公立はこだて未来大学　2016 年度　システム情報科学実習
グループ報告書
Future University Hakodate 2016 System Information Science Practice
Group Report

プロジェクト名
複雑系の数理とシミュレーション
Project Name
Mathematics and simulation of the complex system

プロジェクト番号/Project No.6

プロジェクトリーダー/Project Leader
104250　平井力　Chikara Hirai

グループリーダー/Group Leader

グループメンバー/Group Member
1013184　宮川聖士　Syoto Miyagawa
1014034　佐藤大生　Daiki Sato
1014040　庭田勘佑　Kansuke Niwata
1014069　笹田紘平　Kohei Sasada
1014086　池田陽彦　Akihiko Ikeda
1014101　水岡祐介　Yusuke Mizuoka

指導教員
斉藤朝輝　川越敏司　川口聡
Advisor
Asaki Saito　Toshiji Kawagoe　Satoshi Kawaguchi

提出日
2017 年 1 月 18 日
Date of Submission
January. 18, 2017

概要

外部からの情報を刺激として知覚し、それに対して判断や記憶をする人間の脳の働きの過程である認知過程を本プロジェクトで研究する。方法としては、ニューラルネットワークを使った認知過程のシミュレータを作成し、そのシミュレータを用いて数値実験を行うことで研究をする。ニューラルネットワークとは人間の脳の神経細胞の回路網を模倣したものである。神経細胞はニューロンと呼ばれていて、多数のニューロンが結合しネットワークを形成している。その繋がり方によって、ネットワークは多用の形態を持つ。具体的な例としては、表情の認識などである。

前期では基本的なニューラルネットワークの仕組みを理解することを課題とした。解決するために、ニューラルネットワークに関する書籍[1]をメンバー全員で輪読を行い学習した。

後期ではニューラルネットワークを用いた読唇術のシミュレータの作成を行った。読唇術とは人の発話を音声ではなく唇の動きだけで内容を認識する技術である。未来大生が日常的に発話する 5 音からなる 21 種類の単語のリストを作成した。その中から母音による判別を行った。また、母音を一音ずつ出力させてリスト内の単語とマッチングをとる A パターン、リスト内の単語を直接出力させる B パターンの２つのニューラルネットワークを作成した。実験とは別に Kinect を用いて、10 人にリスト内の単語を発話してもらい、学習用のデータとして取得した。実験として、10 人の被験者を募集し、リスト内の単語を発話してもらいそれぞれニューラルネットワークに 100 回入力を行い、比較した。そして、発話した単語と出力した単語が一致した回数から正答率を調べた。結果として、A パターンは 21%、B パターンは 45%となった。したがって、ニューラルネットワークを用いて読唇術を行った場合、単語を直接出力する B パターンが優れていることが分かった。

キーワード　　認知過程,ニューラルネットワーク,読唇術,Kinect

(※文責：佐藤大生)

Abstract

In this project, we study the cognitive process in human beings. The cognitive process in human beings is process of the workings of human brain that they perceive information from outside and judgment and memory. We make a simulation of cognitive process based on neural networks and we do numerical experiment to study the cognitive process. Neural network is model of nerve cells of the brain. Nerve cells are called neuron. Network is formed from combination of some neurons. Furthermore, form of network is decided by what combination. For example, facial expression.
In first semester, we read the book about neural network to learn the structure of basic neural network. In second semester, we created a simulator of lip leading using neural network. Lip leading is technique to recognize human speech only by movement of lips. We made the list of 21 word consisting 5 sounds that students of future university Hakodate seem to speak by daily life, and we did the distinction by vowel. Also, we created two neural network. Pattern A output vowels one by one. Then predict the content of utterance from the sequence of vowels. Pattern B output speech contents directly. We got data by kinect to learn neural network from 10 people. As an experiment, we input each neural network 100 times from other 10 people, and compared. We checked a correct answer rate by number of times that uttered word and outputted word match. In the result, pattern A is 21%, pattern B is 45%. Therefore, we understood that pattern B is superior to pattern A when doing lip reading using neural network.

Keyword cognitive process, neural network, lip reading, kinect

(※文責：佐藤大生)

目次

第 1 章 背景
1.1 該当分野の現状・従来例・・・5
1.2 本プロジェクトの目的・・・5
1.3 課題の概要・・・6

第 2 章 プロジェクト学習の概要
2.1 問題の設定・・・6
2.2 課題設定・・・6
2.3 到達レベル(目標)・・・7
2.4 課題の割り当て・・・7

第 3 章 課題解決のプロセス
3.1 課題解決のプロセスの概要・・・8
3.2 前期の主な活動・・・15
3.3 中間発表に向けての活動・・・16
3.4 最終発表に向けての活動・・・17
3.5 デモンストレーション・・・17

第 4 章 課題解決のプロセスの詳細
4.1 ニューラルネットワーク・・・18
4.2 Hopfield 型ネットワーク・・・19
4.2.1 ネットワークのエネルギー・・・20
4.3 Boltzmann Machine・・・21
4.4 読唇術について・・・22

第 5 章 実験
5.1 実験の目的・・・24
5.2 実験で使用した単語のリスト・・・25
5.3 実験装置について・・・25
5.4 実験の手順・・・25

第 6 章 成果と結果
6.1 前期の活動結果・・・27
6.2 後期の活動結果・・・27
6.3 成果の評価・・・28

第 7 章 今後の課題と展望
7.1 今後の課題・・・29

参考文献・・・30

付録:作成したプログラムのソースコード・・・30

第 1 章 背景

1.1　該当分野の現状・従来例

ニューラルネットワークとは、多くの比較的単純な情報処理要素(ユニット)が、相互に結合し簡単な信号をやりとりするような型のネットワーク上のメカニズムを利用することで情報処理を行う装置である[1](図 1 参照)。中でも機械学習の中でのニューラルネットワークは音声や映像などといったものを解析する装置への応用から、コンピュータ AI への活用など様々な分野で利用されている。具体的な例として、ディープラーニングを用いた人工知能に応用されている。

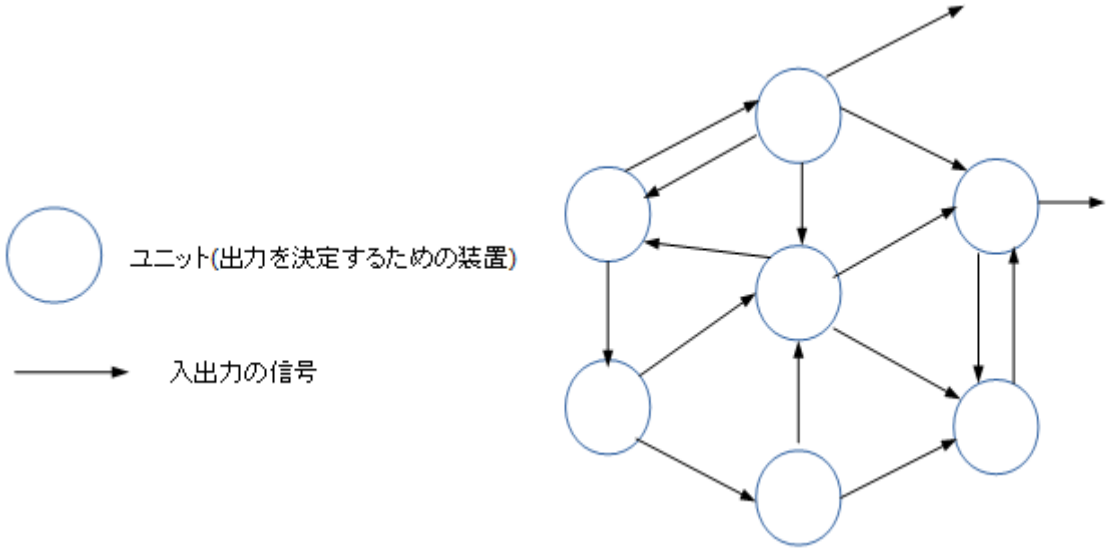

図 1:ニューラルネットワークの簡潔な模式図

（※文責：平井力）

1.2　本プロジェクトの目的

本プロジェクトの目的は、ニューラルネットワークを用いて人間の認知過程について調べることである。ニューラルネットワークの研究の目的は、人間の脳のモデル化を図り、人間の脳の働きを模倣することである。そこで本プロジェクトでは後述する読唇術と呼ばれる技術に着目し、ニューラルネットワークを用いて読唇術を実現するために必要な情報を調べることで、人間が読唇術を行うにあたってどういった情報が必要かを研究することを目標とした。

（※文責：平井力）

1.3　課題の概要

本プロジェクトは、メンバー全員がニューラルネットワークの数理を理解することから始めた。ニューラルネットワークに関する理解を深めるために文献[1]~[3]を利用し、輪読を行う中で参考書[1]内で登場した Hopfield 型ネットワークについてのプログラムの作成等を行うことによってメンバー全員で理解を深めることを前期の目標と設定し、活動を続けた。また、後期の活動では人間の認知過程を研究するにあたり、読唇術について注目しニューラルネットワークを用いて実現する為にはどういった情報が必要という研究を行った。

（※文責：平井力)

第 2 章 プロジェクト学習の概要

2.1　問題の設定

本プロジェクト学習では、グループ内で分担し輪読を行うことでニューラルネットワークの基本的な仕組みを理解し、ニューラルネットワークを使ったシミュレータの作成、それを利用して人間の認知過程に関する研究を目標とした 。そこで本プロジェクトではニューラルネットワークを用いて読唇術を実現させるプログラムを作成することを考えた。読唇術に着目した理由としては、読唇術を習得することができれば、耳が聞こえない人や公共の場で声を出すことが難しい場合でもコミュニケーションを図れるようになると考えたからである。そこで読唇術を実現させるためには口のどのような情報が重要なのか、どのようにすればより正確な読唇術を実現させることができるのか。これらの点について二種類のプログラムを作成し、実験を行い検証した。

(※文責：水岡祐介)

2.2　課題設定

前期では、後期のシミュレータ作成のために、輪読を通してニューラルネットワークについて理解することを課題として設定した。よって五月、六月はニューラルネットワークの基礎について文献[1]を用いて輪読をし理解を深めた。輪読の内容は階層型ネットワークや相互結合ネットワークのメカニズム、それに関する計算式についてである。また輪読した文献[1]に出てきた Hopfield 型ネットワークに関するプログラムを中間発表で行うデモプログラムとして作成した。この課題は前期に解決する課題として設定した。後期では、ニューラルネットワークを用いて読唇術を行うために、読唇術を実現させるプログラムの作成、さらに作成したプログラムを用いてどちらが読唇術を実現させるのに優れているかを比較した。この二つを後期の課題として設定した。まず一つ目の課題を解決するために、参考文献を読みどのようにして言葉を識別するかを考えた。この際に、口の縦幅と横幅の距離を図り面積の違いで言葉を識別するという先行研究である文献[4]を参考にした。そこから、あらかじめ決められた単語のリスト内からどの単語を発したか識別するプログラムを作成した。二つ目の課題を解決するために、作成したプログラムに被験者から取得したデータを入力し比較した。

(※文責：水岡祐介)

