sigmoid()
57          self.layers['BatchNormalization'] = BatchNormalization(1,0)
58          #６層目（中間層）
59          self.layers['Affine'] = Affine(self.params['W6'], self.params['b6'])
60          #７層目（出力層）
61          self.lastLayer = IdentityWithLoss()
62
63      #推論（x:入力データ）
64      def predict(self, x):
65          for layer in self.layers.values():
66              x = layer.forward(x)
67          return x
68
69      #損失関数（x:入力データ, t:教師データ）
70      def loss(self, x, t):
71          y = self.predict(x)
72          return self.lastLayer.forward(y, t)
73
74      #学習精度を求める（x:入力データ, t:教師データ）
```

```
75      def accuracy(self, x, t):
76          y = self.predict(x)
77          y = np.argmax(y, axis=1)
78          if t.ndim != 1 : t = np.argmax(t, axis=1)
79
80          accuracy = np.sum(y == t) / float(x.shape[0])
81          return accuracy
82
83      #伝播計算を行う(x:入力データ, t:教師データ)
84      def gradient(self, x, t):
85          # forward（順伝播）
86          self.loss(x, t)
87
88          # backward（逆伝播）
89          dout = 1
90          dout = self.lastLayer.backward(dout)
91
92          layers = list(self.layers.values())
93          layers.reverse()
94          for layer in layers:
95              dout = layer.backward(dout)
96
97          # 重み及びバイアスの設定
98          grads = {}
99          grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
100         grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
101         grads['W3'], grads['b3'] = self.layers['Affine3'].dW, self.layers['Affine3'].db
102         grads['W4'], grads['b4'] = self.layers['Affine4'].dW, self.layers['Affine4'].db
103         grads['W5'], grads['b5'] = self.layers['Affine5'].dW, self.layers['Affine5'].db
104         grads['W6'], grads['b6'] = self.layers['Affine6'].dW, self.layers['Affine6'].db
105
106         return grads
```

B.2　誤差逆伝播対応の各レイヤプログラム (laeyrs.py)

シグモイド関数の計算を行う Sigmoid レイヤ，前層から流れてきたデータに対して乗和計算を行う Affine レイヤ，恒等関数及び損失関数の計算を行う IdentityWithLoss レイヤ，BatchNormalization の計算を行う BatchNormalization レイヤの定義をしているプログラム.

```
1 """
2 各レイヤで用いるプログラム(誤差逆伝播対応)
3 """
```

```
import numpy as np
from common.functions import *
from common.util import im2col, col2im

#Sigmoid レイヤ
class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = sigmoid(x)
        self.out = out
        return out

    def backward(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx

#Affine レイヤ
class Affine:
    def __init__(self, W, b):
        self.W =W
        self.b = b

        self.x = None
        self.original_x_shape = None
        # 重み・バイアスパラメータの微分
        self.dW = None
        self.db = None

    def forward(self, x):
        # テンソル対応
        self.original_x_shape = x.shape
        x = x.reshape(x.shape[0], -1)
        self.x = x

        out = np.dot(self.x, self.W) + self.b

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
```

```
48              self.db = np.sum(dout, axis=0)
49
50              dx = dx.reshape(*self.original_x_shape) # 入力データの形状に戻す(テンソル対応)
51              return dx
52
53  #出力層(恒等関数及び2乗和誤差)
54  class IdentityWithLoss:
55      def __init__(self):
56              self.loss = None
57              self.y = None # softmax の出力
58              self.t = None # 教師データ
59
60      def forward(self, x, t):
61              self.t = t
62              self.y = y
63              batch_size = y.shape[0]
64
65              return 0.5 * np.sum((y-t)**2) / batch_size
66
67      def backward(self, dout=1):
68              batch_size = self.t.shape[0]
69              dx = (self.y - self.t) / batch_size
70
71              return dx
72
73  #BatchNormalization レイヤ
74  class BatchNormalization:
75      """
76      http://arxiv.org/abs/1502.03167
77      """
78      def __init__(self, gamma, beta, momentum=0.9, running_mean=None, running_var=None):
79              self.gamma = gamma
80              self.beta = beta
81              self.momentum = momentum
82              self.input_shape = None # Conv 層の場合は 4 次元、全結合層の場合は 2 次元
83
84              # テスト時に使用する平均と分散
85              self.running_mean = running_mean
86              self.running_var = running_var
87
88              # backward 時に使用する中間データ
89              self.batch_size = None
```

