公立はこだて未来大学 2015 年度 システム情報科学実習
グループ報告書

Future University Hakodate 2015 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

グループ名

なし

Group Name

Nothing

プロジェクト番号/Project No.

17

プロジェクトリーダ/Project Leader

1013152　斉藤健士郎　Kenshiro Saito

グループリーダ/Group Leader

グループメンバ/Group Member

1013110　大場和貴　Kazuki Oba
1013118　齋藤達哉　Tatsuya Saito
1013129　尾ヶ瀬拓哉　Takuya Ogase
1013152　斉藤健士郎　Kenshiro Saito
1013163　藤島真央　Mao Huzishima
1013204　中谷奨吾　Syogo Nakaya

指導教員

斉藤朝輝　川越敏司　川口聡

Advisor

Asaki Saito　Toshizi Kawagoe　Satoshi Kawaguchi

提出日

2016 年 1 月 20 日

Date of Submission

January 20, 2016

概要

本プロジェクトでは複雑系の中でも人間の脳の働きに関心を持ち, それらを深く理解するために, ニューラルネットワークを用いて人間の認知活動を模倣する数理モデルを作成した. そのモデルを利用し, 人間の認知活動の中でも私たちが特に関心を持った表情認識の数値実験を行った.

ここで述べる人間の認知活動とは, 外界からの影響を知覚し, それが何であるか判断, 解釈, 学習する脳の働きの過程のことを表す.

前期は, ニューラルネットワークに関しての学習に重点を置き, プロジェクトメンバー全員で輪読を計った. その際参考書は「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-」を使用した.

後期は, システム班と理論班に分かれ, 人の顔の表情識別について取り組んだ. システム班は真顔から怒り, 喜び, 驚きの 3 つの表情の変化を読み取り学習する装置を開発し,15 人の顔の表情のデータを読み取り, 理論班はその得られたデータから真顔と他の表情との特徴点の誤差を計算した. その後 15 人の被験者を募集し, 決められた 3 つの表情の中から自由な表情を浮かべてもらい, カメラで表情を撮影した際, 学習したデータの中から正しい表情が検出されるかの整合性を確認した.

その結果, 正しく表情を識別できた数を被験者の総数で割った値を識別率として定義すると, 特徴点をすべて取った場合の各表情の識別率は驚きが 93 %，喜びが 87 %，怒りが 80 %，目の特徴点を取らない場合の各表情の識別率はは驚きが 93 %，喜びが 80 %，怒りが 47 %，口の特徴点を取ない場合の各表情の識別率はは驚きが 20 %，喜びが 27 %，怒りが 80 %となった．特徴点をすべて取った場合は驚き，喜び，怒りの表情をそれぞれ識別できたが，目の特徴点を取らない場合には怒りの識別率が，口の特徴点を取らない場合には驚きと喜びの識別率が大きく減少した．

以上のことから, 表情を識別するにあたって重要となる顔のパーツが存在し, 重要となる顔のパーツは表情ごとに異なることが明らかになった.

キーワード　認知活動、ニューラルネットワーク、バックプロパゲーション、数値実験、表情認識

（※文責: 齋藤達哉）

Abstract

In this project we are interested in a systems of the brain in person. Using neural network, we made the mathematical model which imitate cognitive function of person to understand them. Moreover, we conducted numerical experiment with using neural network. Cognitive activities define the process that they perceive a influence from outside world and a brain determine, interpret and learn what it is. In the first term, we had learned neural network, and conducted reading circle. In the second term, we divided themselves into system group or theory group. The system group made equipment which read facial 3 expressions of person such as smile, anger and surprise. Moreover, we got 15 data before made actual experiment. The theory group manually made a calculation of 15 data in difference straight face and other facial expression. We performed connective expression for other 15 people data. Finally, we checked whether their expression is the right answer. The results were defined as follows. If there are all feature points, surprise is 93If there are only feature point of eyes, surprise is 93If there are only feature point of mouth, surprise is 20Thus, characterlistic point of mouth affected anger, also It' s of eyes affected smile and surprise. Taking each charcterlistic points were to increase correct answer. Therefore, there is the importance at each feature point when we perform each cognitive expression.

Keyword Cognitive function, Neural Network, Imitation of Cognitive function, Numerical Experiment

（※文責: 大場和貴）

目次

第 1 章　背景

生き物の中には感情を抱くと、それを表現することができる種が多く存在する。人間にとって身近な生き物である犬や猫などは人間にもわかるように感情を表すことができる。例えば、犬は喜ぶと尻尾を振るし、猫はお腹がすくと体をこすりつけてくる。ここで、人間の感情表現の方法を考えてみる。身振り手振り、声色などでも感情を表現できるが、「表情」による表現はそれら以上に感情を表現できると言える。喜ぶと笑顔になり、怒ると顔を歪め、驚くと目や口を大きく開く。他にも悲しかったり、疑問をもったときにする表情を人間は大体判別することができる。実際の生活では、人の顔色を見て機嫌の良し悪しを観察することは人付き合いやコミュニケーションを取る場合において重要である。近年では、人間が対象の人物の表情によって感情を識別するように、機械による表情識別ができるようになってきた。機械による表情識別の代表的な例として、Kinect と呼ばれる装置を用いることで、表情識別はもちろん、その表情を用いたゲームが出来る。また、人物の特徴を捉えることができるという点から防犯カメラとして設置し、容疑者の特定に利用できるとも考えられている。機械による表情識別は、主に人間の表情を基本 6 感情 (驚き、恐怖、嫌悪、怒り、幸福、悲しみ) を示す表情に分類することである。基本 6 感情を示す表情は図 1.1 のような表情である。