2.2.1 具体的な習得技術と解決過程について

前期のデモプログラム作成に関しては C 言語を用いて作成した。輪読では、個人の力では理解が困難な箇所が出てきた場合には、担当の教授に質問をする、または一番ニューラルネットワークの内容を理解していたメンバーに質問するなどして協力し合いながら進めていった。後期の課題については、読唇術のニューラルネットワークのプログラムを作成するにあたって C 言語を用いたので必要な知識はインターネットや参考書を利用して補った。また、口の面積を測定する際に Kinect を用いたので、同様にインターネットや参考書を利用して知識を補った。それでも補いきれなかった部分に関しては担当の教授や、昨年度のプロジェクトメンバーであった先輩に質問することで補った。
比較実験を行う際にはメンバー全員のデータ、実験協力者のデータを利用した。また、後期には複雑系コースの講義であるニューロコンピューティングを受講し学習した。

(※文責：水岡祐介)

2.3 到達レベル(目標)

本プロジェクトでは 2.2 で設定した課題を解決することによって人間の認知過程について調べられると考え、これらの課題を解決することを本プロジェクトの目標とした。ニューラルネットワークとは何かを理解するための輪読に関しては、プロジェクトメンバーがニューラルネットワークについての基本的な知識を身に付け理解する必要があった。ニューラルネットワークを用いた読唇術の実現に関しては Kinect を使用してでもプログラムを作成することを目標として設定した。読唇術のプログラムを利用した比較実験では、どちらのプログラムが読唇術をおこなうのにより優れているかを比較し考察することを目標として設定した。

(※文責：水岡祐介)

2.4 課題の割り当て

前期ではニューラルネットワークの基礎の理解を深めるためにプロジェクトメンバー全員で輪読をおこなった。輪読では各人の負担が均等になるように一人一人ページ数を決めて分担し、それぞれが担当した箇所の理解を深めて、メンバー全員の前で発表した。また輪読で使用した文献[1]内に出てきた Hopfield 型ネットワークの例をメンバーの一人がプログラムで再現し、そのプログラムをメンバー全員に説明した。後期の割り当ては、まずニューラルネットワークのプログラミングを担当するプログラミング班と Kinect のプログラムを担当する Kinect 班、そして参考文献などを探しその中から有用な方法を探し、最終成果物にどのように活かせるかを考える文献班に分かれて活動した。グループを分けた理由としては、同時進行で別々の作業をして最後にそれを組み合わせる方法が一番効率が良いと考えたからである。

(※文責：水岡祐介)

第 3 章 課題解決のプロセス

3.1 課題解決のプロセスの概要

前期の活動内容

前期の活動では始めにニューラルネットワークの基本的な部分の理解を深めることが必要だった。そこで本プロジェクトでは文献[1]を使って学習した。それぞれ担当となる範囲を決め、担当となった人はみんなの前で担当している範囲を解説し、他のメンバーがそれについて意見を交わすといった輪読を行った。輪読は予定していた範囲まで終わらなかったが、それぞれが担当となったページとその内容を記述する。

平井　力　　2p～14p

ニューラルネットワークのもっとも基本的な知識とそこで使われる用語と記号を説明した。また、ニューラルネットワークのモデルの一つである階層型ネットワークについても解説した。

池田　陽彦 14p～20p

Perceptron 型ネットワークの情報処理能力について説明した。主に Minsky と Papert による研究内容が示されており、その解説を行った。また Perceptron 以後の研究の歴史についても触れている。

水岡　祐介 20p～24p

ニューラルネットワークの一つのモデルである相互結合型ネットワークの Hopfield 型ネットワークについて説明した。Hopfield 型ネットワークの特徴や状態変化規則を自己想起的な連想記憶を例に学んだ。

庭田　勘佑 24p～27p

Hopfield 型ネットワークの平衡状態を調べるためにネットワーク全体のエネルギーを、公式を用いて説明した。また、前回の連想記憶の例をもとに実際に計算して説明した。

宮川　聖士 27p～31p

ニューラルネットワークの一つのモデルである相互結合型ネットワークの Boltzmann Machine について、状態変化の公式について解説しネットーワークの基本的な動作について理解を深めた。

笹田　紘平 31p～33p

Boltzmann Machine についての理解を深めるために緩和メカニズムの一般的な定式化の一つである Gibbs Samplar を取り上げ説明した。

佐藤大生 34p～36p
ネットワークの形態について、フィードバック結合を含むか含まないか、そのうちで相互結合型ネットワークや階層型ネットワークの分類についての担当だったが、輪読は回ってこなかった。

中間発表に向けての活動
　輪読が終わった後は中間発表にむけての活動を行った。輪読が予想以上に時間がかかったため、以下の二つの班に分かれてそれぞれが役割分担をし作業を進めていった。
プログラム作成班　笹田　紘平, 水岡　祐介, 宮川　聖士, 佐藤　大生
スライド作成班　　平井　力, 池田　陽彦, 庭田　勘佑
プログラム作成班ではニューラルネットワークの一つのモデルである Hopfield 型ネットワークと Boltzmann Machine についてのプログラムを作成した。スライド作成班ではプログラム作成班が作ったデモプログラムを含み前期のプロジェクト活動で何をしてきたかを具体的にし、スライドを作成した。ポスター作成についてはメンバー全員でそれぞれ役割分担をし、最後に全員で最終確認を行い完成させた。

前期の間の各月の活動内容は以下に記述する

4 月：メンバーと担当教員の顔合わせ及び、今後の作業としてニューラルネットワークについての参考書を使用して輪読を行うことを決定し、各自の担当順番を決め輪読の準備を行った。

- 宮川 聖士
    - 月半ばより輪読の準備を行った。
- 佐藤 大生
    - 輪読の準備を行った。
- 庭田 勘佑
    - 関連書物の捜索を行い、輪読の準備を行った。
- 笹田 紘平
    - 輪読の準備を行った。
- 池田 陽彦
    - 輪読の準備を行った。
- 水岡 祐介
    - テーマの関連書物の捜索、輪読準備を行った。
- 平井 力

- 輪読発表を行い、予習を行った。

5 月：全体での集合時に輪読を行ったが、各自の担当場所の内容に対しての理解が足らず、一人当たりの担当場所の配分を減らし、再配分をすることとなった。また、各自でニューロンのモデルとなる基礎的なプログラムの実装を processing や C 言語を用い行った。

- 宮川 聖士
    - 先月に引き続いて輪読の準備を行った。またニューロンのモデルについての学習を行った。
- 佐藤 大生
    - 輪読の準備を行った。また、ニューロンのモデルについての学習を行った。
- 庭田 勘佑
    - 輪読の準備を行い、ニューロンのモデルについての学習を行った。
- 笹田 紘平
    - 輪読の準備を行い,及び Boltzmann Machine についての基礎的なプログラムの作成を行った。
- 池田 陽彦
    - 輪読の準備を行い、またニューロンのモデルについての学習を行った。
- 水岡 祐介
    - 輪読の準備を行い、ニューロンのモデルについての学習を行った。
- 平井 力
    - 輪読発表を行い、ニューロンのモデルについての学習を行った。

6 月：5 月に引き続き輪読を行い、また中間発表に向けて先輩から昨年の活動についての発表を行っていただき、各自発表に向けて、中間発表のスライドを作成する班、ポスターを作成する班、デモプログラムを作成する班に分かれ、作業を行った。

- 宮川 聖士
    - 輪読発表を行い、また中間発表に向けてのスライドを作成した。
- 佐藤 大生
    - 輪読発表を行い、また中間発表に向けたポスター作成を行った。
- 庭田 勘佑
    - 輪読の準備を行い、また中間発表に向けた準備を行った。
- 笹田 紘平

- 先月に引き続き輪読の準備を行った。また、Hopfield 型ネットワーク及び、Boltzmann Machine のデモプログラムを作成し、中間発表時に見やすいものになるよう調整を行った。

- 池田 陽彦
    - 輪読準備及び中間発表用のスライド作成を行った。
- 水岡 祐介
    - 輪読準備及びスライドの作成を行った。
- 平井 力
    - 中間発表のポスター作成を行った。

７月：中間発表会へ向け発表練習及びデモプログラム、ポスターの作成を行い、発表練習を行い、注意された点の修正を行った。中間発表会では、2 グループに分かれ、発表を行った。中間発表会後は、夏季休暇中に各自が読み込む参考書の配分決定、後期に行うことの決定及び、中間提出物の作成を行った。

- 宮川 聖士
    - 中間発表用のスライド作成を行った。中間発表では、主に研究の目的、ニューラルネットワークについて説明を行った。
- 佐藤 大生
    - 中間発表用のポスター作成及び改善を行った。中間発表では、主に Boltzmann Machine と Hopfield 型ネットワークとは何なのかについて説明した。
- 庭田 勘佑
    - 中間発表用のポスター及びスライドの修正を行った。中間発表では、主に研究の目的、ニューラルネットワークの概要及び実際の応用例について説明を行った。
- 笹田 紘平
    - 月の前半に輪読の発表及び中間発表に向けてのプログラムの調整を行った。中間発表では、主にデモプログラムに関する説明を行った。
- 池田 陽彦
    - 中間発表のスライドとポスターの作成と改善を行った。中間発表では、Hopfield 型ネットワーク、Boltzmann Machine 及び連想記憶についての説明を行った。
- 水岡 祐介
    - 中間発表用のスライドの修正を行った。中間発表では、主に、ニューラルネットワークの実際の利用例及び連想記憶についての説明を行った。
- 平井 力
    - 輪読の準備及びスライド、ポスターの作成を行った。中間発表では、主にでもプログラムの内容説明と実際に実験を行った。

夏季休暇中（８月及び９月）

前期の輪読を行ってきた部分の復習、ネットワークメカニズムの分類及び応用、ニューラルネットワークの学習メカニズムについての学習を行った。また、認知過程の先行研究についても各自学習を行った。さらに、期末成果物作成のために先行研究を各自で調査した。

後期の活動内容

本研究の目的はニューラルネットワークを使って読唇術を実現させることである。そのためには先行研究や参考本などを調べる必要があった。そして、読唇術を実現するためには Kinect を使用する必要があったため Kinect についても調べる必要があった。また、ニューラルネットワークのプログラムも作成する必要があった。そこで本プロジェクトの後期の活動では、先行研究や参考本を調べる資料班とニューラルネットワークと Kinect のプログラム作成をするプログラム班に分かれて研究を進めた。また、作成した A と B のシステムで比較実験を行った。

プログラム班　平井　力　池田　陽彦　笹田　紘平　佐藤大生

資料班　　水岡　祐介　庭田　勘佑　宮川　聖士

プログラム班の活動

ニューラルネットワークのプログラム作成

本研究で使用したニューラルネットワークは層状ネットワークで三層のものを作成し使用した。また本研究では比較実験を行ったため、2 つ（A と B）のニューラルネットワークを作成した。

Kinect の設定、プログラムの作成

Kinect の設定とプログラムの作成では文献[6]を参考にして、口の周りの特徴点の座標を取得しニューラルネットワークのプログラムに渡すところまでを作成した。

資料班の活動

先行研究を集めるためにネットやライブラリーを使い探した。先行研究に書いてあった読唇術の方法で、口の縦幅と横幅の距離を測り面積を計算し、面積の違いで言葉を判別するといった方法を参考にすることで研究を進めた。

また、プログラム班の進行が少し遅れていたためデータを整理する等の作業を協力した。

実験の概要

本研究では二つ（A と B）のニューラルネットワークを作りこれらの正答率を比較してどちらが読唇術を実現させるのに優れているのかを調べた。 A は母音を 1 音ずつ出力し、その結果からリスト内の単語をマッチングする方式で B はあらかじめ設定しておいたリスト内の言葉のどれかを直接出力する方式である。