```
90          self.xc = None
91          self.std = None
92          self.dgamma = None
93          self.dbeta = None
94
95      def forward(self, x, train_flg=True):
96          self.input_shape = x.shape
97          if x.ndim != 2:
98              N, C, H, W = x.shape
99              x = x.reshape(N, -1)
100
101         out = self.__forward(x, train_flg)
102
103         return out.reshape(*self.input_shape)
104
105     def __forward(self, x, train_flg):
106         if self.running_mean is None:
107             N, D = x.shape
108             self.running_mean = np.zeros(D)
109             self.running_var = np.zeros(D)
110
111         if train_flg:
112             mu = x.mean(axis=0)
113             xc = x - mu
114             var = np.mean(xc**2, axis=0)
115             std = np.sqrt(var + 10e-7)
116             xn = xc / std
117
118             self.batch_size = x.shape[0]
119             self.xc = xc
120             self.xn = xn
121             self.std = std
122             self.running_mean = self.momentum * self.running_mean + (1-self.
                    momentum) * mu
123             self.running_var = self.momentum * self.running_var + (1-self.
                    momentum) * var
124         else:
125             xc = x - self.running_mean
126             xn = xc / ((np.sqrt(self.running_var + 10e-7)))
127
128         out = self.gamma * xn + self.beta
129         return out
130
131     def backward(self, dout):
```

```
132             if dout.ndim != 2:
133                 N, C, H, W = dout.shape
134                 dout = dout.reshape(N, -1)
135
136             dx = self.__backward(dout)
137
138             dx = dx.reshape(*self.input_shape)
139             return dx
140
141         def __backward(self, dout):
142             dbeta = dout.sum(axis=0)
143             dgamma = np.sum(self.xn * dout, axis=0)
144             dxn = self.gamma * dout
145             dxc = dxn / self.std
146             dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
147             dvar = 0.5 * dstd / self.std
148             dxc += (2.0 / self.batch_size) * self.xc * dvar
149             dmu = np.sum(dxc, axis=0)
150             dx = dxc - dmu / self.batch_size
151
152             self.dgamma = dgamma
153             self.dbeta = dbeta
154
155             return dx
```

B.3　各関数の計算を行うプログラム (function.py)

シグモイド関数，勾配計算，2 乗和誤差の計算を定義しているプログラム．

```
1  """
2  活性化関数及び損失関数の計算を行うプログラム
3  """
4  import numpy as np
5
6  #Sigmoid
7  def sigmoid(x):
8      return 1 / (1 + np.exp(-x))
9
10 #勾配計算
11 def sigmoid_grad(x):
12     return (1.0 - sigmoid(x)) * sigmoid(x)
13
14 # 2 乗和誤差
```

```
15 def mean_squared_error(y, t):
16     return 0.5 * np.sum((y−t)**2)
```

B.4　微分計算を行う (gradient.py)

微分計算を定義したプログラム.

```
1  """
2  微分計算を行うプログラム
3  """
4  import numpy as np
5
6  def numerical_gradient(f, x):
7      h = 1e−4 # 0.0001
8      grad = np.zeros_like(x)
9
10     it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
11     while not it.finished:
12         idx = it.multi_index
13         tmp_val = x[idx]
14         x[idx] = float(tmp_val) + h
15         fxh1 = f(x) # f(x+h)
16
17         x[idx] = tmp_val − h
18         fxh2 = f(x) # f(x−h)
19         grad[idx] = (fxh1 − fxh2) / (2*h)
20
21         x[idx] = tmp_val # 値を元に戻す
22         it.iternext()
23
24     return grad
```

付 録C　学習の実行プログラム

C.1　学習データ読み込みプログラム (dataset.py)

学習に用いるデータセットを読み込むためのプログラム.