図 1.1　基本 6 感情

（※文責: 中谷奨吾）

1.1　問題設定

本プロジェクトの本来の目的は、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源流は人間の脳のモデル化であり、人間の脳の働きを模倣をするための分野ともいえる。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、表情識別プログラムを作成することを考えた。またその過程において、あることわざに焦点を当てた。「目は口ほどにものを言う」ということわざだ。このことわざは本来、「情のこもった目つきは、口で話すのと同じくらい気持ちを表現する。」という意味である。つまり、言葉で言わずとも目を見れば感情が理解できるという意味にとることができる。それでは、実際の表情による識別において目の持つ情報はどれほど大きいものなのか。また、目以外で大きな情報を持つと予想される「口」は情報をどれほど持つのか。本プロジェクトでは、ニューラルネットワークを用いた表情識別プログラムを作成し、さらにニューラルネットワークを用いた表情識別における顔部位の重要性を調べる実験を行うことによって、以上の点について検証していく。

（※文責: 中谷奨吾）

1.2 課題設定

本プロジェクトで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ために課題を 3 つ設定した。以下にその 3 つの課題を示す。

- ニューラルネットワークの数理と学習の理解
  本プロジェクトは、全てのメンバーがニューラルネットワークの知識が全く無い状態からスタートする。そのため、まず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る。また、ニューラルネットワークの数理を学んだ後に、ニューラルネットワークの学習について理解を深める。ニューラルネットワークの数理や学習は文献 [1]～文献 [5] を用いて理解を深める。またこの課題は前期に解決する課題として設定した。

次に後期に解決するべき 2 つの課題を示す。

- ニューラルネットワークを用いた表情識別プログラムの作成
  先行研究である文献 [6] を参考に、怒り、喜び、笑顔の 3 つの表情を識別するプログラムを作成する。そしてそのプログラムを用いて実際に 3 つの表情を識別させ、その識別率を求める。

- ニューラルネットワークを用いた表情識別の際に顔のどの部位が重要かを調べる。
  作成した表情認識プログラムを用いてニューラルネットワークによる表情識別の際に顔のどの部位が重要になるのかを調べる。この際、人が実際に表情を識別することによって表情識別の際に顔のどの部位が重要かを調べた先行研究 (文献 [7]) を参考にした。

（※文責: 中谷奨吾）

1.3 本プロジェクトの課題の特徴

前期に解決するべき課題であるニューラルネットワークの数理について理解する課題は、プロジェクトメンバーのほぼ全員がプロジェクトが始まった段階ではニューラルネットワークについてほぼ何の知識もなかった。よって各プロジェクトメンバーはニューラルネットワークに関する文献を読み、ニューラルネットワークの数理を理解する必要があった。また、ただ文献を読んで理解するだけでなく、実際にニューラルネットワークをコンピュータ上に実装して動かしてみて動作を確

認する必要もあった。

次に後期に解決するべき 1 つ目の課題であるニューラルネットワークを用いた人の表情識別プログラムの作成は、先行研究である文献 [6] を再現する形で達成できるものであるので、各プロジェクトメンバーが文献 [6] を読みニューラルネットワークによる表情識別を理解する必要があった。また実際にプログラムを作成する必要があるのである程度のプログラミングの力も要求された。

次に、後期に解決するべき 2 つ目の課題であるニューラルネットワークを用いての表情識別の際に重要となる顔部位を調べる課題は、先行研究として文献 [7] があり、この文献では人が表情を識別する際に重要となる顔の部位が調べられていた。しかしニューラルネットワークを用いた表情識別については調べられていなかったので、プロジェクトメンバーは実際に実験を行ってみることによってこの課題を解決しなければならなかった。

最後に上に挙げた 3 つ以外にも、発表会の時のデモンストレーションに使用するためのプログラム作成も上の 3 つの課題を解決するための補助的な課題として挙げられる。この補助的な課題を解決するためにマイクロソフトから発売されたジェスチャー・音声認識によって操作できるデバイスである Kinect を使用する必要があった。よって、プロジェクトメンバーはこの Kinect の使い方も学ぶ必要があった。さらにデモンストレーションのプログラムを作成するために、プロジェクトメンバーはプログラミングのスキルを上げる必要もあった。

（※文責: 中谷奨吾）

第 2 章　到達目標

2.1　本プロジェクトにおける目標

本プロジェクトでは複雑系の中でも人間の脳の働きに興味を持ち、その中でも特に人の認知過程について興味がある。そこで 1.2 節で挙げた 3 つの課題を解決することによって人の認知過程について調べられると思い、この 3 つの課題を解決することを本プロジェクトの目標とした。

