卒 業 論 文

# 機械学習を使った
# ギターエフェクターのパラメータ抽出

北海道科学大学 工学部
情報工学科
2-15-3-001
2-15-3-023
2-15-3-037
指導教員　松﨑 博季
2019 年 (平成 31 年)2 月

# 目 次

# 第1章　緒言

ギターを演奏する時、奏者はその曲の雰囲気に合わせた音色を選ぶ必要がある。例えば、激しい曲では歪ませた音を用い、静かな曲では透き通った音を用いるのが一般的である。このギターエフェクトのセッティングはギターの種類や演奏環境により無数に存在する。その為、理想とするギター音をセッティングするためにはアンプやエフェクトを幾度となく調整しなければならない。しかし、ギターの初心者にはどのような値に設定すれば良いかという判断が困難であると推測される。

そこで、この作業を自動化し所望する音を得るための方法として、自動化の手法の 1 つであるニューラルネットワーク (以下、NN と称す。) を活用する事ができないかと考えた。

本研究では数多くあるエフェクトの中から、演奏で頻繁に用いられる歪みエフェクターの歪み量の調整を対象とした。

プロギタリストのギター音をパラメータ単位で抽出し、その音の再現と音作りに役立てることを目的とし、歪みエフェクターの調査及び NN の実装を行った。

# 第2章　歪みエフェクターとアンプシミュレータ

## 2.1　歪みエフェクター

エフェクターとはギターの音色を変化させることが出来る機器であり、ダイナミクスの圧縮、イコライザー（特定の周波数領域を強調または減衰させる機能）の調整、残響の付加が出来るもの等様々な種類がある。

本実験で対象とする歪みエフェクターの起源は1920年代にまで遡る。当時、バンド体制での演奏の為大音量のサウンドを求めたギタリストは、ギターをアコスティック・ギターからエレキ・ギターに持ち替え、音声信号を増幅させるアンプという機器に繋ぐことでギターのボリュームを上げていた。しかしアンプには真空管を搭載しており、音の増幅域を超えるようなボリュームで音を出力しようとすると、音が飽和し潰れたような音が出力されてしまうことがあった。この音が歪みの始まりである。

当時のクリーンなギター音とはかけ離れた音であったが、この音はブルースやロックという音楽に活用されていった。その後、音量を過度に大きくしなくとも機器を通して歪みの付与、歪み量の調整ができるようになった。これが歪みエフェクターである [1]。

歪みエフェクターには大別して、ファズ、ディストーション、オーバードライブと呼ばれるものがある。歪みの強さの度合としては、ファズ＞ディストーション＞オーバードライブのように感じられるのが一般的である。これらは回路の違いはあるが、共通して波形の上下を切り取るクリッピングという処理を行っている。クリッピングを行うことで倍音成分が増え、歪んだ音が生まれる [2]。

クリッピングには上下対称に切り取る対称クリッピングと、上下非対称に切り取る非対称クリッピングがある [3]。クリッピングを行うことにより音量が均一化されることも歪みエフェクターの特徴である。また、歪みの量を調節するパラメータとしてGainという名称が多く使われる。

## 2.2　アンプシミュレータ

エレキギターを演奏する際には先ほど述べたアンプと、増大させた音を出力するスピーカーを用いる必要がある。ギター用アンプにはスピーカーの他、歪みエフェクター、イコライザーなどのエフェクターが付属しているものが多い。

アンプシミュレータとは実在するアンプの音色や使用感、エフェクターなどの機能をコンピュータ上で再現したもので、アンプの音を捉えるマイクの種類やマイクの位置を変更することもできる。

# 第3章　研究に用いたデータ、編集ソフト及び機材

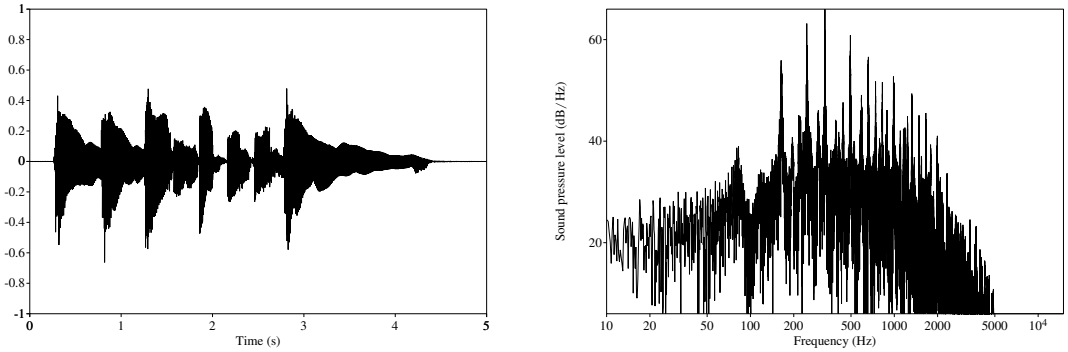
学習・調査に用いた編集ソフト及び機材を以下に示す。

- GarageBand - DAW(音声編集ソフト)
- AmpliTube4[British Tube Lead 1] -アンプシミュレータ
- Automator - mac 付属ソフト
- Pycharm - python 統合開発環境
- USB Audio Interface [UA-55, QUAD-CAPTURE]
- ギター 1 [YAMAHA PACIFICA112V]
- ギター 2 [Ginson レスポール]
- Strymon Sunset - ギター用歪みエフェクター (特性調査用)

エレキギター音源は、エレキギター中級者がギター 1 、2 を 5 秒ほど演奏した物を収録した。なお、本研究に用いた音源は、サンプリング周波数 44,100Hz、量子化ビット 16bit に統一した。

学習用データとしては、ギター 1 でエフェクターをかける元となるフレーズを 4 種類収録し、自動的に Amplitube4 の Gain の設定値を 0.0〜9.9 の 100 段階からランダムに設定して約 1400 個のデータを作成した。

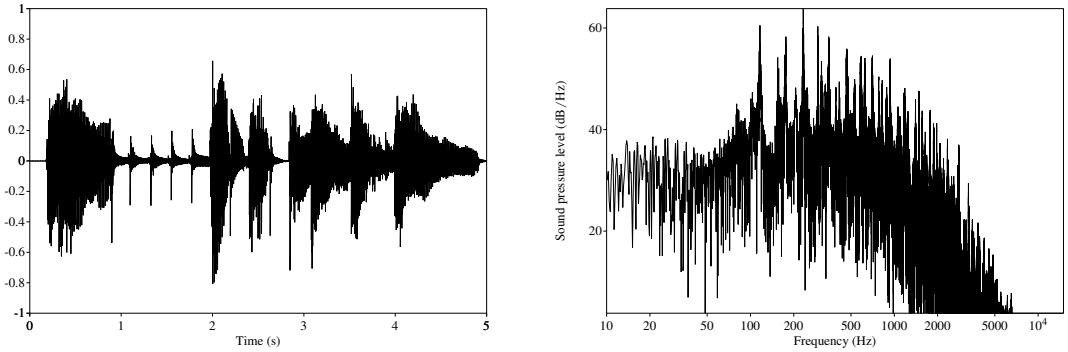
一つ目のフレーズは単音と和音が混じったフレーズを収録した。その波形と振幅スペクトラムを図 3.1 に示す。




波形　　振幅スペクトラム

図 3.1: 一つ目のフレーズにおける波形と振幅スペクトラム

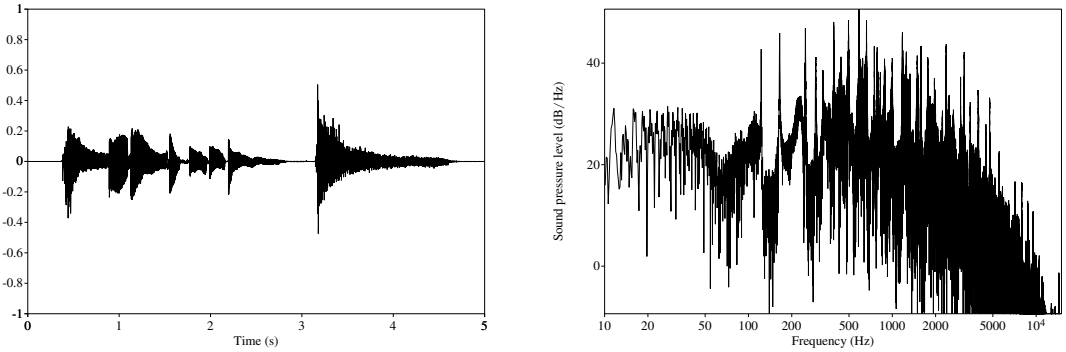
二つ目のフレーズは和音とブリッジミュートが混じったフレーズを収録した。その波形と振幅スペクトラムを図 3.2 に示す。