```
"""
学習データ読み込みプログラム
"""
import gc
import time
from tqdm import tqdm
import glob
import os
import numpy as np
import random

def load_data():
    l_f = []
    file_num = 1000 #使用するファイル数

    #音声伝達特性ファイル読み込み
    peaks = []
    for filepath in tqdm(glob.glob('/Users/takuya/NetBeansProjects/Peaks/peaks/*.
        v')):
        with open(filepath, 'r') as f:
            l_f = list(map(float,f.readlines()))
            peaks.append(l_f)

    #訓練データ (x_train)、テストデータ (x_test)
    random.seed(1)
    x_train = np.array(random.sample(peaks, int(file_num*0.7)))
    x_test = np.array(random.sample(peaks, int(file_num*0.3)))

    #声道断面積関数ファイル読み込み
    area = []
    for filepath in tqdm(glob.glob('/Users/takuya/NetBeansProjects/Peaks/area
        /*')):
```

```
31          with open(filepath, 'r') as f:
32              l_f = list(map(float,f.readlines()))
33              del l_f[0:2] #声道長、セクション数を削除
34              area.append(l_f)
35
36      #訓練データ (t_train)、テストデータ (t_test)
37      random.seed(1)
38      t_train = np.array(random.sample(area, int(file_num * 0.7)))
39      t_test = np.array(random.sample(area, int(file_num * 0.3)))
40
41      return (x_train, t_train),(x_test, t_test)
```

C.2　Adam を用いたパラメータ更新プログラム (optimizer.py)

学習を行う際のパラメータ更新に用いる Adam を定義したプログラム.

```
1  """
2  Adam を用いたパラメータ更新プログラム
3  """
4  import numpy as np
5
6  class Adam:
7      #初期化
8      def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
9          self.lr = lr
10         self.beta1 = beta1
11         self.beta2 = beta2
12         self.iter = 0
13         self.m = None
14         self.v = None
15
16     #パラメータ更新
17     def update(self, params, grads):
18         if self.m is None:
19             self.m, self.v = {}, {}
20             for key, val in params.items():
21                 self.m[key] = np.zeros_like(val)
22                 self.v[key] = np.zeros_like(val)
23
24         self.iter += 1
25         lr_t = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 - self.beta1**self.
               iter)
26
```

```
27            for key in params.keys():
28                self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
29                self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])
30
31                params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)
```

C.3　機械学習実行プログラム (akemi.py)

表 A.2 に記載しているプログラムを用いて機械学習を実行するプログラム．学習後に正答率 (Accuracy) と損失関数の値 (Loss) を表示する．

```
1  """
2  機械学習実行プログラム
3  """
4  import sys, os
5  sys.path.append(os.pardir)
6  import matplotlib.pyplot as plt
7  import numpy as np
8  import pickle
9  from tqdm import tqdm
10 from dataset import load_data
11 from SevenLayerNet import SevenLayerNet
12 from common.optimizer import Adam
13
14 # データの読み込み
15 """
16 x_train.shape : (訓練データ数,1024)
17 t_train.shape : (訓練データ数,34)
18 x_test.shape : (テストデータ数,1024)
19 t_test.shape : (テストデータ数,34)
20 """
21 (x_train, t_train), (x_test, t_test) = load_data()
22
23 network = SevenLayerNet()
24 optimizer = Adam(lr=0.001) #学習率lr
25
26 iters_num = 10000 #学習回数
27 train_size = x_train.shape[0] #訓練データ数
28 test_size = x_test.shape[0] #テストデータ数
29 batch_size = 100 #ミニバッチ数
30
31 train_loss_list = []
```

```
test_loss_list = []
train_acc_list = []
test_acc_list = []

iter_per_epoch = max(train_size / batch_size, 1)

for i in tqdm(range(iters_num)):
    #ミニバッチを選ぶ
    batch_mask = np.random.choice(train_size, batch_size, replace=False)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    batch_mask_for_test = np.random.choice(test_size, batch_size, replace=False)
    x_t_batch = x_test[batch_mask_for_test]
    t_t_batch = t_test[batch_mask_for_test]