1 つ目のニューラルネットワークの数理を理解する課題では、プロジェクトメンバー全員が基本的なニューラルネットワークの構造、ニューラルネットワークの種類、ネットワークの学習といった内容を理解する必要があった。さらにコンピュータ上にニューラルネットワークを実装する力をつけることも目標として設定した。

2 つ目のニューラルネットワークを用いての表情識別の課題は、実際にコンピュータ上にニューラルネットワークを実装して実験を行って見ることを目標とした。

3 つ目のニューラルネットワークによる表情識別の際の顔部位の重要性を調べる課題は 2 つ目の課題で作った表情識別の装置を用いて実際に実験を行い、ニューラルネットワークを用いての表情識別で顔のどの部位が重要かを調べることを目標とした。

（※文責: 斉藤健士郎）

2.1.1　通常の授業ではなく、プロジェクト学習で行う利点

1 つ目の課題は、本学ではパターン認識・ニューロコンピューティングという講義でニューラルネットワークについて学ぶことが出来る。しかしそこではニューラルネットワークのメカニズムについて学ぶことがメインで、実際に目に見えるものを作ったり、認知機能と深く結び付けて学習する機会が少ない。したがって、プロジェクト学習で取り上げることで、ニューラルネットワークがどのように認知機能と結びついているのかを学ぶ機会が増え、視覚化ツールを作成することによってニューラルネットワークについて理解できる人が増えることが利点となる。

2 つ目の課題はニューラルネットワークをコンピュータ上に実装する必要がある。プロジェクト学習でこれを行うことで複数人で協力して行うことができ、効率がよい。さらに、多くの顔のデータを集める必要もあるので、やはり複数人で協力して行ったほうが効率が良い。

3 つ目の課題は実際に実験を行う必要があり、一人では時間がかかる。しかし複数人で行うことによって効率よく実験を行うことが出来る。

以上から通常の授業で行うよりもプロジェクト学習で行ったほうがよいと考えられる。

（※文責: 斉藤健士郎）

2.2　手順・課題設定

前期では、後期に入ってからすぐに GUI ソフトウェアを作成できるよう、プロジェクトメンバー全員が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。そのために 5 月

から 7 月までニューラルネットワークの基礎・メカニズムについて、メンバー全員で週に一度輪読を行った。輪読では、ニューラルネットワークの基礎である階層型ネットワークと相互結合型ネットワークを中心に考え、そこに関係する計算式について学んだ。また、輪読で学んだネットワークを目に見える形で数値出力するプログラムを作成した。

後期では、人間の脳の働きを理解するために二つの課題を設定した。一つ目はニューラルネットワークに人の表情を識別させる課題、二つ目はニューラルネットワークによる表情識別の際の顔部位の重要性を調べる課題である。一つ目の課題を解決するために、プロジェクトメンバーそれぞれが文献 [6] を読み、ニューラルネットワークによる表情識別の方法を学んだ。そして実際に表情識別を行うためにコンピュータ上にニューラルネットワークを実装して実験を行った。二つ目の課題を解決するために、実際に人が表情を識別する際に顔のどの部位が重要かが調べられている文献 [7] を読み、この実験をニューラルネットワークを用いて行ってみることにした。

（※文責: 斉藤健士郎）

2.2.1 具体的な習得技術と解決過程について

各課題を解決過程に関する講義として、3 年次後期の複雑知能学科に「ニューロコンピューティング」の授業がある。前期では、この授業のレポート課題を用いてニューラルネットワークの学習を深めていった。後期では、ニューラルネットワークをプログラムする上で、C++ や C ♯などの言語を取り扱った。また、人の認知機能に関する文献を熟読した。これらを学習したうえで、ニューラルネットワークによる人の表情認識の文献を見つけ (文献 [6])、これを題材にプロジェクトで人間の表情認識のモデルを開発することに至った。

（※文責: 斉藤健士郎）

2.3 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を理解、実践できるようにするため、プロジェクトメンバー全員で参考書を用いて輪読を行った。輪読は各人の負担が均等になるように 1 人が 1 章を担当し、それぞれが担当した章について学習を進めて、ニューラルネットワークの基礎知識を身につけることにした。それとは別にニューラルネットワークの数値実験のシミュレーションをするために、階層型ネットワークと相互結合ネットワークを視覚化させるプログラムをグループリーダーが作成した。また、そのプログラムを他のグループメンバーに説明し、前期の目標であるニューラルネットワークの理解を深めた。後期の割り当ては、実際に表情認識を行うに当たって必要な参考文献を探した。また実際に各メンバーで被験者を集めて、顔の特徴点のデータを各々の表情のデータとの差を手計算をするグループと、実際にバックプロパゲーションと呼ばれるニューラルネットワークのプログラムの作成を行うものにグループに分かれた。数理モデル自体の全体の手法やその他のプログラムはリーダーが主体となって割り当てた。理由として、各個人の得意とする部分で分担して行うことにより、プロジェクト全体の作業効率が上がるからである。