波形　　振幅スペクトラム

図 3.2: 二つ目のフレーズにおける波形と振幅スペクトラム

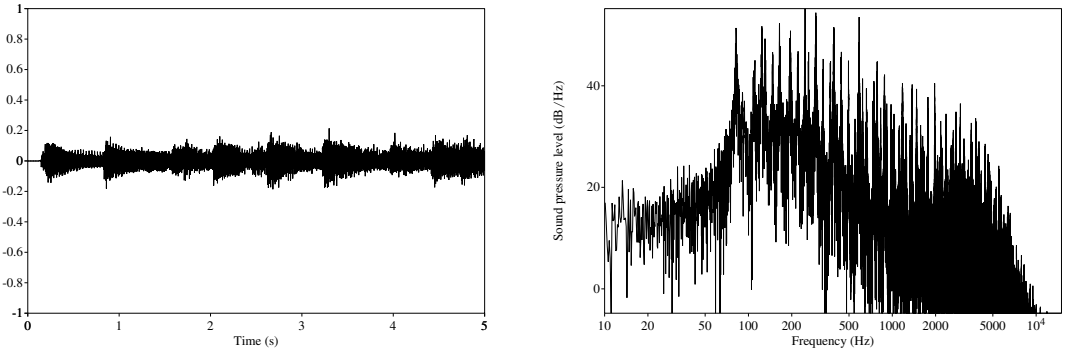
三つ目のフレーズはほとんど単音のみのフレーズを収録した。その波形と振幅スペクトラムを図 3.3 に示す。




波形　　振幅スペクトラム

図 3.3: 三つ目のフレーズにおける波形と振幅スペクトラム

四つ目のフレーズは和音のみのフレーズを収録した。その波形と振幅スペクトラムを図 3.4 に示す。




波形　　振幅スペクトラム

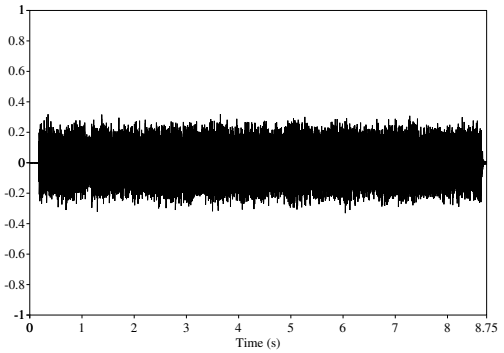
図 3.4: 四つ目のフレーズにおける波形と振幅スペクトラム

Amplitube4 を用いた理由はフリーで使用することができるアンプシミュレータの中で特に品質が良く、Gain や Volume などの調節を数値で行うことができるからである。自動化にあたっては、mac 上でマウスやキーボードの操作を自動化する Automator を用いた。操作手順については付録 A.1 を参照されたい。

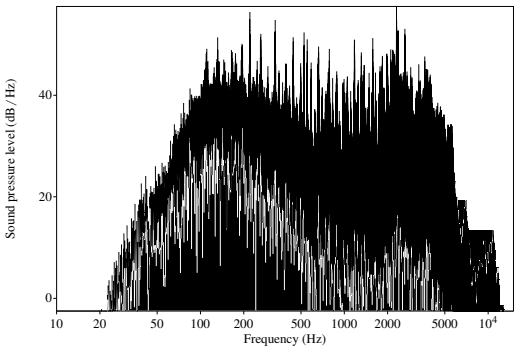
テスト用のデータとしては、ギター 1 とギター 2 で同じ Gain 値で同じフレーズを演奏したものを 5 つずつ用いた。表 3.1 にテスト用のデータ一覧、図 3.5〜図 3.14 にテスト用データの波形と振幅スペクトラムを示す。なお、接頭辞に Y がついた物はギター 1 、G がついたものはギター 2 を使用しており、各ギターで同じフレーズを録音した。

表 3.1: 機械学習の汎化性能を調査するために作成したテスト用のギターフレーズの音声データ

| 識別 | Gain の値 | 特徴 |
|---|---|---|
| Y01, G01 | 5.0 | 和音のみのフレーズ |
| Y02, G02 | 2.0 | 和音のみのフレーズ |
| Y03, G03 | 8.0 | 単音のみのフレーズ |
| Y04, G04 | 1.0 | アルペジオ奏法を用いたフレーズ |
| Y05, G05 | 9.9 | 和音と単音が混ざったフレーズ |