最終発表に向けての活動

最終発表に向けての活動では実験が予定していた期間より少し時間がかかってしまったため、手の空いてる人がスライドとポスターの作成を行った。

９月:夏季休暇中に調査してきたものを基に何についてのシミュレータを作成するかについて話し合い、引き続き、そのテーマの先行研究を調査した。

後期の各月の活動は以下に記述する

- 宮川 聖士
    - 期末成果物作成に向けた先行調査を行った。
- 佐藤 大生
    - 期末成果物作成に向けた先行調査を行った。
- 庭田 勘佑
    - 期末成果物作成に向けた先行調査を行った。
- 笹田 紘平
    - 期末成果物のテーマ調査及び参考文献の調査を行った。その後バックプロパゲーション法についての調査を行った。
- 池田 陽彦
    - 期末成果物のテーマ決定のための調査を行った。
- 水岡 祐介
    - 期末成果物作成に向けた先行調査を行った。
- 平井 力
    - 期末成果物作成に向けた先行調査を行った。

10 月：テーマとして読唇術の研究を行うことを決定し、ニューラルネットワークを用い学習を行わせるプログラムを作成する班、Kinect を利用しデータを収集するプログラムを作成する班、データや実験方法を決定する班の 3 班に分かれて作業を始めた。途中昨年度のプロジェクトメンバーの方々にお越しいただきご指導を賜った。

- 宮川 聖士
    - Kinect を利用しデータを収集するプログラムの作成を行った
- 佐藤 大生
    - ネットワークに学習させ、識音に利用するデータについて話し合いを行った。
- 庭田 勘佑
    - ネットワークに学習させ、識音に利用するデータについて話し合った。
- 笹田 紘平
    - バックプロパゲーション法についての学習を行い、その方法を用いて一音ごとにどの母音が発せられたか学習するニューラルネットワークプログラムの作成及び学習させるデータの収集、顔認識における先行研究の調査を行った。
- 池田 陽彦
    - Kinect を利用しデータを収集するプログラムの作成を行った。
- 水岡 祐介
    - ネットワークに学習させ、識音に利用するデータのについての話し合いを行った。
- 平井 力
    - 全体の進度の調整および実験の参加者の募集を行った。

11 月：10 月に引き続いて、3 班に分かれ、実験を行いつつ、制作物を作成し、成果発表会に向けた準備を行った。

- 宮川 聖士
    - 実験用のプログラムの作成および期末発表用のスライド作成を行った。
- 佐藤 大生
    - 実験用のプログラムの作成を行った。その後再実験に向けての追加する学習データ収集を行った。
- 庭田 勘佑
    - Kinect を用い特徴点抽出を行うプログラムの作成を行った。
- 笹田 紘平
    - ニューラルネットワークプログラムの作成及び学習させるデータの収集、及びネットワークの調整を行った。

- 池田 陽彦
    - 実験用のプログラム作成及びスライドの作成を行った。
- 水岡 祐介
    - 実験用のプログラム作成を行った。
- 平井 力
    - スライドおよびポスターの作成を行った。

12 月：成果発表会に向け、ポスターやスライドの準備および並行して実験を行った。本番では、二組に分かれ実験に利用したシミュレータを使用し発表を行い、報告書の執筆および再実験を行った。

- 宮川 聖士
    - 中間発表用のスライド作成を行った。期末発表においては、実験の概要及び結果とその考察について説明した。
- 佐藤 大生
    - 実験用のプログラム作成を行った。期末発表においては、デモプログラムを使い、実際に読唇術の実験を行った。
- 庭田 勘佑
    - Kinect を用い特徴点抽出を行うプログラムを作成し、実際に実験を行った。期末発表においては、層状ネットワークとは何か、実験の概要、結果および考察を説明した。
- 笹田 紘平
    - ネットワークの調整を行った。期末発表においては、主に研究のテーマである読唇術についての説明を行った。その後、期末報告書を執筆した。
- 池田 陽彦
    - 実験用のプログラム作成及びスライドの作成を行った。期末発表においては、主に研　究の概要、読唇術、及びニューラルネットワークについての説明を行った。
- 水岡 祐介
    - 実験用のプログラム作成を行った。期末発表においては、層状ネットワークの説明および実験に用いた条件の説明を行った。
- 平井 力
    - スライド、ポスターの作成を行った。期末発表においては、デモプログラムを使用した読唇術の実験を行った。また、期末報告書の章立てを行った。

１月:期末提出物の点検を行い完成させた。

（※文責：笹田紘平）

3.2　前期の主な活動

前期ではニューラルネットワークの基礎の理解を深めるためにプロジェクトメンバー全員で輪読をおこなった。輪読では各人の負担が均等になるように一人一人ページ数を決めて分担し、それぞれが担当した箇所の理解を深めて、メンバー全員の前で発表した。また輪読で使用した文献[1]内に出てきた Hopfield 型ネットワークの例をメンバーの一人がプログラムで再現し、そのプログラムをメンバー全員に説明した。後期の割り当ては、まずニューラルネットワークのプログラミングを担当するプログラミング班と Kinect のプログラミングを担当する Kinect 班、そして参考文献などを探しその中から有用な方法を探し、最終成果物にどのように活かせるかを考える文献班に分かれて活動した。グループを分けた理由としては、各々の得意な分野を分担しておこなえば作業効率が上がると考えたからである。

(※文責：水岡祐介)

3.3　中間発表に向けての活動

輪読が終わった後は中間発表にむけての活動を行った。輪読が予想以上に時間がかかったため、以下の二つの班に分かれてそれぞれが役割分担し、作業を進めていった。

プログラム作成班　笹田　紘平, 水岡　祐介, 宮川　聖士, 佐藤　大生

スライド作成班　　平井　力, 池田　陽彦, 庭田　勘佑

プログラム作成班ではニューラルネットワークの一つのモデルである Hopfield 型ネットワークと Boltzmann machine についてのプログラムを作成した。スライド作成班ではプログラム作成班が作ったデモプログラムを含み前期のプロジェクト活動で何をしてきたかを具体的にし作成した。ポスター作成についてはメンバー全員でそれぞれ役割分担をし、最後に全員で最終確認を行い完成させた。

3.4　最終発表に向けての活動

デモプログラムの作成と実験結果の解析が終わり次第、最終発表に向けてポスターとスライドの作成を行うつもりだったが、実験に少し時間がかかってしまったため、各々手が空いた人からポスターとスライドの作成を行った。スライドには我々が行ってきた活動をわかりやすく説明するように工夫し作成した。ポスターは研究内容を詳しく書き実験の目的や実験方法、実験結果などを具体的に書いた。

笹田　紘平　実験および実験結果の解析　ポスターの作成

水岡　祐介　スライドの作成

宮川　聖士　ポスターの作成及びスライドの作成

佐藤　大生　実験及び実験結果の解析

平井　力　　スライドの作成及び実験結果の解析

池田　陽彦　実験及び実験結果の解析

庭田　勘佑　ポスターの作成及びスライドの作成

それぞれが役割分担をし時間がない中最終発表に間に合わせることができた。

（※文責：池田陽彦）

3.5　デモンストレーション

後期の最終発表では実際に読唇術のシステムを紹介し、体験してもらうためにデモンストレーションを行った。最終発表中に一人募り、用意した Kinect から 80cm 離れたイスに座ってもらった。単語のリストは発表用のスライドのものを見せてその中から自由に選んで発話してもらい、実験と同じ方法で行えるようにした。Kinect を使用している PC の画面をスクリーンに映し出して被験者がスクリーンを見ることで位置を一定にするための基準点と、発話のタイミングを調節するためのメーターを見ることができるようにした。

また、読唇術の結果をわかりやすく表示するためにリストの単語の画像をそれぞれ作成した。図はその一部である。ニューラルネットワークの結果から、単語の画像を画面に表示させた。デモンストレーションの結果としては、前半後半 3 回ずつの発表で前半は 3 回成功、後半は 1 回の成功であった。

(※文責：佐藤大生)

ありがとう　　あさごはん

図 3　結果の表示のためのリストの単語

第 4 章　課題解決へのプロセスの詳細

本プロジェクトでは、活動の目標としてニューラルネットワークの数理についての理解を深め、人間の認知過程に関する研究をするためのニューラルネットワークを用いた数値実験を行うことを目標に活動してきた。この章ではニューラルネットワークについて理解を深めるために学習した内容の一部とニューラルネットワークで読唇術を実現する方法を説明する。

4.1　ニューラルネットワーク

人間の脳の神経細胞はニューロンと呼ばれており、多数のニューロンが他のニューロンと結合しネットワークを形成する。ユニットの結合によって、ネットワークは様々な形態をとる。大きく分けてフィードバック結合を含まない階層型ネットワークとフィードバック結合を含む相互結合型ネットワークに分けることが出来る。

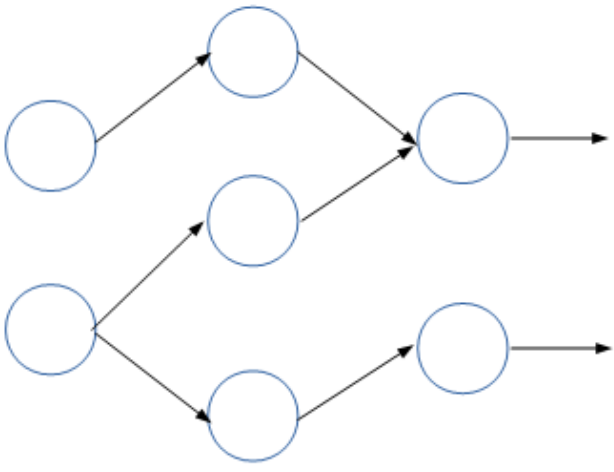

図 4.1:階層型ネットワークの簡易的な例

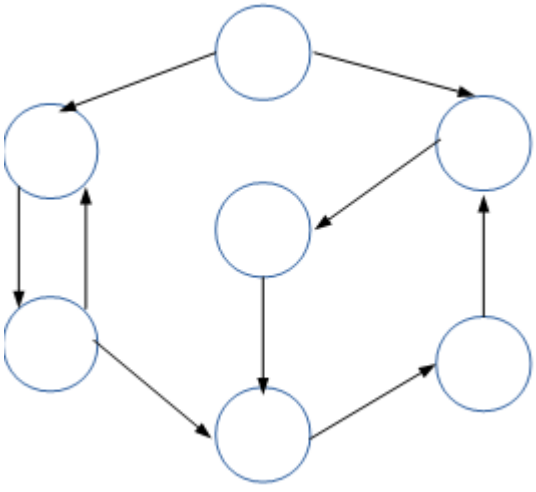

図 4.2:相互結合型ネットワークの簡易的な例

（※文責：池田陽彦）

4.1.1 階層型ネットワーク

パーセプトロンとは 1985 年に Rosenblatt の提案した学習する翻訳機械で、階層型ネットワークの基本となっている。パーセプトロンの各層は適当な数のユニットからなっていて、層内の結合はなく、層間の結合は入力層から出力層にかけて一方向に結合している。

入力層を除いたそれぞれのユニットは前の層のユニットからの入力の重み付き総和を計算しそれに適当な関数 f をかけたものを出力する。i^k は第 k 層の i 番目のユニットの入力の総和として、o_i^kは第 k 層の i 番目のユニットの出力として、$w^{k-1}{}_{ji}^{k}$を第 k-1 層の j 番目のユニットから第 k 層の i 番目のユニットへの結合の強さ、$\theta^k{}_i$を第 k 層の i 番目のユニットのしきい値とすると

$$i^k{}_i = \sum_i w^{k-1}{}_{ji}^{k} o^{k-1}{}_i - \theta^k{}_i \tag{4.1}$$

$$o_i^k = f\,(i\,{}_i^k) \tag{4.2}$$

と表される。ユニットの入出力関数 f としては、しきい関数、区分線形関数、ロジスティック関数、恒等関数がよく使われる。

（※文責：宮川聖士）

4.1.2 相互結合型ネットワーク

相互結合型ネットワークは、ユニットが相互に結合しているネットワークである。原則として、ある 1 つのユニットは他のすべてのユニットと結合を持つ。また、階層型ネットワークとの大きな違いとして、階層型ネットワークは入力層から出力層まで一方向への入出力によって状態変化するのに対し、相互結合型ネットワークはユニット間をひたすらループし状態変化が終わることがない点である。