（※文責: 大場和貴）

第 3 章　課題解決のプロセスの概要

前期の各月は以下のような活動を課題解決のために行った。

- 4 月
  プロジェクト配属期間のため、主な活動はなし。

- 5 月
  プロジェクトリーダーとプロジェクト活動の方針を決めた。方針としては、まず前期はメンバー全員で水曜日に文献 [1] の輪読を行い、それ以外の時間を輪読のための自習とすることにした。後期は、前期の輪読を終えた後にメンバー間で最終成果物に何を作るかを決定し、それのプログラムで視覚化することにした。
  さらに各メンバーの輪読の範囲を決定し、各自ニューラルネットワークの学習を開始した。
  - 大場和貴
    ・輪読では相互結合型ネットワークの学習の箇所を担当
  - 斉藤達哉
    ・輪読では階層型ネットワークの学習の箇所を担当
  - 尾ヶ瀬拓哉
    ・輪読では誤差逆伝搬法の箇所を担当
  - 斉藤健士郎
    ・輪読ではニューラルネットワークの基礎とパーセプトロンのネットワークを担当
    ・プロジェクトリーダーを担当
  - 藤島真央
    ・輪読ではボルツマンマシンの箇所を担当
  - 中谷奨吾
    ・輪読ではホップフィールドネットワークの箇所を担当

- 6 月
  5 月に引き続き輪読を行った。また、輪読で学んだメカニズムを視覚化するプログラムを C 言語と DirectX を用いて作成した。また、7 月にある中間発表のためのプログラムに何を作るか構想を練った。
  - 大場和貴
    ・輪読の担当箇所がまだ来てないので、引き続き発表のために予習
  - 斉藤達哉
    ・輪読の担当箇所がまだ来てないので、引き続き発表のために予習
  - 尾ヶ瀬拓哉
    ・輪読の担当箇所が終わったので、プログラムの作成の補佐
  - 斉藤健士郎
    ・輪読の担当箇所が終わり、プログラムの作成をメインで始めた.
    具体的にはホップフィールドネットワークとボルツマンマシンの実装である.
  - 藤島真央
    ・輪読の担当箇所が終わり、他の参考書や参考論文を探した.
    ・ボルツマンマシンの所を輪読の際担当したので、実装するときに理論的な部分で手助

けをした。
- 中谷奨吾

・輪読の担当箇所がまだ来てないので、引き続き発表のために予習

- 7 月

中間発表のために、今まで学習したことをまとめた。中間発表前に一度、担当教員の前で発表して評価をしてもらい、注意された点を修正した。中間発表では、前半と後半の 2 チームに分かれ、発表する部分を 3 人で均等にわけ、全員で発表を行った。その後中間発表のアンケートを読み、反省点について話合った。また、グループ報告書、個人報告書を Tex を使用し作成した。

- 大場和貴

・中間発表では前半の発表を担当し、主にボルツマンマシンの説明を行った。また、発表の際使用するスライドの作成の補助も担当した。
- 斉藤達哉

・中間発表では前半の発表を担当し、主にニューラルネットワークの基礎的な部分の説明を行った。発表の際用いるポスターの作成の補助も担当した。
- 尾ヶ瀬拓哉

・中間発表では後半の発表を担当し、主にニューラルネットワークの基礎的な部分の説明を行った。さらに発表全体の原稿の作成を中心となって行った。
- 斉藤健士郎

・中間発表では後半の発表を担当し、主にボルツマンマシンの説明を行った。またデモ用プログラムの作成を担当した
- 藤島真央

・中間発表では後半の発表を担当し、主にホップフィールドネットワークの説明を行った。さらに発表の際使用するポスターの作成を中心となって行った。
- 中谷奨吾

・中間発表では前半の発表を担当し、主にホップフィールドネットワークの説明を行った。さらに発表の原稿作成の補助も担当した。

- 8 月・9 月

夏休みを利用し、文献 [8] を各メンバーが読み、後期にニューラルネットワークと認知機能をどう結び付けるかを各々考えてくることにした。

- 大場和貴

・参考文献等を使い予習
- 斉藤達哉

・参考文献等を使い予習
- 尾ヶ瀬拓哉

・参考文献等を使い予習
- 斉藤健士郎

・夏休み中文献 [8] を読み、さらにニューラルネットワークを人の認知的課題に応用するために文献 [6] を読み、ニューラルネットワークによる人の表情識別を後期に行うことをプロジェクトメンバーに提案した。
- 藤島真央

・参考文献等を使い予習
- 中谷奨吾

・参考文献等を使い予習

- 10 月

人の認知機能とニューラルネットワークを結びつけるために、人の表情の識別について調べることにした。また、ニューラルネットワークと表情認識についての先行研究である文献 [6] をメンバー全員で読み、3 層の階層型ネットワークとバックプロパゲーションを用いたアルゴリズムを作成することにした。

- 大場和貴

  ・ニューラルネットワークを用いて人の認知過程を調べる文献 [6] 以外の文献を探した。さらにニューラルネットワークの実装の補助も行った。具体的には学習に用いるデータをフォルダから読みこみ、楽に学習を行えるようにするためのプログラムの作成を行った。
- 斉藤達哉