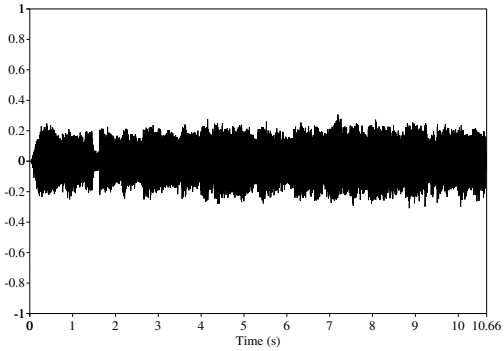



Y01 の波形

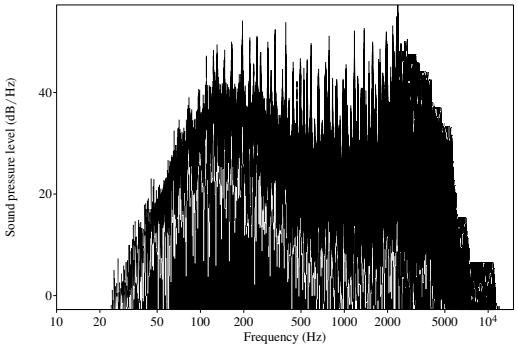



Y01 の振幅スペクトラム

図 3.5: Y01 の波形と振幅スペクトラム

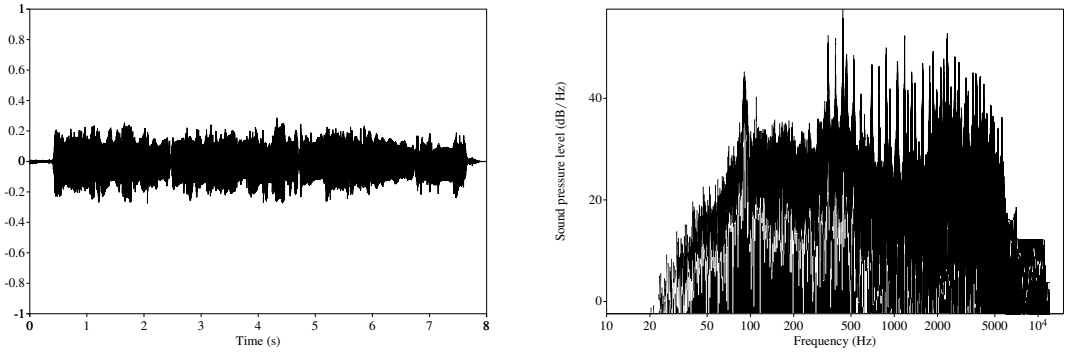



Y02 の波形



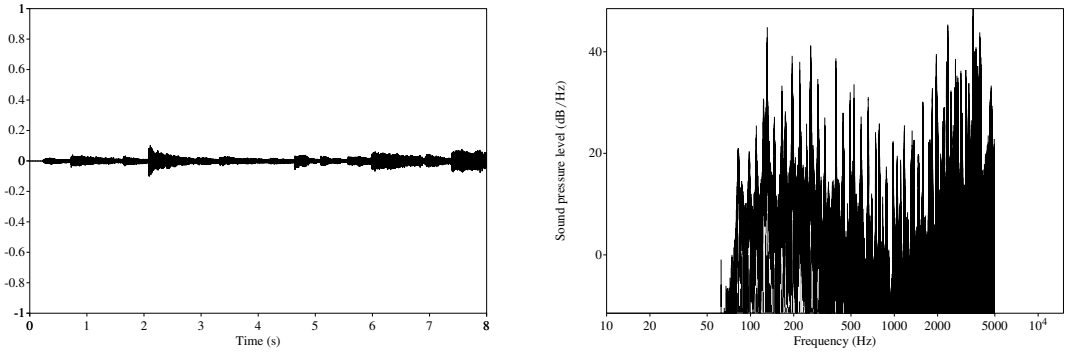

Y02 の振幅スペクトラム

図 3.6: Y02 の波形と振幅スペクトラム



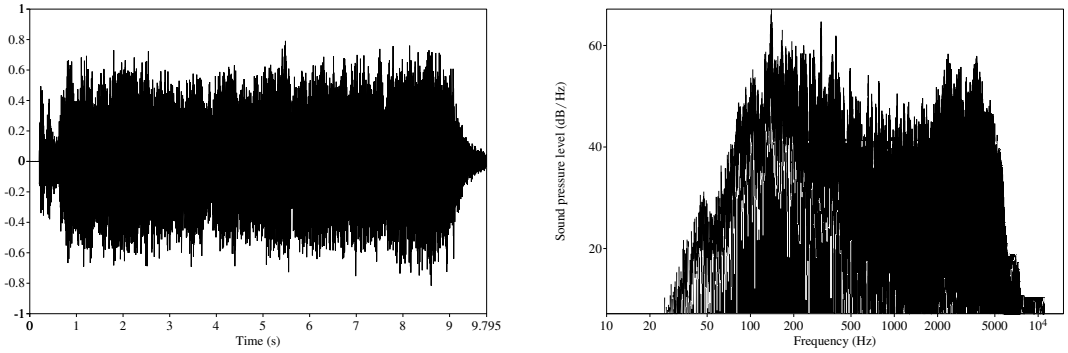

Y03 の波形　　Y03 の振幅スペクトラム

図 3.7: Y03 の波形と振幅スペクトラム



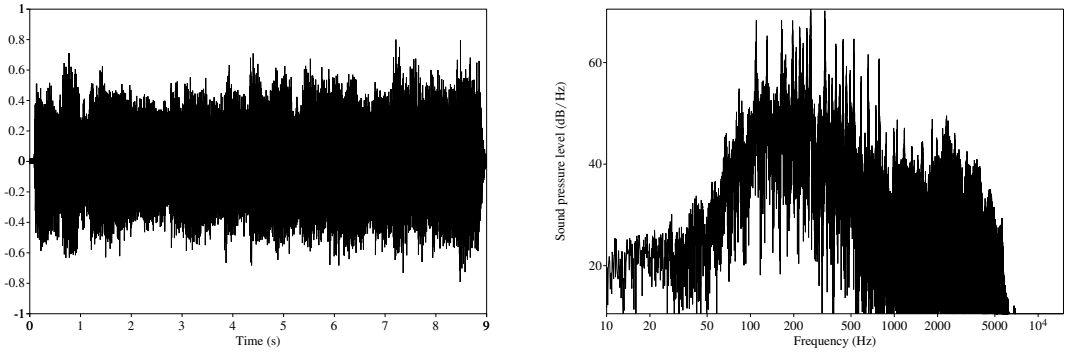

Y04 の波形　　Y04 の振幅スペクトラム

図 3.8: Y04 の波形と振幅スペクトラム



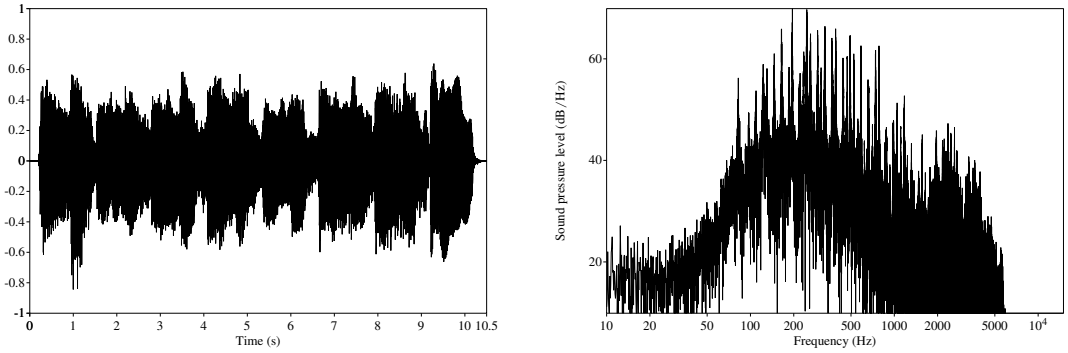

Y05 の波形　　Y05 の振幅スペクトラム

図 3.9: Y05 の波形と振幅スペクトラム



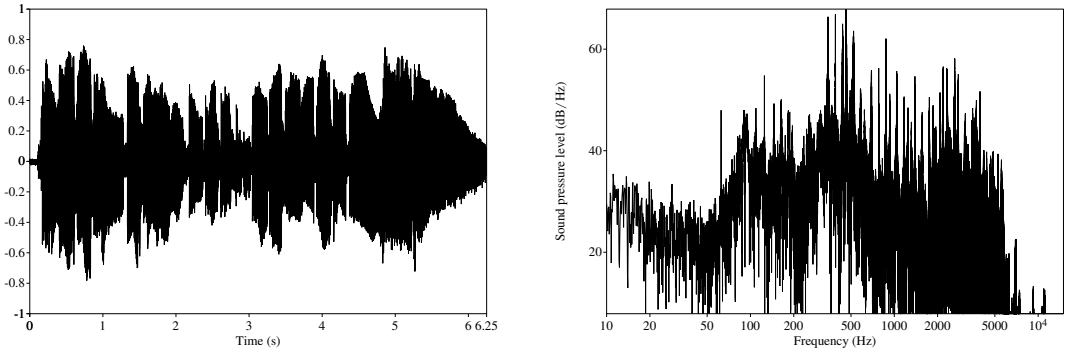

G01 の波形　　G01 の振幅スペクトラム

図 3.10: G01 の波形と振幅スペクトラム



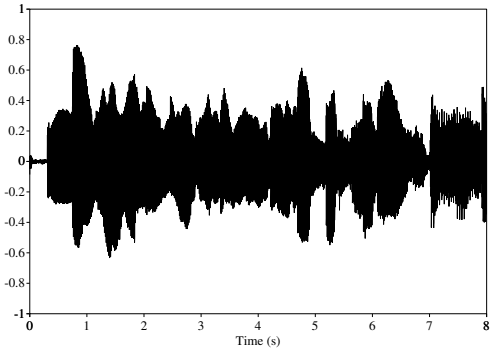

G02 の波形　　G02 の振幅スペクトラム

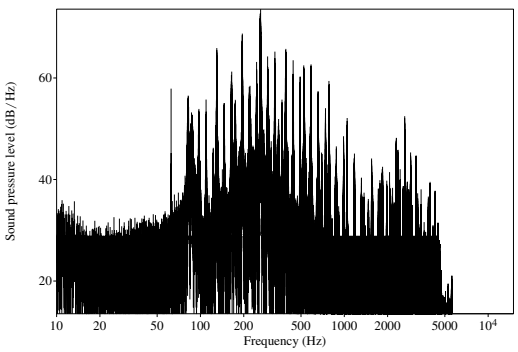
図 3.11: G02 の波形と振幅スペクトラム




G03 の波形　　G03 の振幅スペクトラム

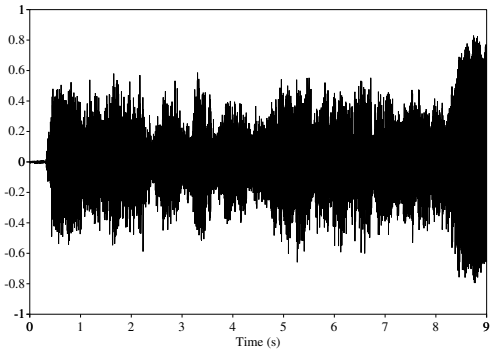
図 3.12: G03 の波形と振幅スペクトラム

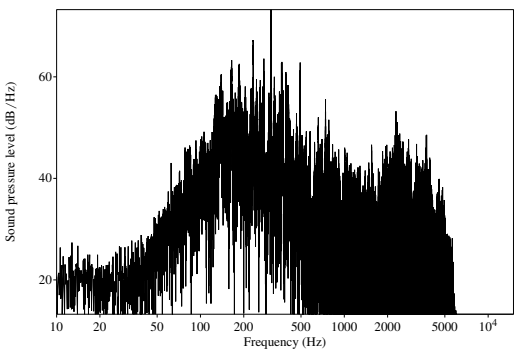



G04 の波形　　G04 の振幅スペクトラム

図 3.13: G04 の波形と振幅スペクトラム




G05 の波形　　G05 の振幅スペクトラム

図 3.14: G05 の波形と振幅スペクトラム

# 第4章　機械学習を用いた実験

歪みエフェクターの歪み量を推測する為、機械学習を用いた実験を行なった。

実験1として振幅スペクトラムをそのまま入力し機械学習を行った。振幅スペクトラムを求める方法として、高速フーリエ変換 (以下 、FFT と称する。) を用いた。

次に、歪みにおける倍音の特徴や過学習の防止に注目し、実験2として倍音成分の比率の平均値のようなものを入力し機械学習を行った。

最後に実験3として、振幅スペクトラムではなく、クリッピングの作用に注目した実験を行った。実験3では-1〜1 の間に正規化した波形の振幅の平均値を入力し、回帰直線を求めた。

なお、検証用データとして学習用音声データの全てからランダムに150のデータを取り出し、学習中の誤差の計算に用いた。

## 4.1　実験1:振幅スペクトラムに着目した実験

図4.1に実験1に用いた機械学習のモデル図を示す。

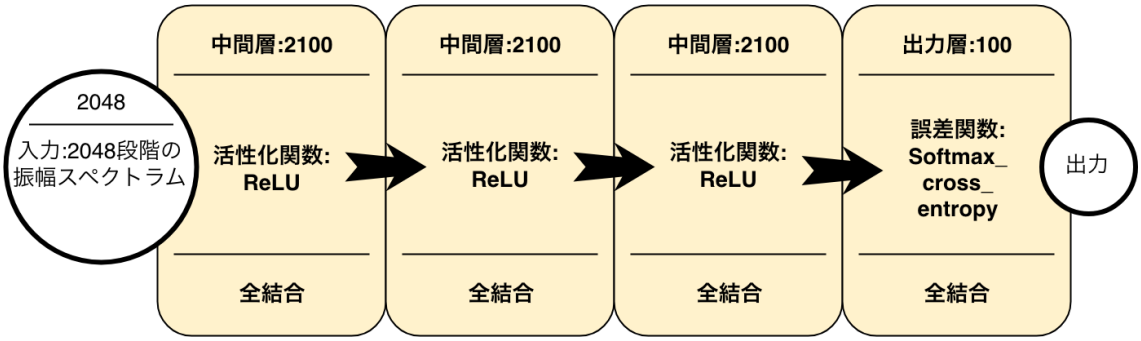



図 4.1: 実験1に用いた機械学習のモデル

この実験では単純な4層全結合のNNを用いて、振幅スペクトラムをそのまま入力し機械学習を行う実験を行った。ソースコードについては付録B.1を参照されたい。

アンプシミュレータのGainの設定値を100段階設定しているため、出力層を100個とし、データを一つずつ入力し学習させるオンライン学習を用いて学習させた。

FFT の設定はサンプル数 4096 とし、サンプルを 75 %ずつずらして FFT を行い 10 個の振幅スペクトラムの平均を算出した。4096 サンプルで FFT を行ったため約 10.7[Hz] の幅で、22050[Hz] までの 2048 段階の振幅スペクトラムを計算することができる。