（※文責：宮川聖士）

4.2　Hopfield 型ネットワーク

米国の物理学者、J.Hopfield が 1982 年に提唱したニューラルネットワークで、最適化問題に応用出来る事で有名になった。Hopfield 型ネットワークは相互結合型ネットワークで非同期的である。ここで、u_jをユニットjの状態、s_iを外部からの入力、θ_iしきい値とすると、状態変化規則の式は次のようになる。

$$\begin{aligned} &\sum_j w_{ji}u_j + s_i - \theta_i > 0 \text{ のとき} u_i = 1 \\ &\sum_j w_{ji}u_j + s_i - \theta_{\mathrm{i}} < 0 \text{ のとき} u_j = 0 \\ &\sum_j w_{ij}u_j + s_i - \theta_i = 0 \text{ のとき変化しない} \end{aligned} \tag{4.3}$$

（※文責：佐藤大生）

4.2.1 ネットワークのエネルギー

4.2.1 ネットワークのエネルギー

この平衡状態はユニット間の結合状態によって決まり、一般に複数ある平衡状態のどの平衡状態に陥るかは初期状態によって決まる。エネルギーは以下の式で定まる

$$E = -\frac{1}{2}\sum_j \sum_i w_{ij}u_j u_i - \sum_i (s_i - \theta_i)u_i \tag{4.4}$$

ここで、w_{ij}をユニット i からユニット j への結合の強さ、u_i、u_jをユニットそれぞれ i 、j の状態、s_iを外部からの入力、θ_iしきい値とする。あるユニットu_iが状態変化規則に従って状態変化するときのエネルギー関数の変化量を(4.4)を用いて求める。

[1]u_iが 1 から 0 に変化した際のエネルギー関数の変化量は、

$$\Delta E = \sum_i w_{ij}u_i + s_i - \theta_i \tag{4.5}$$

となる。u_iが 1 から 0 に変化した場合の状態変化規則の式は、

$$\sum_j w_{ij}u_i + s_i - \theta_i < 0 \tag{4.6}$$

であるので、(4.5)の式の右辺と(4.6)の式の左辺は等しい。よって$\Delta E < 0$となり負の値をとるので、ネットワークのエネルギーは減少した。

[2]u_iが 0 から 1 に変化した場合のエネルギー関数の変化量は、

$$\Delta E = \sum_i w_{ij}u_i - s_i + \theta_i \tag{4.7}$$

となる。u_iが 0 から 1 に変化した場合の状態変化規則の式は、

$$\sum_i w_{ij}u_i + s_i - \theta_i > 0 \tag{4.8}$$

である。よって$-\Delta E < 0$となり負の値となりネットワークのエネルギーは減少した。

[1][2]よりu_iが 1 から 0 へ変化した際も、0 から 1 へ変化した際も状態変化規則にしたがってそれぞれのユニットが非同期的に状態変化を繰り返していき、その度にエネルギーは減少しネットワークの状態が変化していく。状態変化が起こるたびにネットワークのエネルギーは減少するためエネルギーが増加する際、状態変化は起こらない。こうして状態変化を繰り返していくと、ある状態で変化しなくなる。それを平衡状態とする。この時ネットワークのエネルギーは極小値になる。

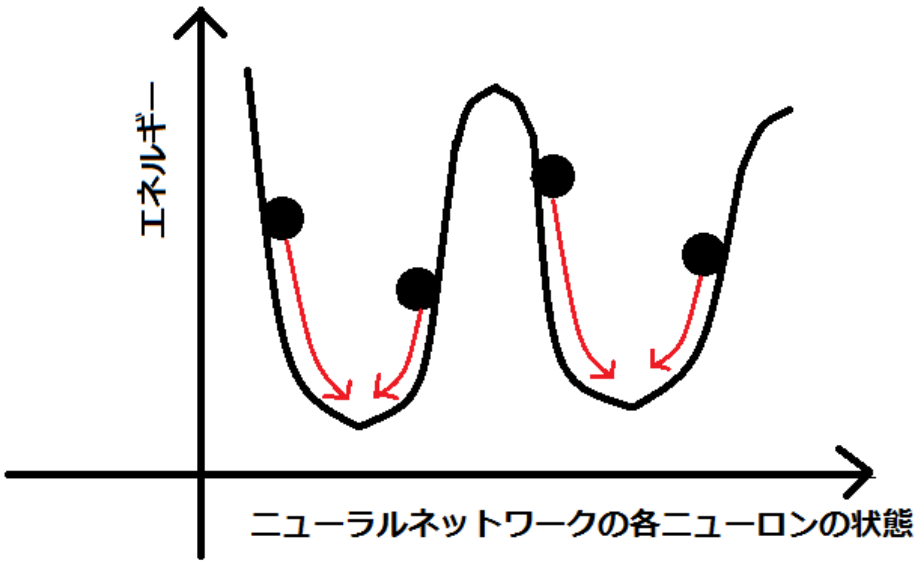

図 4.3 ニューラルネットワークのエネルギー

（※文責：庭田勘佑）

4.3 Boltzmann Machine

Boltzmann Machine は Hopfield 型ネットワークのメカニズムの動作を確率的にしたものである。各ユニットの新しい状態は

$$p = \frac{1}{1+\exp(\frac{-i}{T})} \tag{4.5}$$

の確率で 1 になる。ここで T はネットワークの温度を示し、この値が上昇する程、状態が変化する確立が 1/2 に収束する。このように設定をすることで、Hopfield 型ネットワークでは平衡状態になった後に状態変化が起きることがないが、Boltzmann Machine では次の平衡状態に向かって状態変化が起きるという大きな違いが現れる。

（※文責：宮川聖士）

4.4　読唇術

文献[4]では「あ」、「い」、「う」、「え」、「お」、「ん」と発音した時のそれぞれの口(図 4.3)の縦幅と横幅の距離のデータを取り、それらから面積を求めてその違いによって単語を識別しているが、今回本プロジェクトでは発音する言葉は同じだが、口の周りの特徴点の座標を抽出し、口の縦幅と横幅、鼻の頂点からあごの先までの距離、目頭間の距離を測り、面積は求めず距離だけでの単語を識別させることを課題とした。以下でニューラルネットワークによる読唇術の方法について説明する。

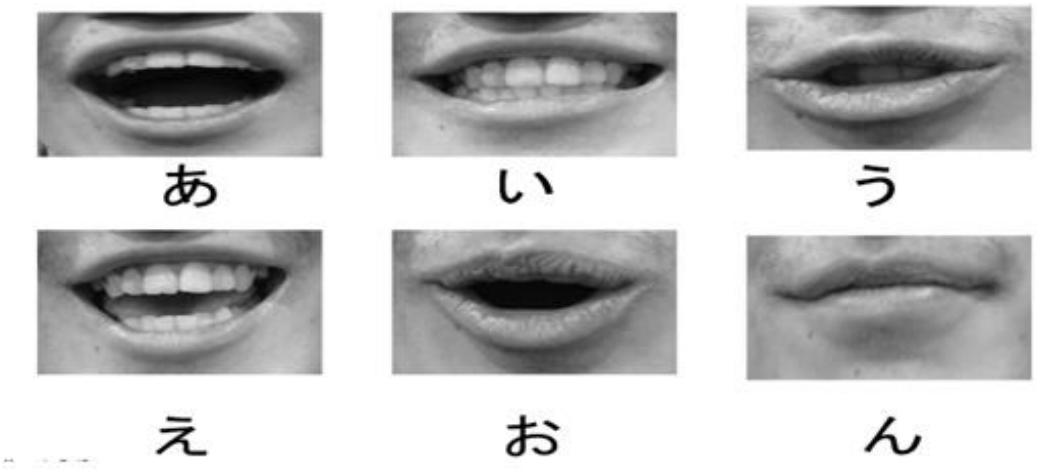

図 4.4:発音するときの口の形

今回、使用するニューラルネットワークは 2 種類あり、それぞれパターン A、パターン B と名前を付けた(以降では A、B と表記する)。A は入力層のユニットが 3 個、中間層のユニットが 50 個、出力層のユニットが 6 個(図 4.4)で、B は入力層のユニットが 15 個、中間層のユニットが 50 個、出力層のユニットが 21 個(図 4.5)の 3 層のバックプロパゲ―ションモデルを採用した。このような 3 層の階層型ネットワークのモデルは中間層に必要なだけのユニットを用いて各ユニット間の結合の重みを適切な値にすることによって任意の非線形の連続関数をいくらでも精密に近似できることが知られている。よって、このモデルを用いることで読唇術を高い精度で行うことができると考えられる。

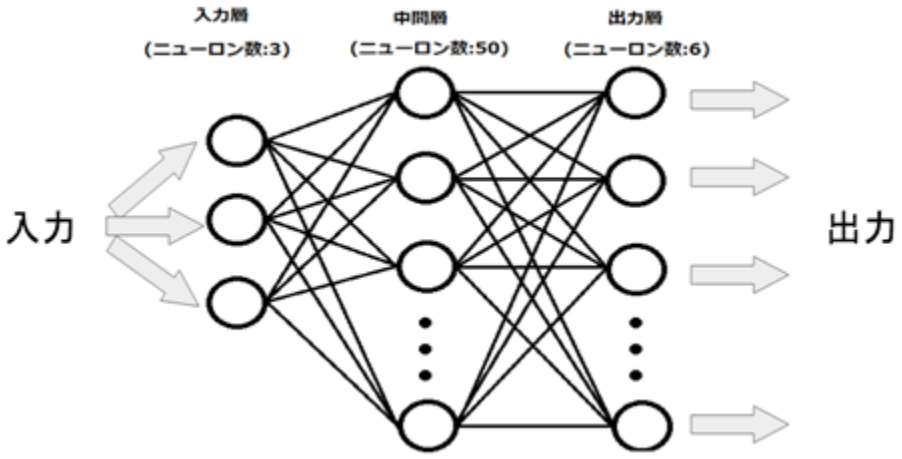

図 4.5:A のニューラルネット

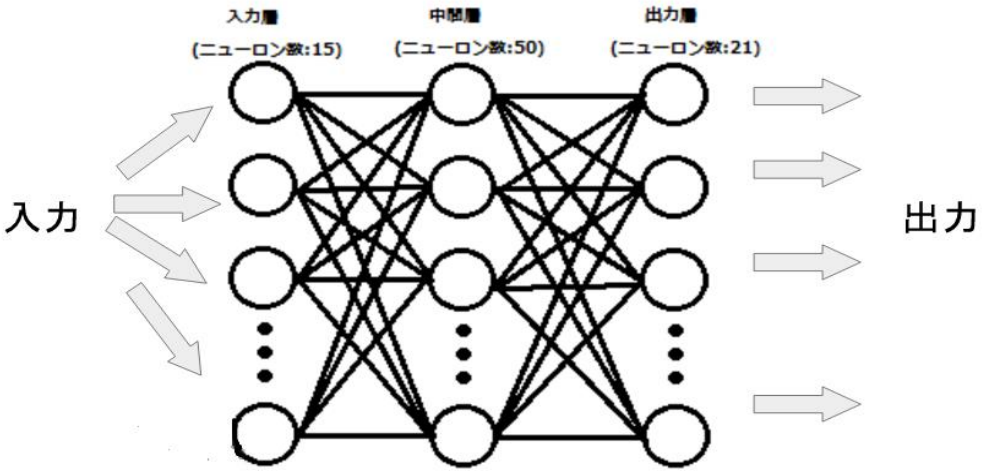

図 4.6:B のニューラルネットワーク

次に、口の周りの特徴点のデータの取得方法について説明する。被験者 1 人 1 人に母音を発音してもらっている時に口の周りの特徴点の座標を抽出して x1 と x2、x3 と x4、x5 と x6、x7 と x8 間のそれぞれの距離を取得する(図 4.6)。この時の特徴点は Kinect を用いて取得した。また、被験者の顔と Kinect のカメラとの距離によって生じる口の大きさの違いを修正するためにどんな発音をしたときでも変化しないと考えられる目頭間の距離(図の x1 と x2 間の距離)で割る。付録にその時のプログラムのソースコードを示す。