    # 微分を求めてパラメータの更新
    grad = network.gradient(x_batch, t_batch)
    optimizer.update(network.params, grad)

    #損失関数(loss:訓練データの損失、loss_test:テストデータの損失)
    loss = network.loss(x_batch, t_batch)
    loss_test = network.loss(x_t_batch, t_t_batch)

    train_loss_list.append(loss)
    test_loss_list.append(loss_test)

    #Accuracy（正答率）を保存
    train_acc = network.accuracy(x_train, t_train)
    test_acc = network.accuracy(x_test, t_test)
    train_acc_list.append(train_acc)
    test_acc_list.append(test_acc)

#最後のAccuracy と Loss を表示
print("train_loss_list : " + str(train_loss_list[-1]))
print("test_loss_list : " + str(test_loss_list[-1]))
print("train_acc_list : " + str(train_acc_list[-1]))
print("test_acc_list : " + str(test_acc_list[-1]))

#with open('/Users/takuya/CNN/grad_sample.pkl',mode='wb') as f: #改善前のパラメータを保存
#with open('/Users/takuya/CNN/grad_sample2.pkl',mode='wb') as f: #改善後のパラメータを保存
    pickle.dump(grad, f)
```

```
74
75
76 #グラフの表示
77 plt.rcParams["font.size"] = 15
78
79 plt.figure()
80 plt.xlabel('iterations')
81 plt.ylabel('loss')
82 plt.plot(train_loss_list, label = 'train data', linestyle = 'solid')
83 plt.plot(test_loss_list, label = 'test data', linestyle = 'dashed')
84 plt.legend()
85
86 plt.figure()
87 plt.xlabel('iterations')
88 plt.ylabel('accuracy')
89 plt.ylim(0.0,1.0,0.1)
90 plt.plot(train_acc_list, label = 'train data', linestyle = 'solid')
91 plt.plot(test_acc_list, label = 'test data', linestyle = 'dashed')
92 plt.legend()
93
94 plt.show()
```

付 録D　声道断面積関数をグラフ化するプログラム

D.1　声道断面積関数をグラフ化するプログラム (neuralnet_area.py)

表 A.3 に記載されているプログラムを用いて声道断面積関数をグラフ化する.

```
1  import sys, os
2  sys.path.append(os.pardir)
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import pickle
6  import random
7  from dataset2 import load_data
8  from common.functions import *
9  from common.layers import *
10
11 def get_data():
12       (x_test, t_test) = load_data()
13       return x_test, t_test
14
15 #改善前のパラメータを読み込む
16 def init_network_before():
17       with open('/Users/takuya/CNN/grad_sample.pkl', 'rb') as f:
18            network = pickle.load(f)
19       #print(network)
20       return network
21
22 #改善後のパラメータを読み込み
23 def init_network_after():
24       with open('/Users/takuya/CNN/grad_sample.pkl2', 'rb') as f:
25            network = pickle.load(f)
26       #print(network)
27       return network
28
29 #３層ニューラルネットワークの推論
30 def predict1(network, x):
31       W1 = network['W1']
```