## 4.2　実験 1 の結果と考察

学習用データを一通り学習させた後に検証用データセットを入力し誤差を計算した結果、図 4.2 のように約 7 周の学習で誤差が 1.0 以下に収束した。

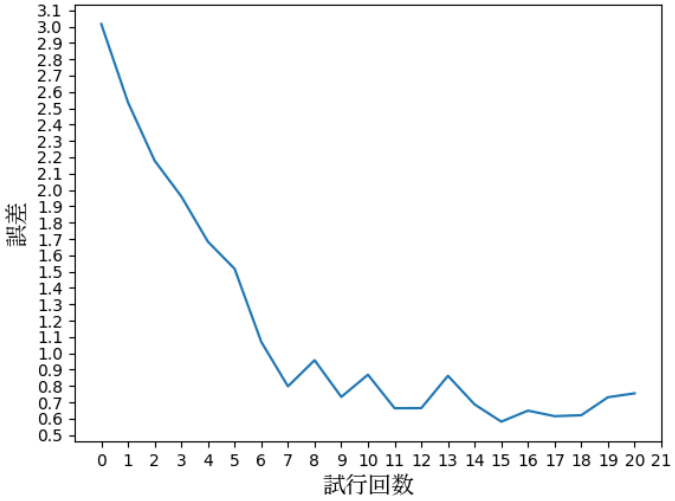



図 4.2: 実験 1 に用いた機械学習における誤差の推移

学習結果済みモデルに表 3.1 のテスト用データを入力しパラメータを判別した結果、表 4.1 のようになった。

表 4.1: 実験 1 に表 3.1 のテスト用のデータを入力した結果

| 識別 | 出力 | 設定された Gain の値 |
|---|---|---|
| Y01 | 7.1 | 5.0 |
| Y02 | 4.8 | 2.0 |
| Y03 | 3.0 | 8.0 |
| Y04 | 7.3 | 1.0 |
| Y05 | 9.5 | 9.9 |

学習結果済みモデルに表 3.1 のテスト用データを入力しパラメータを判別した結果、同じ

ギターを使ったテストデータですら誤差が大きく、正しく判別できていないことが分かった。学習に用いたギターフレーズが 4 つしかないため、様々なフレーズに対応できていない可能性が考えられる。

## 4.3　実験 2:倍音成分に着目した実験

図 4.3 に機械学習のモデル図を示す。

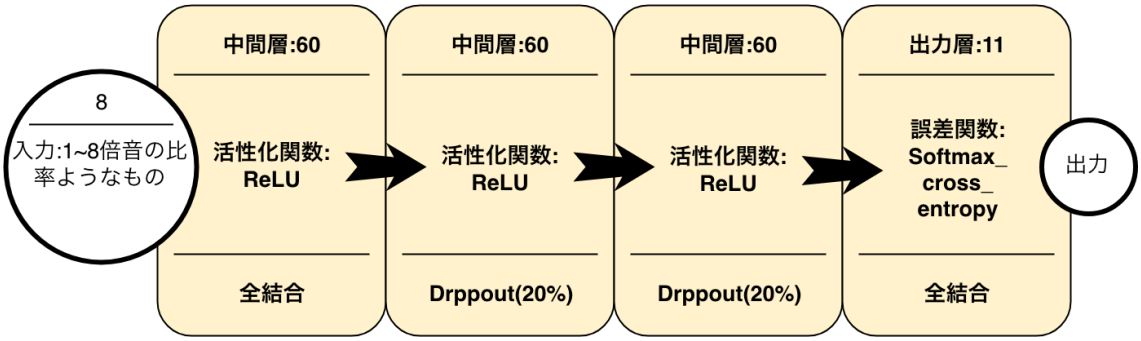



図 4.3: 実験 2 に用いた機械学習のモデル

一定の割合で一部の結合を遮断し過学習を防ぐ仕組みであるドロップアウトを含めた 4 層のニューラルネットワークを構築した。ソースコードについては付録 B.2 を参照されたい。

入力には振幅スペクトラムからギターの音程範囲（82Hz〜1245Hz）を参照し、例えば 82Hz を参照したならば、その 1 〜 8 倍音 [82Hz, 164Hz, 264Hz, 328Hz, 410Hz, 492Hz, 574Hz, 656Hz] の振幅が含まれている部分を取り出す。このような操作を 1245Hz まで繰り返し、取り出したものを平均化した 8 つのデータを用いた。

出力層は入力層を減らしたことや、アンプシミュレータの Gain 設定値による振幅スペクトラムの変化が少ないことを懸念し、0 〜 10 までの 11 段階とした。このようなモデルを用い、バッチサイズは 256 として、ミニバッチ学習を行った。

FFT の設定はサンプル数 8192 とし、サンプルを 75 %ずつずらして FFT を行い 10 個の振幅スペクトラムの平均を算出した。8192 サンプルで FFT を行ったため約 5.4[Hz] の幅で、22050[Hz] までの 4096 段階の振幅スペクトラムを計算することができる。

## 4.4　実験 2 の結果と考察

バッチ毎に検証用データセットで誤差を計算すると、図 4.4 のように約 600 回の学習で誤差が 1.0 近くまで収束した。

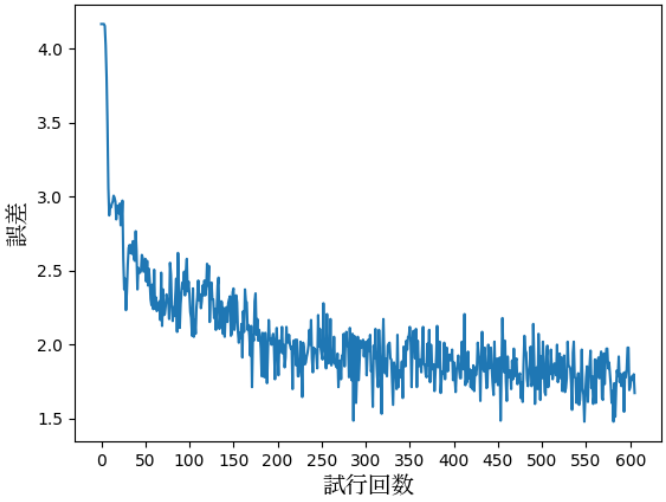



図 4.4: 実験 2 に用いた機械学習における誤差の推移

表 3.1 のテスト用データを入力した場合の結果は表 4.2 のようになった。

表 4.2: 実験 2 に用いた機械学習のモデルに新たに録音した物を入力

| 識別 | 出力 | 設定された Gain の値 | 識別 | 出力 | 設定された Gain の値 |
|---|---|---|---|---|---|
| Y01 | 5 | 5.0 | G01 | 1 | 5.0 |
| Y02 | 5 | 2.0 | G02 | 1 | 2.0 |
| Y03 | 9 | 8.0 | G03 | 7 | 8.0 |
| Y04 | 0 | 1.0 | G04 | 2 | 1.0 |
| Y05 | 5 | 9.9 | G05 | 5 | 9.9 |

Y01、Y03～Y04、G02～G04 についてはそれぞれ設定した Gain 値に近い値を出力しているが、Y02、Y05、G01、G05 については設定した Gain 値と大きく外れた値を出力している。また、聴感上では Y02 と G01 の出力が、正解の値の音と大きく違っている。

実験 1 より正解に近い値を出せている可能性もあるが、まだ精度が高いとは言えず、倍音の比率のようなものを用いることも限界があると考えた。

## 4.5　実験 1 及び実験 2 の結果からの考察

振幅スペクトラムについて、本研究に用いたアンプシミュレータの、Gain 設定値の変化による振幅スペクトラムの変化具合がそれほど大きくないのではないかという問題が考えられた。

よって、アンプシミュレータの Gain 設定値による振幅スペクトラムの変化を見ることにした。

一つのギターフレーズについて、Gain 設定値 0.0 ～ 2.5 における振幅スペクトラムを 0.1 刻みで計算し三次元グラフを作成した結果が図 4.5 である。

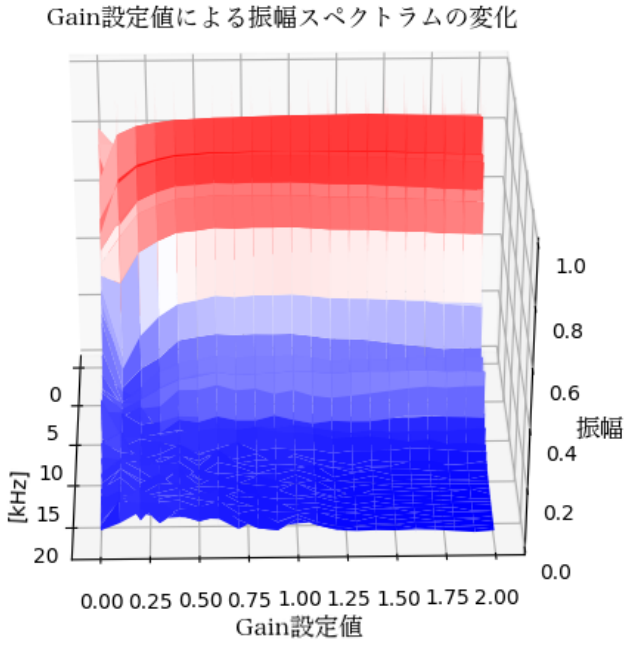



図 4.5: アンプシミュレータの Gain 設定値による振幅スペクトラムの変化

Gain 設定値が 0.0～0.5 辺りまでは倍音成分が滑らかに増えていく様子が見えるが、それ以上の Gain 設定値においては規則性がないことが分かる。しかしながら、聴感上は Gain 設定値が 0.5 以上になっても音に変化が感じられ、特に単音を多く使ったフレーズに顕著にその傾向が見られる。