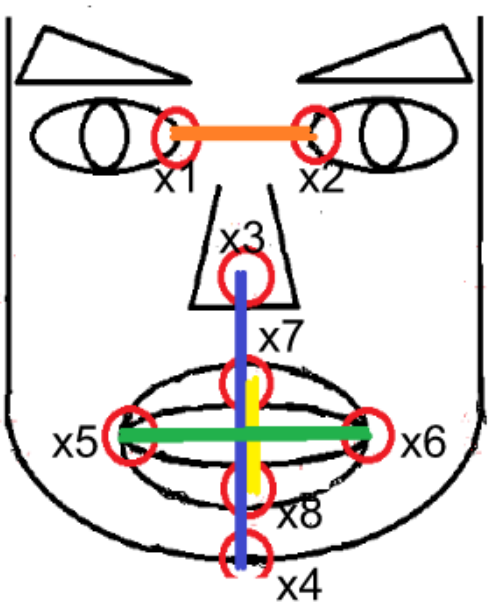

図.4.7 使用した特徴点

第 5 章　実験について

5.1　実験の目的

文献[5]では今の技術で口の動きだけを見て母音と子音を正確に識別し読唇術を行うことは難しいと述べられていた。よって、本プロジェクトでは公立はこだて未来大学の学生が普段から使っているであろう 5 文字の単語 21 種類のリストを作り、その中の単語を先述の A、B のニューラルネットワークで識別させた。A は「あ」、「い」、「う」、「え」、「お」、「ん」と発音したときの口の周りの特徴点を学習させて、単語を発音したときの特徴点を入力して単語の母音 1 つ 1 つを判別してその結果からリスト内の単語とマッチングして識別する方法で、B はリスト内の単語を発音したときの特徴点を学習させてリスト内の単語の何を発音したのかを識別する方法である。これら 2 種類のニューラルネットワークで実験を行ってどちらのパターンが読唇術の成功率が高いのかを調べる。

(※文責：宮川聖士)

5.2　実験で使用した単語のリストについて

本実験では使用する単語は母音が一致しない 5 文字の単語 21 種類を使用した。以下に今回使用した単語と母音の並びをそれぞれ記述した。なお、長音符の母音は前の単語と同じものとして扱う

・あさひかわ(aaiaa)　・あさごはん(aaoan)　・ありがとう(aiaou)
・えいがかん(eiaan)　・おかあさん(oaaan)　・おしょうがつ(oouau)
・おむらいす(ouaiu)　・くりすます(uiuau)　・こんにちは(oniia)
・こんばんは(onana)　・さようなら(aouaa)　・しいげんご(iieno)
・しんぶんし(inuni)　・すみません(uiaen)　・どらえもん(oaeon)
・ひるやすみ(iuaui)　・ふゆやすみ(uiaui)　・ぷろじぇくと(uoeuo)
・ぷろぐらむ(uouau)　・みらいだい(iaiai)　・ゆきだるま(uiaua)

(※文責：平井力)

5.3 実験装置について

読唇術を実現するにあたって、顔の特徴点を取るために Kinect for windows v2 を使用した。Kinect について学習するために[6]を参考にした。HD face tracking を使用し、特徴点は HighDeteilFacePoints から 8 つの点を利用した。また、実験時に被験者の位置が一定であるようにするために、画面の中央に黄色い点を基準点として設置した。被験者の鼻の頭の点を基準点に合わせることで一定の位置で実験を行うことができるようにした。

実験で被験者の発話時の音を一音ずつに切り分ける方法としては、自動で切り分けることが難しかったため、キーボードでキーを押して特徴点の座標のデータを記録するようにした。そして、画面左にメーターを設けて、そのメーターに従って発話してもらうことで、被験者がゆっくりとした発話ができるようにした。

ニューラルネットワークとの連携は、Kinect で取得した特徴点座標のデータを txt 形式で保存し、ニューラルネットワークに入力した。

(※文責：佐藤大生)

5.4 実験の手順

実験は以下の手順に従って行った

I : ニューラルネットワークの学習を行うため被験者にリスト内の単語を発話してもらい、その単語に関する顔の座標データを取得し、そのデータを利用しニューラルネットワークに学習させる。今回の実験ではそれぞれのネットワークには 20 人分の学習用データを用いた。データを取得するための環境については下の図 5.1 に記述する。

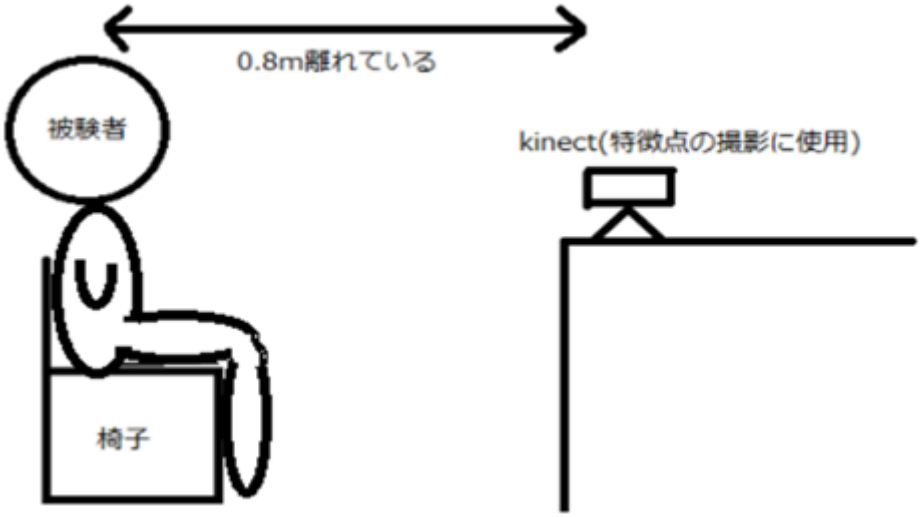

図 5.1:実験を行う環境

II：実験で使用するリスト内の単語のデータを被験者から取得する。この時にIとIIの被験者は同一人物だと学習用データと実験で使用するデータが完全に一致してしまう可能性があるため、異なる人間からデータを収集するようにした。また、データの収集環境はIと同様の環境で行った。

III：リスト内の単語からランダムに 1 つ選出し、A,B のそれぞれのニューラルネットワークに入力をし、それぞれのニューラルネットワークが出した結果と実際に入力した単語が一致しているかを調べ、一致している場合は 1、不一致の場合は 0 と記録する。それぞれのニューラルネットワークは 2 つ以上の単語が予測される可能性が存在するため、複数予測された場合に、それらの中に入力した単語データと一致したものが存在する場合は一致した場合と同じく 1 と記録する。

IV：IIIの手順を 100 回繰り返すことで、それぞれのニューラルネットワークの正答率求め、それらを比較し A と B のどちらのニューラルネットワークのほうがより優れているかを検証する。この比較を行うにあたってそれぞれの正答率に有意な差が存在するかどうかを確認する為にカイ二乗検定を行うことで二つの結果の優位性を調べる。

上記の手順で 2 つのパターンで差が生じるかを調査した。

(※文責：宮川聖士)

第 6 章　成果と結果

6.1　前期の活動結果

前期では、ニューラルネットワークのメカニズムの学習を行った。結果として、ニューラルネットワークの基本的な仕組みを理解した。また、ニューラルネットワークはデータの流れが 1 方向で階層構造をもつ階層型ネットワークとニューロン同士が相互に対等に結合している相互結合型ネットワークに分類されることがわかった。階層型ネットワークの代表的な例としてパーセプトロンが挙げられ、相互結合型ネットワークの代表的な例として状態変化が決定論的である Hopfield 型ネットワークや状態変化が確率的である Boltzmann Machine が挙げられる。私たちは本プロジェクトの前期の成果物として Hopfield 型ネットワークと Boltzmann Machine に関するプログラムを作成し、この 2 つのモデルの特徴や相違点を明確にすることができた。(図 6.1,6.2 参照)

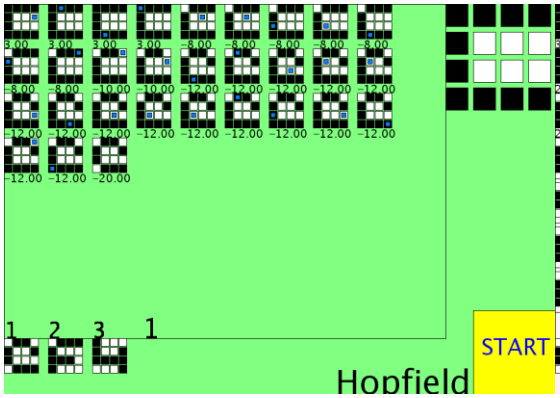

図 6.1 Hopfield 型ネットワーク

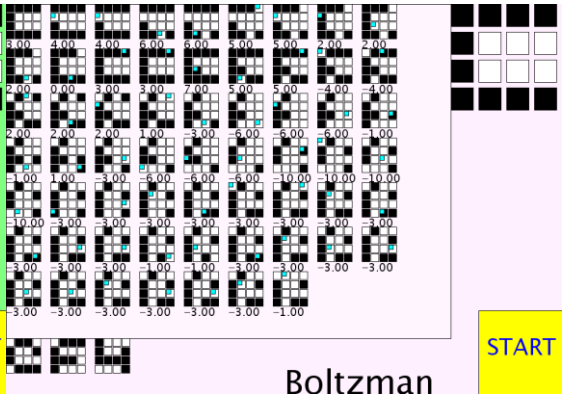

図 6.2　Boltzmann Machine

（※文責：庭田勘佑）

6.2 後期の活動結果

後期では、前期で学んだニューラルネットワークを用いて人間の認知過程について調査した。本プロジェクトにおいては、発話時の鼻や口の周りなどの特徴点をニューラルネットワークに入力することで読唇術が実現できるかどうか実験を行った。結果として、口の周りや鼻や顎のデータを入力し、学習したデータと一音ずつマッチングさせる実験 A の正答率は 21%となり、リスト内に含まれる単語を直接出力させる実験 B の正答率は 45%となった。これによりリスト内の単語を直接出力させる実験 B のほうが読唇術を実現するのにより適しているということがわかった。