```
32      W2 = network['W2']
33      b1 = network['b1']
34      b2 = network['b2']
35
36      a1 = np.dot(x, W1) + b1
37      z1 = sigmoid(a1)
38      a2 = np.dot(z1, W2) + b2
39
40      y = identity_function(a2)
41
42      return y
43
44  #7層ニューラルネットワークの推論
45  def predict2(network, x):
46      W1,W2,W3,W4,W5,W6 = network['W1'], network['W2'], network['W3'], network
            ['W4'], network['W5'], network['W6']
47      b1,b2,b3,b4,b5,b6 = network['b1'], network['b2'], network['b3'], network['b4'],
            network['b5'], network['b6']
48
49      """a1 = np.dot(x, W1) + b1
50      a2 = np.dot(a1, W2) + b2
51      z1 = sigmoid(a2)
52      a3 = np.dot(z1, W3) + b3
53      a4 = np.dot(a3, W4) + b4
54      z2 = sigmoid(a4)
55      a5 = np.dot(z2, W5) + b5
56      z3 = sigmoid(a5)
57      a6 = np.dot(z3, W6) + b6
58      y = identity_function(a6)"""
59
60      a1 = np.dot(x, W1) + b1
61      z1 = sigmoid(a1)
62      a2 = np.dot(z1, W2) + b2
63      z2 = sigmoid(a2)
64      a3 = np.dot(z2, W3) + b3
65      z3 = sigmoid(a3)
66      a4 = np.dot(z3, W4) + b4
67      z4 = sigmoid(a4)
68      a5 = np.dot(z4, W5) + b5
69      z5 = sigmoid(a5)
70      a6 = np.dot(z5, W6) + b6
71      y = identity_function(a6)
72      return y
73
```

```
74
75 network1 = init_network_before()
76 network2 = init_network_after()
77 error = 0
78 error_index = []
79 error_list = []
80 sum_error = 0
81 sum_error_list = []
82 output_before = []
83 output_after = []
84 sumple_num = 0
85
86 #データのダウンロード
87 x,t = get_data()
88
89 #推論 1
90 y = predict1(network1, x)
91 output_before.append(y)
92 print(output_before)
93
94 for i in range(len(y)): #要素数分のループ
95     if t[1][i] > y[i]:
96         error = t[1][i] - y[i]
97         error_index.append(error)
98     else:
99         error = y[i] - t[1][i]
100        error_index.append(error)
101    sum_error = sum(error_index)
102    sum_error_list.append(sum_error)
103
104 print('average:',sum(sum_error_list)/len(t))
105 print(output_before)
106
107 #推論 2
108 y = predict2(network2, x)
109 output_after.append(y)
110 print(output_after)
111
112 for i in range(len(y)): #要素数分のループ
113     if t[i] > y[i]:
114         error = t[1][i] - y[i]
115         error_index.append(error)
116     else:
117         error = y[i] - t[1][i]
```

```
118             error_index.append(error)
119         sum_error = sum(error_index)
120         sum_error_list.append(sum_error)
121
122 print('average:',sum(sum_error_list)/len(t))
123 print(output_after)
124
125 plt.figure()
126 left = np.arange(0, 17, 0.5)
127 plt.xlim([−0.5, 17])
128 plt.xlabel('glottis to lips[cm]', fontsize=18)
129 plt.ylabel('cross sectional area[cm^^c2^^b2]', fontsize=18)
130 plt.hlines(0, −0.5, 17)
131 plt.plot(left, t[sumple_num], label = 'load data', drawstyle = 'steps')
132 plt.plot(left, output_before[sumple_num], label = 'output data', drawstyle = 'steps',
        linestyle = 'dashed')
133 plt.plot(left, output_after[sumple_num], label = 'output data', drawstyle = 'steps',
        linestyle = 'dashed')
134 plt.legend()
135
136 left = np.arange(1, 4096, 4)
137 plt.figure()
138 plt.xlabel('frequency[Hz]', fontsize = 18)
139 plt.ylabel('amplitude spectrum[db]' ,fontsize = 18)
140 plt.plot(left, x[sumple_num])
141 plt.legend()
142
143 plt.show()
```

謝 辞

本研究は著者が北海道科学大学工学部　情報工学科在学中に，北海道科学大学工学部　情報工学科　音声処理分野，松﨑博季教授のもとで 2019 年度に行なったものである．

◇

本研究を進めるにあたり，終始，ご指導，ご鞭撻をいただいた北海道科学大学工学部　情報工学科　音声処理分野，松﨑博季教授に心より感謝いたします．また本研究の遂行にあたり様々な御助言，御協力を頂いただき支えてくださった北海道科学大学工学部　情報工学科　松﨑ゼミの皆さんに深く感謝いたします．最後に本研究の遂行にあたり著者を常に支援し応援してくれた両親に心より感謝申し上げます．