  ・ニューラルネットワークを用いて人の認知過程を調べる文献 [6] 以外の文献を探した。
- 尾ヶ瀬拓哉

  ・ニューラルネットワークの根幹部分の実装を行った。
- 斉藤健士郎

  ・顔写真から手動で特徴点を抽出するプログラムを作成した。顔画像を読み込み、そこに人が手動で点を打ち込むプログラムである。
- 藤島真央

  ・ニューラルネットワークの根幹部分の実装を行った。
- 中谷奨吾

  ・ニューラルネットワークの根幹部分の実装の補助を行った。具体的には学習に用いるデータをフォルダから読み込めるようにして、楽にネットワークの学習を行えるようにするプログラムを作成した。

- 11 月

文献 [6] を基礎として、喜び・怒り・驚きの 3 つの表情を識別するための装置として、Processing を用いて抽出した顔の特徴点から自動的にニューラルネットワークに与える入力データを作るプログラムを作成し、被験者を募ってデータを収集した。さらに人が表情を識別する際に顔のどの部位が重要であるかを調べている文献 [7] を読みニューラルネットワークに応用した。

また、12 月に控える期末発表に向けてスライドの作成と、Kinect を用いたデモ用のプログラムを作成することを決めた。

- 大場和貴

  ・最終発表の時に使用するポスターの作成を中心となって行った。
- 斉藤達哉

  ・最終発表の時に使用するポスターの作成を中心となって行った。
- 尾ヶ瀬拓哉

  ・ニューラルネットワークによる表情識別の際に顔のどの部位が重要かを実際に実験をして調べた。
- 斉藤健士郎

  ・最終発表のデモンストレーションの際に使用するプログラムの作成を中心となって行った。
- 藤島真央

  ・最終発表の時に使用する発表スライドの作成を中心となって行った。
- 中谷奨吾

  ・最終発表の時に使用するポスターのを中心となって行った。また表情識別の実験の補佐も行った。

- 12 月

中間発表の時と同じように、期末発表の前に数度、担当教員の前で発表の練習を行い、注意された点を修正していった。期末発表では前期と同じグループに分かれ発表した。その後、冬休みを利用し個人・グループ期末報告書を作成した。

- 大場和貴

  ・最終発表では主に表情識別には顔のどの部位が重要であるかを調べる実験の結果とデモンストレーションの説明をし、グループ報告書ではニューラルネットワークの部分を担当した。
- 斉藤達哉

  ・最終発表では主に表情識別には顔のどの部位が重要であるかを調べる実験の概要の説明をし、グループ報告書ではプロジェクトの概要の部分を担当した。
- 尾ヶ瀬拓哉

  ・最終発表では主に表情識別には顔のどの部位が重要であるかを調べる実験の概要の説明をし、グループ報告書ではニューラルネットワークによる表情識別の実験と、表情識別の際に顔のどの部位が重要かを調べる実験についての部分を担当した。
- 斉藤健士郎

  ・最終発表では主に表情識別には顔のどの部位が重要であるかを調べる実験の結果とデモンストレーションの説明をし、グループ報告書では書くプロジェクトメンバーの役割の部分を担当した。
- 藤島真央

  ・最終発表では主に表情識別には顔のどの部位が重要であるかを調べる実験の方法について説明をし、グループ報告書ではプロジェクト学習の結果の部分を担当した。
- 中谷奨吾

  ・最終発表では主に表情識別には顔のどの部位が重要であるかを調べる実験の方法について説明をし、グループ報告書ではプロジェクト学習を行うにあたっての背景についての部分を担当した。

（※文責: 斉藤健士郎）

第 4 章　課題解決のプロセスの詳細

本プロジェクトでは第 1 章で述べたように、第 1 の課題としてニューラルネットワークの数理を理解することを設定した. さらに第 2 の課題としてニューラルネットワークでの表情識別を行い、第 3 の課題としてニューラルネットワークにおける表情識別の際に重要となる顔部位を調べた。この章では、これらのニューラルネットワークや表情識別、さらに顔部位の重要性を調べる実験について説明する. そして、補助的な課題であるデモンストレーションの際のプログラム作成で必要になる Kinect についても説明する。さらにプロジェクトメンバーそれぞれの役割も説明する.

（※文責: 尾ヶ瀬拓哉）

4.1　ニューラルネットワーク

人間の脳はニューロンと呼ばれる多数の神経細胞で構成されていて、それぞれのニューロンは他のニューロンと結合しネットワークをなしている. ニューラルネットワークとはこのニューロンを数理モデル化し (ユニットという)、多数に繋いだもののことをいい、ユニットの繋ぎ方によって階層型ネットワーク (図 4.1) と相互結合型ネットワーク (図 4.2) の二つに分類することができる.