（※文責：庭田勘佑）

6.3　成果の評価

6.3.1　前期の活動に対しての評価

前期では、ニューラルネットワークに関する参考書[1]を用いてグループメンバーで分担して輪読を行った。この作業によりニューラルネットワークにおける基本的な構造や仕組みを学習した。また Hopfield 型ネットワークと Boltzmann Machine に関するプログラムを作成した。このプログラムを実装することでこれらのネットワークの仕組みの理解を深めた。輪読の進捗状況としては進度が遅く前期の目標としていたネットワークの学習手法まで進むことができなかったが、時間をかけて学んだことで基礎知識を固めることができた。中間発表では、聴講者にニューラルネットワークの基本的な仕組みを理解してもらうためにスライドと私たちが作成したプログラムの説明を行った。デモンストレーションを行った結果として Hopfield 型ネットワークと Boltzmann Machine の違いを理解することができたという回答が得られたが、スライドに関しては専門的な用語や数式により内容が理解できなかったという回答が得られた。

（※文責：庭田勘佑）

6.3.2　後期の活動に対しての評価

後期では、ニューラルネットワークの分類法の一つである層状ネットワークを用いて読唇術を実現させるための装置の作成をグループメンバーで分担し行った。実験結果として実験 A の正答率では約 2 割、実験 B の正答率では約 5 割という結果になった。実際に人間が読唇術を行った場合の正答率は約 4 割なので実験 B においては人間が実際に行うよりも正答率の高い読唇術を実現することができた。後期に入り読唇術に関する研究を行うことが決定してから先行研究がなかなか見つからずスケジュール通りに進行できなかった。しかし先行研究が見つかってからは活動時間を拡大し各々分担することで成果物作成まで至ることができた。最終発表では、聴講者にニューラルネットワークを用いた読唇術を実現するためのプログラムを作成しデモンストレーションを行った。発表後の評価としてスライドに関しては２つの行った実験においての説明が少しわかりづらい、ニューラルネットワークについての基本的な説明がないことですべてにおいて明確な理解を得ることができなかったと改善が必要な評価が得られたのに対し、専門用語の説明や各々のスライドにおける補足説明があってよかったなどの評価が得られ前期の改善点は修正することができた。デモンストレーションを行った際の評価としては、実験を行うことでスライドでの不明点も明確になり理解が深まるので良いや実験の成果を見ることができてよかったなどの評価が得られた。

（※文責：庭田勘佑）

6.3.3　総評

１年間のプロジェクト活動を行ってきて前期にグループメンバー分担してニューラルネットワークに関する参考書の輪読を行ったが、目標としていたニューラルネットワークの学習手法まで進めることができなかった。しかし、輪読で学んだ範囲が短かった分、基礎知識を固めることができ後期への方針を立てることできた。後期に入ってからは認知過程に関するテーマとして読唇術を実現しようと作業を進めようとしたが利用できそうな先行研究があまり見つからずスケジュール通りに進めることができなかった。先行研究が見つかったあとは講義時間外に実験やデータ分析などメンバーで分担し行うことができたため本来のスケジュールに追いつくことができた。実験においては人間が読唇術を行った場合よりも良い結果を得られたが学習用のデータを増やすことで精度が向上し今回の正答率より高い結果が得られると考える。また今回の活動において母音に着目して実験を行ってきたが子音を識別できるようなればより実用的になると考える。

成果発表会においては、中間発表会の改善となる専門的な用語や聴講者が容易に理解しづらいスライドの説明を後期の最終発表会では補足説明やスライドに図や画像を取り入れ工夫を施すことで改善でき評価としてもわかりやすいという回答が得られた。しかし、発表時において聴講者に目を向けずスライドばかりを見ているのを多く指摘されていたため発表技術としてはあまり良い評価を得られなかった。人のグループメンバーがそれぞれグループに分かれ作業を進めていたが、それぞれのグループで難題が出てきた際も全員が団結して行うことができたため満足のいく成果物を作成することができた。

(※文責：庭田勘佑)

第 7 章 今後の課題

7.1　今後の課題

本プロジェクトでは、ニューラルネットワークを用いた読唇術の実現を行った。今回は識別する単語が 21 種類であったり、母音だけの識別であったりしたため前章で述べたような結果となった。識別する単語の数を少量に絞ることで実際に人間が行うよりも正答率の高い結果が得られたが、すべての単語を今回作成したもので識別させようとするとやはり正答率はかなり下がってしまうと考えられる。そのため、今よりも多くの単語を識別できるようにニューラルネットワークの学習に用いるデータの量を増やす必要がある。さらに実験では被験者すべてが男子学生であったため、年齢や性別によっては、異なる結果が得られていた可能性も考えられるので、検証が必要である。また、今回は母音だけ識別をしているので、さらに子音も識別しようとするとより工夫が必要になってくる。他にも正答率を上げるために特徴点の数や被験者の人数を増やすことも必要になり、これらを今後の課題とする。

（※文責：宮川聖士）

参考文献

[1] 麻生秀樹.ニューラルネットワーク情報処理—コネクショニズム入門、あるいは柔らかな記号に向けて—.産業図書 1998.

[2] 甘利俊一.神経回路網モデルとコネクショニズム.東京大学出版会 1989.

[3] 中野馨.ニューロコンピュータの基礎.コロナ社 1990.

[4] 守啓祐,園田頼信.発話時の口唇形状の認識実験.電子情報通信学会総合大会 1995.

[5] 小林宏,原文雄.ニューラルネットワークによる人の基本表情認識.計測自動制御学会論文集 1993.

[6] 中村薫.KINECT for Windows SDK プログラミング Kinect for Windows v2 センサー対応版.秀和システム 2015.

付録

1 新規習得技術

中間発表時に使用したデモのソースコード

```
1.hopfield network
void setup() {
  size(1000, 700);
  background(128, 255, 128);
}
int pat[][]= {
  {0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1},
  {0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0},
  {1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1}
};//記憶させたパターン

int now[]= {
  1, 1, 1, 1,
  1, 1, 1, 1,
  1, 1, 1, 1,
  1, 1, 1, 1
};//ネットワークの状態
int col[]= {
  1, 1, 1, 1,
  1, 1, 1, 1,
  1, 1, 1, 1,
  1, 1, 1, 1
};//初期状態を決めるための配列
int flag=0;
int wij[][]=new int[16][16];
int weight(int i, int j) {
```

```
  int y=0;
  for (int a=0;a<3;a++) {
          y=y+((2*pat[a][i])-1)*(2*pat[a][j]-1);
  }
  if (i==j) return 0;
  else return y;
}//ユニット間の重みづけをする関数
int sikaku() {

  for (int i=0;i<16;i++) {
          fill((1-col[i])*255);
          rect(i%4*50+800, i/4*50, 40, 40);
  }
  fill(255, 255, 0);
  rect(850, 550, 150, 150);
  if (flag==0) {
          fill(0, 0, 255);
          textSize(40);
          text("START", 865, 630);
  }
  return 0;
}//ネットワークの初期状態を決めるためのボタンを作る関数
int table() {
  for (int c=0;c<16;c++) {
          for (int d=0;d<16;d++) {
          wij[c][d]=weight(c, d);
          }
  }
  return 0;
}//ネットワークの間の重みづけの表を作成する関数




float ene() {
  table();
  float e=0.0;
  for (int q=0;q<16;q++) {
          for (int w=0;w<16;w++) {
          e=e-((wij[q][w]*now[q]*now[w])/2.0);
          }
  }
  return e;
}//ネットワークのエネルギーを計算する関数
int chk(int k) {
  table();
```

```
  int b=0;
  for (int a=0;a<16;a++) {
          if (a!=k) {
          b=b+wij[k][a]*now[a];
          }
  }
  if (b>0)return 1;
  else if (b<0)return 0;
  else return -1;
}//Hopfield network の状態変化規則

int chk2() {
  table();
  int flag=0;
  for (int a=0;a<16;a++) {
          if (chk(a)==now[a])flag++;
  }
  if (flag==16)return 1;
  else return 0;
}//どのユニットもエネルギーを減らすように変化できないか確認する関数
int change(int a) {
  if (chk(a)!=-1)now[a]=chk(a);
  return 0;
}//実際に状態を変化させる関数
int go(int a, int kai) {
  change(a);
  pic(a, kai);
  return 0;
}//状態変化を実際に描写する関数
int pics() {
  for (int i=0;i<16;i++) {
          if (now[i]==1)fill(0);
          else fill(255);
          rect((i%4)*16, (i/4)*16, 14, 14);
  }
  float w=ene();
  int lc;
  if ((w<10.0)&&(w>-10.0)) {
          lc=1;
  }
  else {
          lc=2;
  }//文字列としてエネルギーを表現するため
  fill(0);
  String wa=nf(w, lc, 2);
  textSize(20);
  text(wa, 0, 80);
```

```
  return 0;
}//初期状態を描写する関数

int pic(int q, int kai) {
  for (int i=0;i<16;i++) {
          if (now[i]==1)fill(0);
          else fill(255);
          rect((i%4)*16+((kai+1)%9)*80, (i/4)*16+((kai+1)/9)*80, 14, 14);
  }
  float w=ene();
  fill(0);
  int lc;
  if ((w<10.0)&&(w>-10.0)) {
          lc=1;
  }
  else {
          lc=2;
  }
  fill(0);
  String wa=nf(w, lc, 2);
  textSize(20);
  text(wa, ((kai+1)%9)*80, 80+((kai+1)/9)*80);
  fill(0, 128, 255);
  rect((q%4)*16+((kai)%9)*80+3, (q/4)*16+((kai)/9)*80+3, 8, 8);
  return 0;
}//変化中の状態を描写する関数
int picp(int kai) {
  for (int i=0;i<16;i++) {
          if (pat[kai][i]==1)fill(0);
          else fill(255);
          rect((i%4)*16+(kai%9)*80, (i/4)*16+(kai/9)*80+600, 14, 14);
          fill(0);
  }
  fill(0);
  textSize(40);
  text(kai+1, kai*80, 600);
  ene();
  return 0;
}//記憶させた状態を描写する関数

int OK() {
  int sum=0;
  int ans=-1;
  for (int a=0;a<3;a++) {
          sum=0;
          for (int b=0;b<16;b++) {
          if (now[b]!=pat[a][b])break;
```

```
            else sum=sum+1;
            }
            if (sum==16)ans=a;
            else sum=0;
  }
  return ans;
}//もし記憶させたパターンのどれかを想起した場合どのパターンを想起したか調べる関数
void draw() {
  sikaku();
  for (int a=0;a<3;a++) {
            picp(a);
  }
  if (flag==1) {
            for (int x=0;x<16;x++) {
            now[x]=col[x];
            }
            fill(128, 255, 128);
            rect(0, 0, 800, 600);
            pics();
            int ka;
            for ( ka=0;ka<62;ka++) {
            int q=int(random(0, 16));
            go(q, ka);

            if (chk2()==1)   break;
            if (ka==61) {
            fill(0);
            textSize(40);
            text("......", 710, 515);
            }
            }
            textSize(50);
            if (OK()!=-1) {
            fill(0);
            text(OK()+1, 250, 600);
            }
            else if (ka==62) {
            fill(255, 128, 0);
            text("TIMEOUT!", 250, 600);
            }
            else {
            fill(255, 0, 0);
            text("MISS!", 250, 600);
            }
            textSize(60);
            fill(0);
            text("Hopfield", 600, 700);
```

```
        flag=0;
        //stop=1;
  }
}
void mouseClicked() {
  int x=5;
  int y=5;
  if (mouseX>800) {
        if (mouseX%50<=40&&mouseY%50<=40&&mouseX<=1190&&mouseY<=190) {
        x=mouseX/50-16;
        y=mouseY/50;
        }
        else if (mouseX>850&&mouseY>550) {
        if (flag==0) {
        flag=1;
        }
        else {
        flag=2;
        }
        }
        else {
        x=5;
        y=5;
        }
        if (x<4&&y<4&&flag==0)
        col[x+4*y]=(1+col[x+4*y])%2;
  }
}
```