以上から本研究に用いたアンプシミュレータの Gain の特性として、歪み効果が与える倍音の変化はすぐに飽和してしまうことが分かった。よって、クリッピングの量で変化する歪みの持続時間などの他の要因による音の変化の方が大きい可能性が考えられる。このことから、振幅スペクトルのみを用いた学習から歪みエフェクターの歪み量を抽出することは限界があるのではないかと考えられる。

## 4.6　実験 3:クリッピングに着目した実験

この実験では、歪みエフェクターのクリッピング処理に着目した実験を行った。

クリッピング処理をすると-1 ～ 1 の間に正規化された波形の面積は増えることになる。よって、その振幅の平均値が歪みに関係があると考えた。

この関係を調べるために、各フレーズの各 Gain 設定値における振幅の平均値の推移を調べた結果、図 4.6 のようになった。

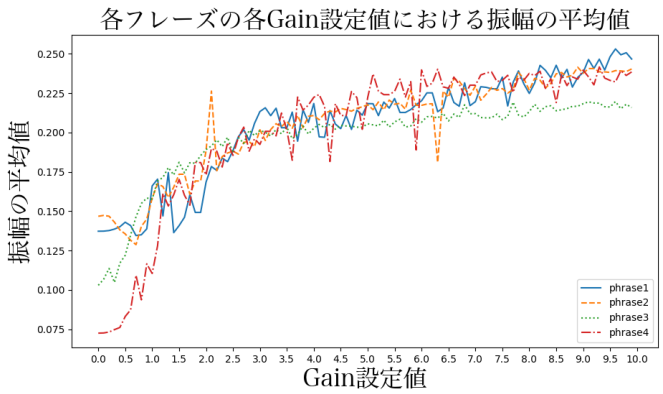



図 4.6: 各フレーズの各 Gain 設定値における振幅の平均値の推移

Gain 設定値が 1.5 の辺りで倍音成分の変化がなくなった振幅スペクトラムに対し、振幅の平均値は Gain 設定値が 10 になるまで段階的に増えていて、聴感上の Gain 設定値による音の変化と似ているように思われた。従って、この値を NN に入力し Gain 設定値と紐づけ、回帰直線を求める事にした。

図 4.7 に機械学習のモデル図を示す。

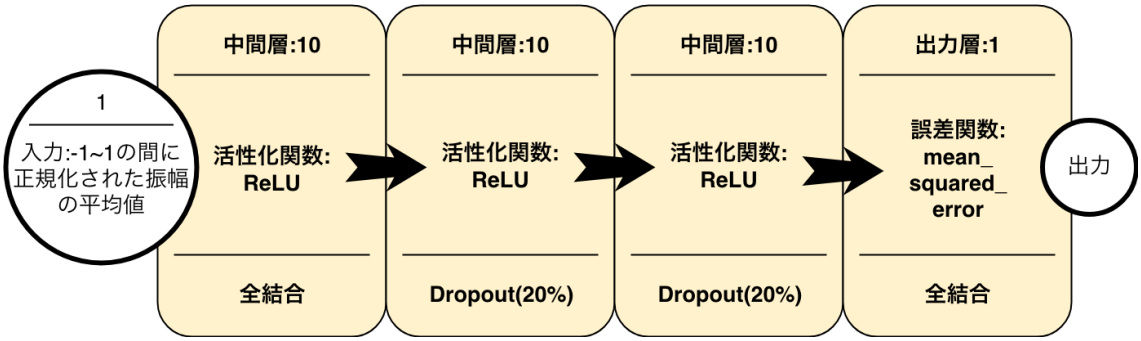



図 4.7: 実験 3 に用いた機械学習のモデル

一定の割合で一部の結合を遮断し過学習を防ぐ仕組みであるドロップアウトを含めた 4 層のニューラルネットワークを構築した。ソースコードについては付録 B.3 を参照されたい。

入力には-1 ～ 1 の間に正規化された波形の振幅の平均値、出力には NN の収束が遅くなったため Gain 設定値を 10 で割った物を使用した。誤差関数には平均二乗誤差を用い、バッ

チサイズは 8 としてミニバッチ学習を行った。

## 4.7　実験 3 の結果と考察

バッチ毎に検証用データセットで誤差を計算すると、図 4.8 のように約 2000 回の学習で誤差が 1.5 近くまで収束した。

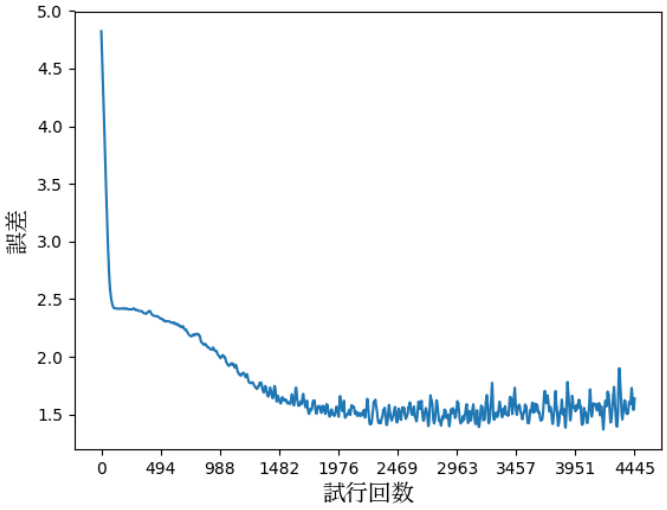



図 4.8: 実験 3 に用いた機械学習における誤差の推移

また、図 4.9 にこの NN を用いて導かれた回帰直線を示す。

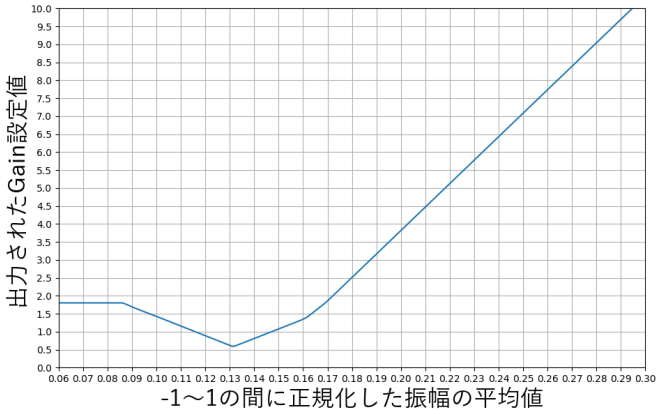



図 4.9: 実験 3 で導かれた回帰直線

-1 〜 1 の間に正規化された波形の振幅の平均値が 0.13 より低い値の場合に、出力される Gain 設定値が少し上がってしまうのは、歪みがあまりかかっていない時の振幅の平均値が

フレーズによって差が大きいからだと考えられる。これは、より多くのフレーズを学習に用いるか、フレーズを一つだけに絞って学習する事で解消するものと思われる。

次に、表 3.1 のテスト用データを入力した場合の結果は表 4.3 のようになった。

表 4.3: 実験 3 のモデルに表 3.1 のテスト用のデータを入力した結果

| 識別 | 出力 | 設定された Gain の値 | 識別 | 出力 | 設定された Gain の値 |
|---|---|---|---|---|---|
| Y01 | 7.0 | 5.0 | G01 | 5.2 | 5.0 |
| Y02 | 3.1 | 2.0 | G02 | 2.5 | 2.0 |
| Y03 | 4.4 | 8.0 | G03 | 4.1 | 8.0 |
| Y04 | 1.4 | 1.0 | G04 | 1.0 | 1.0 |
| Y05 | 4.5 | 9.9 | G05 | 2.2 | 9.9 |

Y01、Y02、Y04、G01、G02、G04 については設定した Gain 値に近い値を出力し、聴覚上でも近い歪みを感じられた。Y03、Y05、G03、G05 については設定した値と比べ、大きく外れた値を出力したが、聴感上では NN から出力された Gain 設定値の音と、正解の Gain 設定値の音を比べても歪み度合いに大差がなかった。

これらの結果から聴感上では実験 1 よりも良い結果を出していると考えられる。

## 4.8　全ての実験を通しての考察

用いるギターなどによって音響的特徴が違い、それに伴いアンプシミュレータの Gain 設定値による音の変化も変わってくることが考えられた。音響的特徴の違いをもたらすものとしてギターのピックアップが挙げられる。ピックアップとは、ギターによる振動を電気信号に変換する装置である。ピックアップには、2 つのコイルでノイズを打ち消しあうハムバッカー、ノイズがそのまま出力されるシングルコイルがある。歪み効果が大きくなるほどノイズ差も大きくなる事から、出力される音に大きな違いをもたらすと言える。

上記に記載した要因の他、懸念すべき点がいくつかある。ギター音の音程、使用する弦、弦の劣化具合はスペクトルに大きく影響を与える。また、ギターの共鳴作用や、ピッキングの位置、強さもスペクトルに影響を与えている。シールド・ギターの内部回路・エフェクターの内部回路の特性、演奏時の温度や湿度、奏者の気分といった外的要因によっても僅かながら影響が生じると考えた。このように様々な要因で音が変化するため、それを吸収するための工夫が必要である。

また、例えば Gain 設定値を 5.0 に設定しても、使用するギターや演奏するフレーズによって聴感上の歪みの度合が変化してしまうため、教師データの設定方法も工夫する必要がある