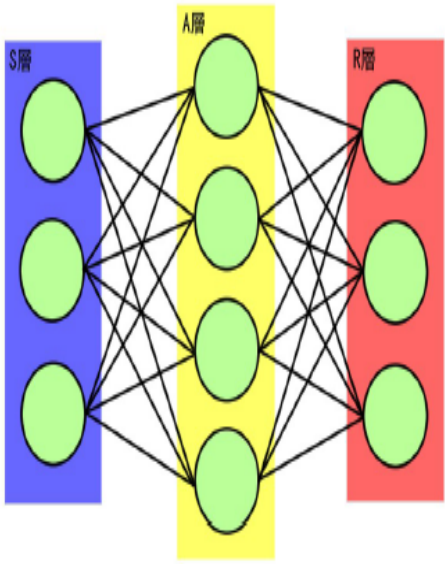

図 4.1　階層型ネットワーク

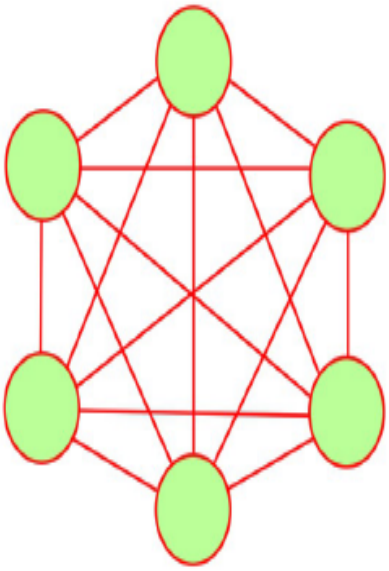

図 4.2　相互結合型ネットワーク

（※文責: 尾ヶ瀬拓哉）

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、学習する識別機械であり、階層的ネットワークの基本形である。パーセプトロンの各層は適当な数のユニットからなり、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。${i^k}_i$ は第 k 層の i 番目のユニットの入力の総和とし,${o^k}_i$ は第 k 層の i 番目のユニットの出力とし、${w^{k-1\,k}}_{j\,i}$ 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、

$$ {i^k}_i = \sum_j {w^{k-1\,k}}_{j\,i} o_j^{k-1} - {\theta^{\,k}}_i \tag{4.1} $$

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

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、それぞれ以下の定義である。

$$ f(x) = \begin{cases} 1(x \geqq 0 \text{ のとき}) \\ 0(x < 0 \text{ のとき}) \end{cases} \tag{4.3} $$

しきい値を X とすると

$$ f(x) = \begin{cases} 1(x \geqq X \text{ のとき}) \\ \frac{x+X}{2X}(|X| < x \text{ のとき}) \\ 0(x < X \text{ のとき}) \end{cases} \tag{4.4} $$

$$ f(x) = \frac{1}{1 + exp(-x)} \tag{4.5} $$

$$ f(x) = x \tag{4.6} $$

（※文責: 齋藤達哉）

4.1.2 相互結合ネットワーク

相互結合ネットワークは、ユニットが相互に結合しているネットワークである (図 4.2)。原則として、ある一つのユニットは他の全てのユニットと結合を持つ。また、階層型ネット

ワークと相互結合ネットワーク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が終わるのに対し、相互結合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。(図 4.2)

ホップフィールドネットワーク　相互結合ネットワークの代表的なものとしてホップフィールドネットワークがある。このネットワークの特徴は、

・二つのユニット間の結合荷重の値は対称

・ユニットの自身への結合荷重は 0

の 2 点である。つまり、ユニット i から j への結合荷重を w_{ij} とすると、

$$\begin{aligned} w_{ij} &= w_{ji} \\ w_{ii} &= 0 \end{aligned} \tag{4.7}$$

と表せる。

ホップフィールドネットワークの状態変化は非同期的で、各ユニットは各々のタイミングで結合の強さで重みづけられた入力の総和をとり、その値により状態を遷移させ出力を出す。その状態変化規則式は、

$$\begin{aligned} \sum_j w_{ji}u_j + s_i - \theta_i > 0 \text{ のとき、} \quad u_i = 1 \\ \sum_j w_{ji}u_j + s_i - \theta_i = 0 \text{ のとき、状態変化なし} \\ \sum_j w_{ji}u_j + s_i - \theta_i < 0 \text{ のとき、} \quad u_i = 0 \end{aligned} \tag{4.8}$$

となる。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 をしきい値とする。

また、ネットワーク上の全ユニットの状態をまとめて見たものを「ネットワークの状態」と呼ぶ。ホップフィールドネットワークではこのネットワークの状態という概念を用いて、主に二つの方法により、情報処理＝パターン変換を行う。1 つ目が、入力パターンを初期状態とし、ネットワークを動作させ平衡状態で現れたパターンを出力とみなすものである。2 つ目が、ネットワークを入力部分と出力部分、そのほかの部分に分割し、入力部分を固定、そのほかの部分をランダムに設定したときのパターンから平衡状態に陥った時の出力部分を出力とするものである。

ネットワークのエネルギー　ホップフィールドネットワークの場合、平衡状態は結合荷重によって決まり、一般に複数ある平衡状態の中からどの平衡状態に陥るかは、初期状態によって決まる。このネットワークの平衡状態がホップフィールドの動作を特徴づけるものとして重要である。そこで平衡状態の性質を調べるためにネットワークのエネルギーという概念を導入する。

$$E(\alpha) = -\frac{1}{2}\sum_i\sum_j w_{ij}u_iu_j - \sum_i (s_i - \theta_i)u_i \tag{4.9}$$