2.Boltzman Machine

```
void setup() {
  size(1000, 700);
  background(255, 240, 255);
}
int pat[][]= {
  {0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1},
  {0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0},
  {1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1}
};//記憶させたパターン
int now[]= {
  1, 1, 1, 1,
  1, 1, 1, 1,
  1, 1, 1, 1,
  1, 1, 1, 1
};//ネットワークの状態
int col[]= {
  1, 1, 1, 1,
  1, 1, 1, 1,
  1, 1, 1, 1,
  1, 1, 1, 1
};//初期状態を決めるための配列
int flag=0;
int wij[][]=new int[16][16];
int weight(int i, int j) {
  int y=0;
  for (int a=0;a<3;a++) {
          y=y+((2*pat[a][i])-1)*(2*pat[a][j]-1);
  }
  if (i==j) return 0;
  else return y;
}
int table() {
  for (int c=0;c<16;c++) {
          for (int d=0;d<16;d++) {
          wij[c][d]=weight(c, d);
          }
  }
  return 0;
}
float ene() {
  table();
  float e=0.0;
  for (int q=0;q<16;q++) {
          for (int w=0;w<16;w++) {
          e=e-((wij[q][w]*now[q]*now[w])/2.0);
          }
```

```
  }
  return e;
}
float inter(int k) {
  table();
  float b=0.0;
  for (int a=0;a<16;a++) {
          if (a!=k) {
          b=b+wij[k][a]*now[a];
          }
  }
  return b;
}//ユニットの入力の総和


int pic(int q, int kai) {
   for (int i=0;i<16;i++) {
  fill(col(i));
          rect((i%4)*16+((kai+1)%9)*80, (i/4)*16+((kai+1)/9)*80, 14,14);
  }
  float w=ene();
  fill(0);
  int lc;
  if ((w<10.0)&&(w>-10.0)) {
          lc=1;
  }
  else {
          lc=2;
  }
  fill(0);
  String wa=nf(w, lc, 2);
  textSize(20);
  text(wa, ((kai+1)%9)*80, 80+((kai+1)/9)*80);
   fill(0,255,255);
  rect((q%4)*16+((kai+1)%9)*80, (q/4)*16+((kai+1)/9)*80, 8, 8);
  return 0;
}
int picp(int kai) {
  for (int i=0;i<16;i++) {
          if (pat[kai][i]==1)fill(0);
          else fill(255);
          rect((i%4)*16+(kai%12)*80, (i/4)*16+(kai/12)*80+600,14, 14);
          fill(0);
          //float a=inter(i);
          //textSize(20);
          //text(a,350,i*20+300);
  }
```

```
  fill(0);
  ene();
  return 0;
}
float boltz(float ii,float T){
  return 1/(1+exp(-1*ii/T));
}//boltzman machine の状態変化における状態を 1 に変化させる確率
float Tdown(int kai){
if(kai<10) return 20.0;
else if(kai<20) return 15.0;
else if(kai<40) return 10.0;
else if(kai<60) return 7.5;
else if(kai<85) return 5.0;
else return 3.0;
}//Boltzman machine においてのネットワークの温度を下げる部分を実装するための関数
int henka(int a,float T){
 float d=inter(a);
 float  c=boltz(d,T);
 int mother[]=new int[100000];
for(int i=0;i<100000;i++){
 if(i<100000*c){
  mother[i]=1;
 }else mother[i]=0;
}
 int b=int(random(0,100000));
return mother[b];
}//実際に状態を変化させるために母集団より値を引くための関数
int sikaku(){
            for(int i=0;i<16;i++){fill((1-col[i])*255);
            rect(i%4*50+800,i/4*50,40,40);
  }fill(255,255,0);
  rect(850,550,150,150);
  if(flag==0){
            fill(0,0,255);textSize(40);
            text("START",865,630);
  }
  return 0;
}

int col(int i){
  if(now[i]==1)return 0;
  else return 255;
}
int change(int a,float T) {
 now[a]=henka(a,T);
  return 0;
}
```

```
int go(int a, int kai,float T) {
  change(a,T);
  pic(a, kai);
  return 0;
}
int pics() {
        for (int i=0;i<16;i++) {
        if (now[i]==1)fill(0);
        else fill(255);
        rect((i%4)*16, (i/4)*16,14,14);
        }
        float w=ene();
        int lc;
        if ((w<10.0)&&(w>-10.0)) {
        lc=1;
        }
        else {
        lc=2;
        }
        fill(0);
        String wa=nf(w, lc, 2);
        textSize(20);
        text(wa, 0, 80);
        return 0;
  }
void draw() {
  for (int a=0;a<3;a++) {
        picp(a);
  }
  sikaku();
  if(flag==1){
        for(int q=0;q<16;q++){
        now[q]=col[q];
        }
 fill(255,245,255);
 rect(0,0,800,600);
  pics();
for (int ka=0;ka<62;ka++) {
        int q=int(random(0, 16));
        float T=Tdown(ka);
        go(q, ka,T);
        fill(0);
        textSize(40);
  text(".....",710,510);
  }
  textSize(60);
 fill(0);
  text("Boltzman",500,700);
```

```
  flag=0;
          }
}
void mouseClicked(){
int x=5;
int y=5;
if(mouseX>800){
if(mouseX%50<=40&&mouseY%50<=40&&mouseX<=1290&&mouseY<=190){
  x=mouseX/50-16;
  y=mouseY/50;
}else if(mouseX>850&&mouseY>550){
  if(flag==0){
  flag=1;
  }else{
   flag=2;
  }
}else{
  x=5;
  y=5;
}
  if(x<4&&y<4&&flag==0)
  col[x+4*y]=(1+col[x+4*y])%2;
}
}
```

付録 ソースコード

kinectV2

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <array>
#define _USE_MATH_DEFINES
#include <cmath>
#include <atlbase.h>
#include <Math.h>
#include <fstream>
#include <Windows.h>
#include <Kinect.h>
#include <Kinect.Face.h>
#include <opencv2/opencv.hpp>

#define ERROR_CHECK( ret )                                          \
        if( FAILED( ret ) ){                                        \
                std::stringstream ss;                               \
                ss << "failed " #ret " " << std::hex << ret << std::endl;          \
                throw std::runtime_error( ss.str().c_str() );       \
```

```
                }

class KinectApp
{
private:
        CComPtr<IKinectSensor> kinect;
        CComPtr<IColorFrameReader> colorFrameReader;
        CComPtr<IBodyFrameReader> bodyFrameReader;
        CComPtr<ICoordinateMapper> coordinateMapper;
        std::vector<BYTE> colorBuffer;
        int colorWidth;
        int colorHeight;
        unsigned int colorBytesPerPixel;
        ColorImageFormat colorFormat = ColorImageFormat::ColorImageFormat_Bgra;
        cv::Mat colorImage;
        CComPtr<IHighDefinitionFaceFrameReader> hdFaceFrameReader;
        CComPtr<IFaceModelBuilder> faceModelBuilder;
        CComPtr<IFaceAlignment> faceAlignment;
        CComPtr<IFaceModel> faceModel;
        std::array<float, FaceShapeDeformations::FaceShapeDeformations_Count> shapeUnits;
        UINT32 vertexCount;
        UINT64 trackingId;
        int trackingCount;
        bool produced;
        std::array<cv::Scalar, BODY_COUNT> colors;
        int font = cv::FONT_HERSHEY_SIMPLEX;

    std::string imgname = "CC"; //保存 file の名前
        std::array<float, 1347> px; //x 座標格納用配列
        std::array<float, 1347> py; //y 格納用配列
        std::array<int, 1347> pp; //必要な特徴点の番号
        std::array <cv::Mat, 40> cimage; //画像(Mat)格納用
        std::array<float, 40> Base; //目頭間
        float ML[40][3]; //MouthLeftcorner
        float MR[40][3]; //MouthRightcorner
        float MUM[40][3]; //MouthUpperlipMidbottom
        float MLM[40][3]; //MouthLowerlipMidtop
        float Nose[40][3]; //NoseTip
        float Chin[40][3]; //ChinCenter
        float LCheek[40][3]; //LeftcheekCenter
        float RCheek[40][3]; //RightcheekCenter
        float LBCheek[40][3]; //LowerjawLeftend
        float RBCheek[40][3]; //LowerjawRightend
        float LUM[40][3]; //口の左上
        float RUM[40][3]; //口の右上
        float LBM[40][3]; //口の左下
```

```
	float RBM[40][3]; //口の右下
	int r = 0;
	float frame = 500;

	std::array<float, 40> w; //口の横幅
	std::array<float, 40> h; //口の縦幅
	std::array<float, 40> d; //鼻-あご

public:
	void initialize()
	{
		// Sensor を取得
		ERROR_CHECK(GetDefaultKinectSensor(&kinect));
		ERROR_CHECK(kinect->Open());
		BOOLEAN isOpen;
		ERROR_CHECK(kinect->get_IsOpen(&isOpen));
		if (!isOpen){
			throw std::runtime_error("failed IKinectSensor::get_IsOpen( &isOpen )");
		}

		ERROR_CHECK(kinect->get_CoordinateMapper(&coordinateMapper));

		// Color Frame Source を取得、Color Frame Reader を開く
		CComPtr<IColorFrameSource> colorFrameSource;
		ERROR_CHECK(kinect->get_ColorFrameSource(&colorFrameSource));
		ERROR_CHECK(colorFrameSource->OpenReader(&colorFrameReader));

		CComPtr<IFrameDescription> colorFrameDescription;
		ERROR_CHECK(colorFrameSource->CreateFrameDescription(colorFormat,
&colorFrameDescription));
		ERROR_CHECK(colorFrameDescription->get_Width(&colorWidth));
		ERROR_CHECK(colorFrameDescription->get_Height(&colorHeight));
		ERROR_CHECK(colorFrameDescription-
>get_BytesPerPixel(&colorBytesPerPixel));

		colorBuffer.resize(colorWidth * colorHeight * colorBytesPerPixel);

		// Body Frame Source を取得、Body Frame Reader を開く
		CComPtr<IBodyFrameSource> bodyFrameSource;
		ERROR_CHECK(kinect->get_BodyFrameSource(&bodyFrameSource));
		ERROR_CHECK(bodyFrameSource->OpenReader(&bodyFrameReader));

		// HDFace を初期化
		initializeHDFace();

		// Lookup Table
		colors[0] = cv::Scalar(255, 0, 0);
```

```
		colors[1] = cv::Scalar(0, 255, 0);
		colors[2] = cv::Scalar(0, 0, 255);
		colors[3] = cv::Scalar(255, 255, 0);
		colors[4] = cv::Scalar(55, 0, 255);
		colors[5] = cv::Scalar(0, 255, 255);

		for (int i = 0; i < 1347; i++){
			pp[i] = -1;
		}

		pp[1124] = pp[1125] = pp[1153] = pp[1152] = 1;//口
		pp[210] = pp[843] = 1;//目頭
		pp[91] = pp[687]= pp[1072] = pp[10] = pp[18] = pp[4] =1; //必要な特徴点の番号
		pp[412] = pp[933] = pp[1307] = pp[1327] = 1;//ほほ
	}

	void run()
	{
		while (1){
			update();
			draw();
			int key = cv::waitKey(10);
			if (key == VK_ESCAPE){
				cv::destroyAllWindows();
				break;
			}
			if (key == VK_SPACE){ //スペースを押すと開始
				//スペース押すたびに
				cimage[r] = colorImage.clone();
				float base = std::sqrt(std::pow(px[210] - px[843], 2) +
std::pow(py[210] - py[843], 2));//目頭間