と考察した。

## 4.9　今後の課題

本研究で望ましい結果が出力されなかった原因に、学習させるデータの不足が考えられる。学習に用いるギター数、フレーズ数が増える程学習パターンが増え、学習結果が正解に近似していくと考えられる。

Convolutional Neural Network、CNN と呼ばれるディープラーニングのアルゴリズムがある。これは、画像の特徴を捉え、比較することで類似性を見出し判断する NN であり、これを用いることで音の波形自体を比較することが出来るのではないかと推測した。また、このアルゴリズムを用いる事で波形の細かい特徴を見出せるようになれば、本研究で用いた NN では判別しきれない違いも判別出来るのではないかと考察した。

# 第5章　まとめ

本研究では、振幅スペクトラムからはデータの少なさや倍音成分がすぐに飽和してしまうなどの理由から、学習用データのアンプシミュレータの Gain 設定値の分類しかできなかった。しかし、最終的に-1 ～ 1 の間に正規化した波形の振幅の平均値を用いる事で、多少誤差はあるが歪みの度合いを推測することができた。但し、歪み効果のかかり具合の絶対的な指標がない為、正しく教師データを設定することが出来ていなかったことが考えられる。

他にも、本研究では比較的新しい弦を張った物と錆びた弦を張った物を使用したが、弦が錆びると倍音成分が減り大きくスペクトラムが変化してしまう。よって、振幅スペクトラムを用いた実験では、新品の弦を張ったギターのみでのサンプリングが好ましいことも考えられた。

最後に本研究を通して、歪みの度合は似せることができても音自体は似せることができなかったことから、ギターの根幹となる音色は元のギター本体の性質に強く現れることが分かった。幾らエフェクターやアンプの設定値を調節しても、ギターの根幹となる音色を変えることはできない。従って、プロのギター音を模倣する為にはプロの演奏環境と同じ条件を揃えるか、ギターの根幹となる音色を変更する仕組みが必要であると考えた。

# 参考文献

[1] 【今さら聞けない】エフェクターの基礎知識編 ～歪み系エフェクター～ https://info.shimamura.co.jp/guitar/feature/basic-knowledge-of-effector-hizumi/

[2] ファズ・ディストーション・オーバードライブの歪み系エフェクターの違いとは？ http://www.riyu-haru.com/guitar/sound/drivetype

[3] エフェクター用語について、ちょっとマニアックな豆知識 https://guitar-hakase.com/9483/

[4] Chainer v4 ビギナー向けチュートリアル https://qiita.com/mitmul/items/1e35fba085eb07a92560

[5] 【Python】WAV ファイルの波形データに FFT かけて周波数スペクトルを複数表示する【サウンドプログラミング】http://tacky0612.hatenablog.com/entry/2017/12/12/174405

# 付 録A　automatorの操作手順

Amplitube4 の操作画面を A.1 示す。


図 A.1: Amplitube4 の操作画面

GarageBand の操作画面を A.2 に示す。


図 A.2: GarageBand の操作画面

Automator に設定した操作手順

1. Amplitube4 を手前に表示する
2. Amplitube4 の Gain つまみをクリック
3. Amplitube4 の Gain の数値表示部分をクリック
4. 以下のように記載した Apple Script を実行することにより、0.0〜9.9 のランダムな値を生成し Amplitube4 の Gain つまみのパラメータを変更すると同時に、その値の頭に”G”を付加したものをクリップボードに保存する。

```
--0から 9の乱数を二つ用意
set gain1 to getrandom() as string
set gain2 to getrandom() as string



--乱数を発生させる
on getrandom()
        return random number from 0 to 9
end getrandom


--クリップボードに保存
set ExportClipBoard to "G" & gain1 & "." & gain2
set the clipboard to ExportClipBoard

--キーボード入力
tell application "System Events"
        delay 0.5
        keystroke gain1 & "." & gain2
        keystroke (ASCII character 13)
end tell
```

5. テキストエディタを手前に表示する
6. 以下のように記載した Apple Script を実行することにより、現在時刻を入力した後、クリップボードに保存していたもの貼り付ける。

```
--現在時刻を取得
set d to (time of (current date))

--現在時刻をキーボード入力し、command+v による貼り付けを行う
tell application "System Events"
        delay 0.5
        keystroke d
```

```
8         delay 0.5
9         keystroke "v" using {command down}
10
11 end tell
```

7. Amplitube4 ををを手前に表示する
8. Amplitube4 の Spring Reverb つまみをクリック
9. Amplitube4 の Spring Reverb の数値表示部分をクリック
10. 以下のように記載した Apple Script を実行することにより、0.0〜9.9 のランダムな値を生成し Amplitube4 の Spring Reverb つまみのパラメータを変更すると同時に、その値の頭に”R”を付加したものをクリップボードに保存する。

```
1  --0から 9の乱数を二つ用意
2  set reverb1 to getrandom() as string
3  set reverb2 to getrandom() as string
4
5  --乱数を発生させる
6  on getrandom()
7         return random number from 0 to 9
8  end getrandom
9
10
11 --クリップボードに保存
12 set ExportClipBoard to "R" & reverb1 & "." & reverb2
13 set the clipboard to ExportClipBoard
14
15
16 --キーボード入力
17 tell application "System Events"
18        delay 0.5
19        keystroke reverb1 & "." & reverb2
20        keystroke (ASCII character 13)
21 end tell
```

11. テキストエディタを手前に表示する
12. 以下のように記載した Apple Script を実行することにより、クリップボードに保存していたものを貼り付ける。

```
1  --command+v による貼り付け
2  tell application "System Events"
3         keystroke "v" using {command down}
4  end tell
```

13. テキストエディタのテキスト領域をクリック

14. 以下のように記載した Apple Script を実行することにより、テキストエディタに入力したもの全てをクリップボードにコピーする。

```
1 --command+a による全選択と command+
      c によるクリップボードへのコピー、さらに delete キー入力によりテキスト領域をクリア

2 tell application "System Events"
3         keystroke "a" using {command down}
4         keystroke "c" using {command down}
5         keystroke (ASCII character 127)
6 end tell
```

15. Garageband 内の音声波形部分をクリック

16. Garageband のメニュー内から”共有”をクリック

17. 共有メニュー内から”曲をディスクに書き出す”をクリック

18. 以下のように記載した Apple Script を実行することにより、保存ファイル名を変更するためにクリップボードに保存していたものを貼り付ける。

```
1 --command+v による貼り付け
2 tell application "System Events"
3         keystroke "v" using {command down}
4
5 end tell
```