式 4.8 を状態 α のときのネットワークのエネルギーと定義する。ここで、あるユニット u_i が状態変化規則に従い、状態変化したときの、上式のエネルギー関数の変化量を求める。ま

ず、u_i が 1 から 0 に変化したとき、エネルギー関数の変化量は、

$$\Delta_i E(\alpha) = \sum_j w_{ij} u_j + s_i - \theta_i \tag{4.10}$$

となる。また、u_i が 1 から 0 に変化する場合の状態変化規則式は、

$$\sum_j w_{ji} u_j + s_i - \theta_i < 0 \tag{4.11}$$

であった。この時、$\Delta_i E(\alpha)$ と上式の左辺は等しい。つまり、$\Delta_i E(\alpha)$ が負の値をとるため、このネットワークのエネルギーは減少したことになる。同様に、u_i が 0 から 1 に変化したとき、エネルギー関数の変化量は、

$$\Delta_i E(\alpha) = -\sum_j w_{ij} u_j - s_i + \theta_i \tag{4.12}$$

であり、状態変化規則式は、

$$\sum_j w_{ji} u_j + s_i - \theta_i > 0 \tag{4.13}$$

である。よって、u_i が 1 から 0 へ変化する場合も、0 から 1 へ変化する場合も、ネットワークのエネルギーは減少することがわかる。状態変化するたびにエネルギーが減少するため、どのユニットを変化させてもエネルギーが増加する場合、状態変化は起こらない。ゆえに、平衡状態とは、エネルギーの値が極小値をとるような状態のことを指す。

（※文責: 中谷奨吾）

ボルツマンマシン

ホップフィールドネットワークでは状態変化を繰り返した後, 想起したときにその後の動作が止まってしまうという問題点があった。1983 年にボルツマンマシンというものが開発された。これは Hopfield Network のメカニズムの動作を確率的にしたものである。入力の総和 i が求まるとその値に基づく次式の確率でユニットの新しい状態を 1 に設定する。

$$p(u_i = 1) = \frac{1}{1+exp(\frac{-i}{T})} \tag{4.14}$$

と定義する。(図 4.3) 今 T はネットワーク温度で $T > 0$ とする。

T →∞のときは一定状態に落ち着きにくく (図 4.5)、T → 0 のときステップ関数に近づき,Hopfield Network と同じ動作をする (図 4.4)。ボルツマンマシンでも, 確率的な意味での平衡状態を考えることができる。つまり, マシンを動かし始めてから十分時間がたった後は滞在確率の分布がユニットの結合関係によって定まるほぼ安定したものになる（マルコフ過程の平衡分布）

平衡状態は以下のようになる。いま 2 つの状態 α , β があって, 一度の状態変化で移り合えるとする。すなわち, α , β は一つのユニットの状態だけが異なる状態のペアである。それぞれの状態のネットワークエネルギーを E(α),E(β) とすると、 α から β への状態変化する確率（遷移確率）は

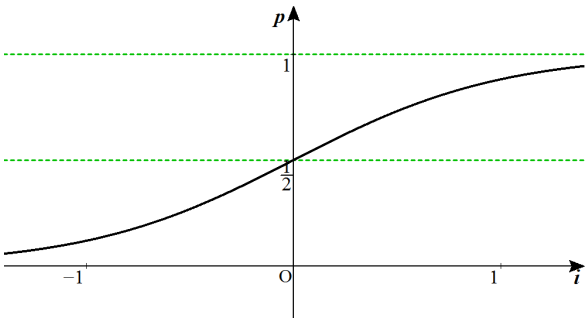

図 4.3　p のグラフ

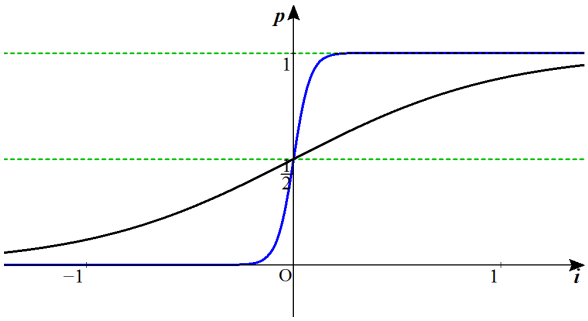

図 4.4　p のグラフ (T → 0)

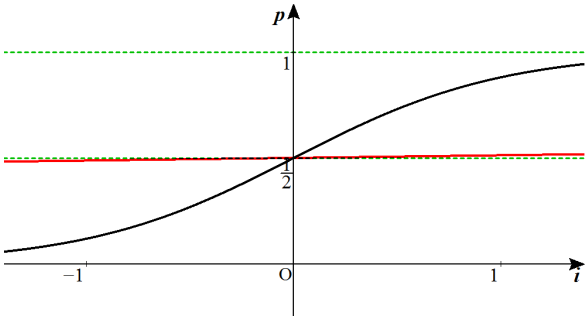

図 4.5　p のグラフ (T →∞)

状態変化規則から

$$p_{\alpha\,\beta} = \frac{1}{1+exp(-\frac{(E(\alpha)-E(\beta))}{T})} \tag{4.15}$$