				ML[r][0] = px[91];
				ML[r][1] = py[91];
				MR[r][0] = px[687];
				MR[r][1] = py[687];
				MUM[r][0] = px[1072];
				MUM[r][1] = py[1072];
				MLM[r][0] = px[10];
				MLM[r][1] = py[10];
				Nose[r][0] = px[18];
				Nose[r][1] = py[18];
				Chin[r][0] = px[4];
				Chin[r][1] = py[4];
				LCheek[r][0] = px[412];
				LCheek[r][1] = py[412];
				RCheek[r][0] = px[933];
```

```
                                RCheek[r][1] = py[933];
                                LBCheek[r][0] = px[1307];
                                LBCheek[r][1] = py[1307];
                                RBCheek[r][0] = px[1327];
                                RBCheek[r][1] = py[1327];
                                LUM[r][0] = px[1124];
                                LUM[r][1] = py[1124];
                                RUM[r][0] = px[1153];
                                RUM[r][1] = py[1153];
                                LBM[r][0] = px[1125];
                                LBM[r][1] = py[1125];
                                RBM[r][0] = px[1152];
                                RBM[r][1] = py[1152];

                                w[r] = std::sqrt(std::pow(ML[r][0] - MR[r][0], 2) +
std::pow(ML[r][1] - MR[r][1], 2));//口の横幅
                                printf("口の横幅: %f\n", w[r]);
                                h[r] = std::sqrt(std::pow(MUM[r][0] - MLM[r][0], 2) +
std::pow(MUM[r][1] - MLM[r][1], 2));//口の縦幅
                                printf("口の縦幅: %f\n", h[r]);
                                d[r] = std::sqrt(std::pow(Chin[r][0] - Nose[r][0], 2) +
std::pow(Chin[r][1] - Nose[r][1], 2));//鼻-あご
                                printf("鼻-あご: %f\n\n", d[r]);
                                Base[r] = base;
                                r++;
                        }
                                if (key == VK_RETURN){ //エンター押すと開始
                                for (int j = 0; j < 5; j++){
                                        /*if (!cimage[j].empty()){
                                                std::string image = imgname + std::to_string(j) +
".png";
                                                cv::imwrite(image, cimage[j]); //画像の保存
                                        }*/
                                        fileout(j); //テキストに保存
                                }
                                }
                                if (key == VK_TAB) finishA();
                                if (key == VK_BACK) finishB();
                        }
        }

private:

        void initializeHDFace()
        {
                // HDFace Frame Source を取得
```

```
		CComPtr<IHighDefinitionFaceFrameSource> hdFaceFrameSource;
		ERROR_CHECK(CreateHighDefinitionFaceFrameSource(kinect,
&hdFaceFrameSource));

		// HDFace Frame Reader を開く
		ERROR_CHECK(hdFaceFrameSource->OpenReader(&hdFaceFrameReader));

		// Face Alignment を作成
		ERROR_CHECK(CreateFaceAlignment(&faceAlignment));

		// Face Model を作成
		ERROR_CHECK(CreateFaceModel(1.0f,
FaceShapeDeformations::FaceShapeDeformations_Count, &shapeUnits[0], &faceModel));
		ERROR_CHECK(GetFaceModelVertexCount(&vertexCount));

		// Face Model Builder を作成、開始
		FaceModelBuilderAttributes attribures =
FaceModelBuilderAttributes::FaceModelBuilderAttributes_None;
		ERROR_CHECK(hdFaceFrameSource->OpenModelBuilder(attribures,
&faceModelBuilder));
		ERROR_CHECK(faceModelBuilder->BeginFaceDataCollection());
	}

	void update()
	{
		updateColorFrame();
		updateBodyFrame();
		updateHDFaceFrame();
	}

	void updateColorFrame()
	{
		CComPtr<IColorFrame> colorFrame;
		HRESULT ret = colorFrameReader->AcquireLatestFrame(&colorFrame);
		if (FAILED(ret)){
			return;
		}

		ERROR_CHECK(colorFrame-
>CopyConvertedFrameDataToArray(static_cast<UINT>(colorBuffer.size()), &colorBuffer[0],
colorFormat));

		colorImage = cv::Mat(colorHeight, colorWidth, CV_8UC4, &colorBuffer[0]);
	}

	void updateBodyFrame()
	{
```

```
                CComPtr<IBodyFrame> bodyFrame;
                HRESULT ret = bodyFrameReader->AcquireLatestFrame(&bodyFrame);
                if (FAILED(ret)){
                        return;
                }

                std::array<CComPtr<IBody>, BODY_COUNT> bodies;
                ERROR_CHECK(bodyFrame->GetAndRefreshBodyData(BODY_COUNT,
&bodies[0]));

                // Sensor に最も近い Body を選択
                findClosestBody(bodies);
        }

        inline void findClosestBody(const std::array<CComPtr<IBody>, BODY_COUNT>& bodies)
        {
                float closest = FLT_MAX;
                for (int count = 0; count < BODY_COUNT; count++){
                        CComPtr<IBody> body = bodies[count];
                        BOOLEAN tracked;
                        ERROR_CHECK(body->get_IsTracked(&tracked));
                        if (!tracked){
                                continue;
                        }

                        // Joint(Head)を取得
                        std::array<Joint, JointType::JointType_Count> joints;
                        ERROR_CHECK(body->GetJoints(JointType::JointType_Count,
&joints[0]));
                        Joint joint = joints[JointType::JointType_Head];
                        if (joint.TrackingState == TrackingState::TrackingState_NotTracked){
                                continue;
                        }

                        // Sensor からの距離を算出( √( x^2 + y^2 + z^2 ) )
                        CameraSpacePoint point = joint.Position;
                        float distance = std::sqrt(std::pow(point.X, 2) + std::pow(point.Y, 2) +
std::pow(point.Z, 2));
                        if (closest <= distance){
                                continue;
                        }
                        closest = distance;

                        // Tracking ID を取得
                        UINT64 id;
                        ERROR_CHECK(body->get_TrackingId(&id));
                        if (trackingId != id){
```

```
                                trackingId = id;
                                trackingCount = count;
                                produced = false;

                                // Tracking ID を登録
                                CComPtr<IHighDefinitionFaceFrameSource> hdFaceFrameSource;
                                ERROR_CHECK(hdFaceFrameReader-
>get_HighDefinitionFaceFrameSource(&hdFaceFrameSource));
                                ERROR_CHECK(hdFaceFrameSource-
>put_TrackingId(trackingId));
                        }
                }
        }

        void updateHDFaceFrame()
        {
                // 最新の HDFace Frame を取得
                CComPtr<IHighDefinitionFaceFrame> hdFaceFrame;
                HRESULT ret = hdFaceFrameReader->AcquireLatestFrame(&hdFaceFrame);
                if (FAILED(ret)){
                        return;
                }

                // Tracking ID の登録確認
                BOOLEAN tracked;
                ERROR_CHECK(hdFaceFrame->get_IsFaceTracked(&tracked));
                if (!tracked){
                        return;
                }

                // HDFace Frame Result を取得
                ERROR_CHECK(hdFaceFrame-
>GetAndRefreshFaceAlignmentResult(faceAlignment));
                if (faceAlignment != nullptr){
                        // 結果を取得、描画
                        result();
                }
        }

        inline void result()
        {
                // Vertexs(1347 点)の取得、描画
                std::vector<CameraSpacePoint> vertexs(vertexCount);
                ERROR_CHECK(faceModel->CalculateVerticesForAlignment(faceAlignment,
vertexCount, &vertexs[0]));
                for (int i = 0; i < 1347; i++){
                        std::array<ColorSpacePoint, 1347> point;
```

```
                        ERROR_CHECK(coordinateMapper->MapCameraPointToColorSpace(vertexs[i], &point[i]));
                        if(pp[i] == 1){
                            int x = static_cast<int>(std::ceil(point[i].X));
                            int y = static_cast<int>(std::ceil(point[i].Y));
                            if ((x >= 0) && (x < colorWidth) && (y >= 0) && (y < colorHeight)){
                                std::string index = std::to_string(i);
                                cv::circle(colorImage, cv::Point(x, y), 3, colors[trackingCount], -1, CV_AA); //特徴点の描画
                                px[i] = point[i].X; //x 座標を格納
                                py[i] = point[i].Y; //y 座標を格納
                            }
                        }
                    }
                }

        int fileout(int i){ //テキストに保存
                std::string dname ="c:\\home\\" + imgname + ".txt"; //ニューラルネットワークの場所
                std::ofstream txt(dname, std::ios::out | std::ios::app);

                txt << ML[i][0] << "," << ML[i][1] << "," << MR[i][0] << "," << MR[i][1] << "," << MUM[i][0] << ","
                        << MUM[i][1] << "," << MLM[i][0] << "," << MLM[i][1] << "," << Nose[i][0] << "," << Nose[i][1] << ","
                        << Chin[i][0] << "," << Chin[i][1] << ","<< LCheek[i][0] << "," << LCheek[i][1] << "," << RCheek[i][0] << ","
                        << RCheek[i][1] << "," << LBCheek[i][0] << "," << LBCheek[i][1] << "," << RBCheek[i][0] << ","  << RBCheek[i][1]<< ","
                        << LUM[i][0] << "," << LUM[i][1] << "," << RUM[i][0] << ","<< RUM[i][1] << "," << LBM[i][0] << "," << LBM[i][1] << ","
                        << RBM[i][0] << "," << RBM[i][1] << ","
                    << w[i] << "," << h[i] << "," << d[i] << "," << Base[i] << std::endl;

                return 0;

        }

        void drawmetro(){ //メトロノームの表示
                        cv::rectangle(colorImage, cv::Point(100, 100), cv::Point(150, 500), cv::Scalar(240, 100, 100), 2, CV_AA);
                        cv::rectangle(colorImage, cv::Point(100, frame), cv::Point(150, 500), cv::Scalar(240, 100, 100), -1, CV_AA);
                        cv::line(colorImage, cv::Point(100, 180), cv::Point(150, 180), cv::Scalar(0, 0, 0), 2, CV_AA);
                        cv::line(colorImage, cv::Point(100, 260), cv::Point(150, 260), cv::Scalar(0, 0,
```

```
0), 2, CV_AA);
                        cv::line(colorImage, cv::Point(100, 340), cv::Point(150, 340), cv::Scalar(0, 0,
0), 2, CV_AA);
                        cv::line(colorImage, cv::Point(100, 420), cv::Point(150, 420), cv::Scalar(0, 0,
0), 2, CV_AA);
                        frame -= 4;
                        if (frame < 100) frame = 500;
        }

        void draw()
        {
                drawmetro();
                cv::circle(colorImage, cv::Point(colorWidth/3, colorHeight/3 - 100), 5,
cv::Scalar(0,255,255), -1, CV_AA); //鼻基準点
                drawHDFaceFrame();
        }

        void drawHDFaceFrame()
        {
                // HDFace の表示
                if (!colorImage.empty()){
                        cv::imshow("HDFace", colorImage);
                }
        }
};

void main()
{
        try{
                KinectApp app;
                app.initialize();
                app.run();
        }
        catch (std::exception& ex){
                std::cout << ex.what() << std::endl;
        }
}
```