19. 曲をディスクに書き出すのメニューから”書き出す”ボタンをクリック

20. 最初に戻ってループする

# 付 録B　実行環境及び各実験におけるソースコード

実行環境は pyCharm という python の統合開発環境である。機械学習を行う上では chainer というライブラリを用いた。

chainer による NN の作成においては参考文献 [4] を、FFT の処理においては参考文献 [5] を参考に構築した。

実験 1 におけるソースコードを、B.1 に示す。

ソースコード B.1: 実験 1 のソースコード

```
1  import os
2  import numpy as np
3  import chainer
4  from chainer import serializers
5  from chainer import optimizers
6  import chainer.functions as F
7  import chainer.links as L
8  import random
9  from audio_process import add_fft
10 from audio_process import wave_length
11 import pickle
12 import math
13
14 overload = 75
15 amount = 10
16 size = 2**13
17
18 #音声ファイルのランダムな場所の振幅スペクトラムを取得するときにデータサイ
       ズを超さないための数値
19 margin = int(size + size * (overload / 100) * (amount - 1))
20
21 output_size = 100
22 hidden_size = 2100
23
24 repeat = 10
25
26 train_folder = "Guiter"
27 valid_folder = "Validation"
```

```
28
29
30 def make_teacher(num):
31         return num
32
33
34 save_name = "exam1"
35
36 save_model = './model/' + save_name + '.model'
37 save_progress_file = './progress/' + save_name + '.txt'
38
39 #バッチの設定
40 batch_size = 1
41 bt_idx = 0
42
43 def calc_idx(freq):
44         idx = int(freq / freq_width)
45         return idx
46
47 class MyChain(chainer.Chain):
48         def __init__(self):
49         super(MyChain, self).__init__()
50
51         with self.init_scope():
52
53                 self.l1 = L.Linear(None, hidden_size)
54                 self.l2 = L.Linear(hidden_size, hidden_size)
55                 self.l3 = L.Linear(hidden_size, hidden_size)
56                 self.l4 = L.Linear(hidden_size, output_size)
57
58         def __call__(self, x):
59         h1 = F.relu(self.l1(x))
60         h2 = F.relu(self.l2(h1))
61         h3 = F.relu(self.l3(h2))
62         o = self.l4(h3)
63         return o
64
65
66 gpu_id = -1
67 net = MyChain()
68 if gpu_id >= 0:
69         net.to_gpu(gpu_id)
70
71 # Setup optimizer
72 optimizer = optimizers.SGD()
73 optimizer.setup(net)
74
75 wav_files = []
```

```
76 for x in os.listdir(train_folder):
77         if (x[-4:] == ".wav"):
78         wav_files.append(x)
79
80 # 検証セットの作成
81 valid_files = []
82 for x in os.listdir(valid_folder):
83         if (x[-4:] == ".wav"):
84         valid_files.append(x)
85
86
87 # 検証
88 progress = []
89 valid_inputs = []
90 valid_t = []
91
92 def validation():
93         error = 0
94
95         accuracy_count = 0
96         loss_avg = 0
97
98         for l in range(len(valid_files)):
99         valid_inputs.clear()
100        valid_t.clear()
101
102        start = random.randrange(wave_length(valid_folder+"/" +
               valid_files[l]) - margin)
103        valid_spect = add_fft(valid_folder+"/" + valid_files[l], overload
               , size, start, amount)
104        valid_inputs.append(valid_spect)
105        valid_t.append(make_teacher(int(float(valid_files[l
               ][-11:-8])*10)))
106        # chainer で使えるようにデータセットの型を変換
107        valid_inputs_np = np.array(valid_inputs, dtype=np.float32)
108        valid_t_np = np.array(valid_t, dtype=np.int32)
109
110
111        # 予測精度
112
113        with chainer.using_config('train', False), \
114                chainer.using_config('enable_backprop', False):
115                y_valid = net(valid_inputs_np)
116        # ロスを計算
117        loss_valid = F.softmax_cross_entropy(y_valid, valid_t_np)
118        loss_avg += loss_valid
119
120        this_accuracy = F.accuracy(y_valid, valid_t_np)
```

```
121         accuracy_count += this_accuracy
122
123         maximum = 0
124         maximum_idx = 0
125         for d in range(len(y_valid[0])):
126                 if maximum < y_valid[0][d].data:
127                 maximum = y_valid[0][d].data
128                 maximum_idx = d
129         print("output=" + str(maximum_idx) + ", correct=" + str(
                valid_t_np))
130
131         error += math.sqrt(abs(maximum_idx - valid_t_np[0])**2)
132
133         print("error_avg = " + str(error / float(len(valid_files))))
134         return error / float(len(valid_files))
135
136
137 #学習セットをランダムに取り出すためのインデックスの配列を作成
138 index = [i for i in range(len(wav_files))]
139 random.shuffle(index)
140
141
142 #学習データセットの作成
143 inputs = []
144 t = []
145 while bt_idx+batch_size < len(wav_files):
146         #net.reset_state()
147         inputs.clear()
148         t.clear()
149
150         for l in range(bt_idx,bt_idx+batch_size):
151         start = random.randrange(wave_length(train_folder+"/" + wav_files[
                index[l]]) - margin)
152         spect = add_fft(train_folder+"/" + wav_files[index[l]], overload,
                 size, start, amount)
153
154         inputs.append(spect)
155         t.append(make_teacher(int(float(wav_files[index[l
                ]][-11:-8])*10)))
156
157         # chainer で使えるようにデータセットの型を変換
158         inputs_np = np.array(inputs, dtype=np.float32)
159         t_np = np.array(t, dtype=np.int32)
160
161         # NN に入力し順方向伝搬
162         y = net(inputs_np)
163         # 損失を計算
164         loss = F.softmax_cross_entropy(y, t_np)
```

```
165         #print(loss)
166         net.cleargrads()
167         # 逆方向伝搬
168         loss.backward()
169         # 学習率を最適化
170         optimizer.update()
171         #progress.append(validation())
172 
173         # バッチの更新
174         bt_idx += batch_size
175         if bt_idx+batch_size >= len(wav_files):
176         repeat -= 1
177         print(repeat)
178 
179         if repeat > 0:
180                 bt_idx = 0
181                 progress.append(validation())
182                 random.shuffle(index)
183 
184         elif repeat == 0:
185                 print("How many times do you wanna repeat? : ")
186                 s = input()
187 
188                 if s != "0":
189                 bt_idx = 0
190                 repeat = int(s)
191                 random.shuffle(index)
192                 else:
193                 progress.append(validation())
194 
195 
196 file = open(save_progress_file, 'wb')
197 pickle.dump(progress, file)
198 
199 serializers.save_npz(save_model, net)
```

実験 2 におけるソースコードを、B.2 に示す。

ソースコード B.2: 実験 2 のソースコード

```
1 import os
2 import numpy as np
3 import chainer
4 from chainer import serializers
5 from chainer import optimizers
6 import chainer.functions as F
7 import chainer.links as L
8 import random
9 from audio_process import add_fft
```

```
10 from audio_process import wave_length
11 import pickle
12 import math
13
14 overload = 75
15 amount = 10
16 size = 2**13
17 #音声ファイルのランダムな場所の振幅スペクトラムを取得するときにデータサイ
       ズを超さないための数値
18 margin = int(size + size * (overload / 100) * (amount - 1))
19
20 samp_freq = 44100
21 freq_width = samp_freq / size
22
23 output_size = 11
24 hidden_size = 60
25
26 repeat = 1
27
28 train_folder = "Guiter"
29 valid_folder="Validation"
30
31 def make_teacher(num):
32         return int(num/10+0.5)
33
34 save_name = "exam2"
35
36 save_model = './model/' + save_name + '.model'
37 save_progress_file = './progress/' + save_name + '.txt'
38
39
40 #バッチの設定
41 batch_size = 256
42 bt_idx = 0
43
44 def calc_idx(freq):
45         idx = int(freq / freq_width)
46         return idx
47
48 def calc_harmonic(freq):
49         list = []
50         for i in range(1, 9):
51         list.append(calc_idx(freq * i))
52         return list
53
54
55 def harmonic_avg(spectrum, first_freq, last_freq):
56         harm_num = 8
```

```
57          list = []
58          list.append(calc_harmonic(first_freq))
59          for i in range(first_freq + 1, last_freq):
60          if calc_harmonic(i)[0] != list[-1][0]:
61                  list.append(calc_harmonic(i))
62
63
64          harm_avg = []
65          for lj in range(harm_num):
66          vol = 0
67          for l in list:
68                  vol += spectrum[l[lj]] * spectrum[l[0]]**2
69          harm_avg.append(vol)
70          return harm_avg / max(harm_avg)
71
72
73
74  class MyChain(chainer.Chain):
75          def __init__(self):
76          super(MyChain, self).__init__()
77
78          with self.init_scope():
79
80                  self.l1 = L.Linear(None, hidden_size)
81                  self.l2 = L.Linear(hidden_size, hidden_size)
82                  self.l3 = L.Linear(hidden_size, hidden_size)
83                  self.l4 = L.Linear(hidden_size, output_size)
84
85          def __call__(self, x):
86          h1 = F.relu(self.l1(x))
87          h2 = F.dropout(F.relu(self.l2(h1)), 0.2)
88          h3 = F.dropout(F.relu(self.l3(h2)), 0.2)
89          o = self.l4(h3)
90          return o
91
92  gpu_id = -1
93  net = MyChain()
94  if gpu_id >= 0:
95          net.to_gpu(gpu_id)
96
97  # Setup optimizer
98  optimizer = optimizers.Adam()
99  optimizer.setup(net)
100
101
102 wav_files = []
103 for x in os.listdir(train_folder):
104         if (x[-4:] == ".wav"):
```

```
105         wav_files.append(x)
106
107 # 検証セットの作成
108 valid_files = []
109 for x in os.listdir(valid_folder):
110         if (x[-4:] == ".wav"):
111         valid_files.append(x)
112
113
114 # 検証
115 progress = []
116 valid_inputs = []
117 valid_t = []
118
119 def validation():
120         error = 0
121
122         accuracy_count = 0
123         loss_avg = 0
124
125         for l in range(len(valid_files)):
126         #net.reset_state()
127         valid_inputs.clear()
128         valid_t.clear()
129         start = random.randrange(wave_length(valid_folder+"/" +
                valid_files[l]) - margin)
130         valid_spect = np.log(add_fft(valid_folder+"/" + valid_files[l],
                overload, size, start, amount))
131         valid_spect = (valid_spect - min(valid_spect))
132         valid_spect /= max(valid_spect)
133
134         valid_inputs.append(harmonic_avg(valid_spect, 82, 1245))
135         valid_t.append(make_teacher(int(float(valid_files[l
                ][-11:-8])*10)))
136         # chainer で使えるようにデータセットの型を変換
137         valid_inputs_np = np.array(valid_inputs, dtype=np.float32)
138         valid_t_np = np.array(valid_t, dtype=np.int32)
139
140
141         # 予測精度
142
143         with chainer.using_config('train', False), \
144                 chainer.using_config('enable_backprop', False):
145                 y_valid = net(valid_inputs_np)
146         # ロスを計算
147         loss_valid = F.softmax_cross_entropy(y_valid, valid_t_np)
148         loss_avg += loss_valid
149
```

```
150         this_accuracy = F.accuracy(y_valid, valid_t_np)
151         accuracy_count += this_accuracy.data
152 
153         maximum = 0
154         maximum_idx = 0
155         for d in range(len(y_valid[0])):
156                 if maximum < y_valid[0][d].data:
157                 maximum = y_valid[0][d].data
158                 maximum_idx = d
159         print("output=" + str(maximum_idx) + ", correct=" + str(
                valid_t_np))
160 
161         error += math.sqrt(abs(maximum_idx - valid_t_np[0])**2)
162 
163         print("error_avg = " + str(error / float(len(valid_files))) + " ,
                acuracy:", + accuracy_count/float(len(valid_files)))
164         return error / float(len(valid_files))
165 
166 
167 
168 #学習セットをランダムに取り出すためのインデックスの配列を作成
169 index = [i for i in range(len(wav_files))]
170 random.shuffle(index)
171 
172 
173 #学習データセットの作成
174 
175 inputs = []
176 t = []
177 while bt_idx+batch_size < len(wav_files):
178         #net.reset_state()
179         inputs.clear()
180         t.clear()
181 
182         for l in range(bt_idx,bt_idx+batch_size):
183         start = random.randrange(wave_length(train_folder + "/" +
                wav_files[index[l]]) - margin)
184         spect = add_fft(train_folder+"/" + wav_files[index[l]], overload,
                 size, start, amount)
185         inputs.append(harmonic_avg(spect, 82, 1245))
186         t.append(make_teacher(int(float(wav_files[index[l
                ]][-11:-8])*10)))
187 
188         # chainer で使えるようにデータセットの型を変換
189         inputs_np = np.array(inputs, dtype=np.float32)
190         t_np = np.array(t, dtype=np.int32)
191 
192         # NN に入力し順方向伝搬
```

```
193         y = net(inputs_np)
194         # 損失を計算
195         loss = F.softmax_cross_entropy(y, t_np)
196         #print(loss)
197         net.cleargrads()
198         # 逆方向伝搬
199         loss.backward()
200         # 学習率を最適化
201         optimizer.update()
202         #progress.append(validation())
203
204
205         # バッチの更新
206         bt_idx += batch_size
207         if bt_idx+batch_size >= len(wav_files):
208         repeat -= 1
209         print(repeat)
210
211         if repeat > 0:
212                 bt_idx = 0
213                 progress.append(validation())
214                 random.shuffle(index)
215
216         elif repeat == 0:
217                 print("How many times do you wanna repeat? : ")
218                 s = input()
219
220                 if s != "0":
221                 bt_idx = 0
222                 repeat = int(s)
223                 random.shuffle(index)
224                 else:
225                 progress.append(validation())
226
227
228 file = open(save_progress_file, 'wb')
229 pickle.dump(progress, file)
230
231 serializers.save_npz(save_model, net)
```

実験 3 におけるソースコードを、B.3 に示す。

ソースコード B.3: 実験 3 のソースコード

```
1 import os
2 from pydub import AudioSegment
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import chainer
```

```
from chainer import serializers
from chainer import optimizers
import chainer.functions as F
import chainer.links as L
import random
from audio_process import volume_avg as vc

import pickle
import math


output_size = 1
hidden_size = 10

repeat = 100

train_folder = "Guiter"
valid_folder = "Validation"


def make_teacher(num):
        return num/10


save_name = "exam3"

save_model = './model/' + save_name + '.model'
save_progress_file = './progress/' + save_name + '.txt'

#バッチの設定
batch_size = 8
bt_idx = 0

class MyChain(chainer.Chain):
        def __init__(self):
        super(MyChain, self).__init__()

        with self.init_scope():

                self.l1 = L.Linear(None, hidden_size)
                self.l2 = L.Linear(hidden_size, hidden_size)
                self.l3 = L.Linear(hidden_size, hidden_size)
                self.l4 = L.Linear(hidden_size, output_size)

        def __call__(self, x):
        h1 = F.relu(self.l1(x))
        h2 = F.dropout(F.relu(self.l2(h1)), 0.2)
        h3 = F.dropout(F.relu(self.l3(h2)), 0.2)
```

```
54          o = self.l4(h3)
55          return o
56
57  gpu_id = -1
58  net = MyChain()
59  if gpu_id >= 0:
60          net.to_gpu(gpu_id)
61
62  # Setup optimizer
63  optimizer = optimizers.Adam()
64  optimizer.setup(net)
65
66  wav_files = []
67  for x in os.listdir(train_folder):
68          if (x[-4:] == ".wav"):
69          wav_files.append(x)
70
71  # 検証セットの作成
72  valid_files = []
73  for x in os.listdir(valid_folder):
74          if (x[-4:] == ".wav"):
75          valid_files.append(x)
76
77
78  # 検証
79  progress = []
80  valid_inputs = []
81  valid_t = []
82
83  def validation():
84          error = 0
85
86          accuracy_count = 0
87          loss_avg = 0
88
89          for l in range(len(valid_files)):
90          #net.reset_state()
91          valid_inputs.clear()
92          valid_t.clear()
93          valid_inputs.append([vc(valid_folder + "/" + valid_files[l])])
94          valid_t.append([make_teacher(float(valid_files[l][-11:-8]))])
95          # chainer で使えるようにデータセットの型を変換
96          valid_inputs_np = np.array(valid_inputs, dtype=np.float32)
97          valid_t_np = np.array(valid_t, dtype=np.float32)
98
99
100         # 予測精度
101
```

```
102         with chainer.using_config('train', False), \
103                 chainer.using_config('enable_backprop', False):
104                 y_valid = net(valid_inputs_np)
105         # ロスを計算
106         loss_valid = F.mean_squared_error(y_valid, valid_t_np)
107         loss_avg += loss_valid
108 
109         #this_accuracy = F.accuracy(y_valid, valid_t_np)
110         #accuracy_count += this_accuracy
111 
112         print("output=" + str(y_valid.data[0]) + ", correct=" + str(
                valid_t_np[0]))
113         error += math.sqrt(abs(y_valid.data[0] - valid_t_np[0])**2)
114 
115         print("error_avg = " + str(error / float(len(valid_files))))
116         return error / float(len(valid_files))
117 
118 
119 
120 #学習セットをランダムに取り出すためのインデックスの配列を作成
121 index = [i for i in range(len(wav_files))]
122 random.shuffle(index)
123 
124 
125 #学習データセットの作成
126 inputs = []
127 t = []
128 while bt_idx+batch_size < len(wav_files):
129         inputs.clear()
130         t.clear()
131 
132         for l in range(bt_idx, bt_idx+batch_size):
133         inputs.append([vc(train_folder + "/" + wav_files[index[l]])])
134         t.append([make_teacher(float(wav_files[index[l]][-11:-8]))])
135 
136         # chainer で使えるようにデータセットの型を変換
137         inputs_np = np.array(inputs, dtype=np.float32)
138         t_np = np.array(t, dtype=np.float32)
139 
140         # NN に入力し順方向伝搬
141         y = net(inputs_np)
142         #print(y)
143         # 損失を計算
144         loss = F.mean_squared_error(y, t_np)
145         #print(loss)
146         net.cleargrads()
147         # 逆方向伝搬
148         loss.backward()
```

```
149         # 学習率を最適化
150         optimizer.update()
151         #progress.append(validation())
152 
153         # バッチの更新
154         bt_idx += batch_size
155         if bt_idx + batch_size >= len(wav_files):
156         repeat -= 1
157         print(repeat)
158 
159         if repeat > 0:
160                 bt_idx = 0
161                 progress.append(validation())
162                 random.shuffle(index)
163 
164         elif repeat == 0:
165                 print("How many times do you wanna repeat? : ")
166                 s = input()
167 
168                 if s != "0":
169                 bt_idx = 0
170                 repeat = int(s)
171                 random.shuffle(index)
172                 else:
173                 progress.append(validation())
174 
175 
176 file = open(save_progress_file, 'wb')
177 pickle.dump(progress, file)
178 
179 serializers.save_npz(save_model, net)
```

音声を処理する関数をまとめた物を、B.4 に示す。

ソースコード B.4: audio_process

```
1  import wave
2  import numpy as np
3  import matplotlib.pyplot as plt
4  
5  
6  def wave_length(filename):
7          w = wave_load(filename)
8          return len(w)
9  
10 def volume_avg(filename):
11         data = wave_load(filename)
12 
13         sum = 0
```

```
14          silent = 0
15
16          for i in range(len(data)):
17          if abs(data[i]) > 0.05:
18                  sum += abs(data[i])
19          else:
20                  silent += 1
21
22          return sum / (len(data) - silent)
23
24
25  def wave_load(filename):
26          # open wave file
27          wf = wave.open(filename, 'r')
28          channels = wf.getnchannels()
29
30          # load wave data
31          chunk_size = wf.getnframes()
32          amp = (2 ** 8) ** wf.getsampwidth() / 2
33          data = wf.readframes(chunk_size) # バイナリ読み込み
34          data = np.frombuffer(data, 'int16') # int に変換
35          data = data / amp # 振幅正規化
36          data = data[::channels]
37
38          return data
39
40
41  def fft_load(filename, size, st):
42          # st=サンプリングする開始位置 size = FFT のサンプル数（2＊＊n）
43
44          wave = wave_load(filename)
45
46          hammingWindow = np.hamming(size) # ハミング窓
47          fs = 44100 # サンプリングレート
48          d = 1.0 / fs # サンプリングレートの逆数
49          freqList = np.fft.fftfreq(size, d)
50          windowedData = hammingWindow * wave[st:st + size] # 切り出した波
                形データ(窓関数あり)
51          data = np.fft.fft(windowedData)
52
53          data = data / max(abs(data)) # 0~1正規化
54          #plt.plot(freqList, abs(data))
55          plt.axis([0, fs / 2, 0, 1]) # 第二引数でグラフのy 軸方向の範囲指定
56          plt.title(filename)
57          return abs(data[0:int(size/2)])
58
59  def add_fft(filename, overload, size, start, amount):
60          fs = 44100 # サンプリングレート
```

```
61          d = 1.0 / fs # サンプリングレートの逆数
62          freqList = np.fft.fftfreq(size, d)
63          f = fft_load(filename, size, start)
64          for i in range(amount-1):
65          start += int(size * overload/100)
66          work = fft_load(filename, size, start)
67          for l in range(len(work)):
68                  f[l] += work[l]
69
70          plt.axis([0, fs / 2, 0, 1])
71          return f/max(f)
```

# 謝 辞

本論文を作成するにあたり、ご指導を頂いた工学部情報工学科教員の松﨑博季教授に心より感謝致します。また、日常の議論を通じて多くの知識や示唆を頂戴し、共に切磋琢磨したゼミ員にも深く感謝致します。