となる

いま、集団の中で状態 α , β にあるネットワークの個数をそれぞれ n_α, n_β とする

$$n_\alpha p_{\alpha\,\beta} = n_\beta p_{\beta\,\alpha}$$

が成り立つような n_α, n_β が平衡分布に対応する。

従って, 平衡においては　$\frac{n_\alpha}{n_\beta} = \frac{p_{\beta\,\alpha}}{p_{\alpha\,\beta}} = exp(-\frac{(E(\alpha)-E(\beta))}{T})$

が成り立つ。従って,

$$\frac{p_\alpha}{p_\beta} = exp(-\frac{(E(\alpha)-E(\beta))}{T})$$

となり, 最後に正規化を行う,

$$p_\alpha = p_\beta exp(-\frac{E_\alpha}{T})exp(\frac{E_\beta}{T})$$

ここで、

$$p_\beta exp(\frac{E_\beta}{T})$$

は平衡状態だから定数 C とおける。
また、全確率 1　すなわち

$$\sum_\alpha p_\alpha = 1$$

だから

$$\sum_\alpha p_\alpha = C\sum_\alpha exp(\frac{-E_\alpha}{T}) = 1$$

$$p_\alpha = \frac{1}{Z}exp(\frac{-E(\alpha)}{T}) \quad Z \text{ は確率の正規化のための定数} \tag{4.17}$$

となる。このエネルギーの符号を変えたものが Boltzmann 分布と呼ばれる。この分布からわかることは以下の 2 点である。
1) エネルギーの小さい状態ほど大きな確率が現れる。
2) ネットワーク温度が小さいほど, エネルギーの差が出現確率の差に大きな影響を与える。

ボルツマンマシンのホップフィールドネットワークと大きな違いは、ホップフィールドネットワークでは正解ではないパターンで状態更新がそれ以上起こらなくなる場合がある。つまり、ホップフィールドネットワークは状態更新が一度止まると、局所解 (正解ではない解) に陥ってしまい、最適解までたどり着けない可能性がある。一方で、ボルツマンマシンは確率的動作なので、動作が止まらずに状態は更新を続け, 最適解 (正解) へとたどりつける可能性を広げるのである。

（※文責: 大場和貴）

4.2　ネットワークの学習

ネットワークの学習とは各ユニット間の結合の重みを望ましい情報処理を行うように修正していくことである. 望ましい情報処理とは、例えばあるデータを入力としてネットワークに与えた時にこちらが想定した正解 (教師信号という) と同じ結果をネットワークが出力するということである. このような学習を教師付き学習ともいう. そこで今回本プロジェクトが用いたネットワークの学習

方法であるバックプロパゲーションのアルゴリズムを説明する.
m 層のネットワークを考える. k 層の第 a ユニットへの入力の総和を ${i^k}_a$, 出力を ${o^k}_a$ とし、$k-1$ 層の第 a ユニットから k 層の第 b ユニットへの結合の重みを $w_{a\,b}^{k-1\,k}$ とする. y_b は教師信号である. 入出力関数を f とすると、

$$ {o^k}_i = f({i^k}_i) \tag{4.18} $$

$$ {i^k}_j = \sum_i = w_{a\,b}^{k-1\,k} {o^{k-1}}_a \tag{4.19} $$

という関係が成り立つ. 結合の重みを修正するために以下の 3 式を用いる.(文献 [1])

$$ \Delta {w^{k-1\,k}}_{a\,b} = -\epsilon {d^k}_b {o^{k-1}}_a \tag{4.20} $$

$$ {d^m}_b = 2({o^m}_b - y_b) f'({i^m}_b) \tag{4.21} $$

$$ {d^k}_b = (\sum_i w_{b\,a}^{k\,k+1} {d^{k+1}}_a) f'({i^k}_b) \tag{4.22} $$

ここで ϵ は小さな正の定数である.

f は

$$ f(x) = \frac{1}{1 + exp(-x)} \tag{4.23} $$

とすると、

$$ f'(x) = f(x)(1 - f(x)) \tag{4.24} $$

となるので

$$ f'({i^k}_j) = {o^k}_j (1 - {o^k}_j) \tag{4.25} $$

となる.
上で得られた w の修正量である Δw を使って、

$$ w = w + \Delta w \tag{4.26} $$

に従って結合の重みを更新していく.

（※文責: 尾ヶ瀬拓哉）

4.3 表情認識

文献 [6] では基本 6 感情をニューラルネットワークに識別させる実験を行っているが今回本プロジェクトでは笑顔、怒り、驚きの三つの表情をニューラルネットワークに識別させることを第 2 の課題とした. 以下でニューラルネットワークによる表情識別の方法を説明する.

今回、使用するニューラルネットワークには入力層のユニット 11 個, 中間層のユニット 100 個, 出力層のユニット 3 個の 3 層のバックプロパゲー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数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表情認識も高精度に行うことができると考えられる.

次に、顔データの作成方法について説明する．まず被験者 1 人 1 人の真顔、笑顔、怒り、驚きの表情の写真を撮り全ての表情の特徴点を手動で抽出した (図 4.6)．この時の特徴点は processing で取得した．付録 B に processing を用いて人の顔の特徴点をとるプログラムのソースコードを示す．

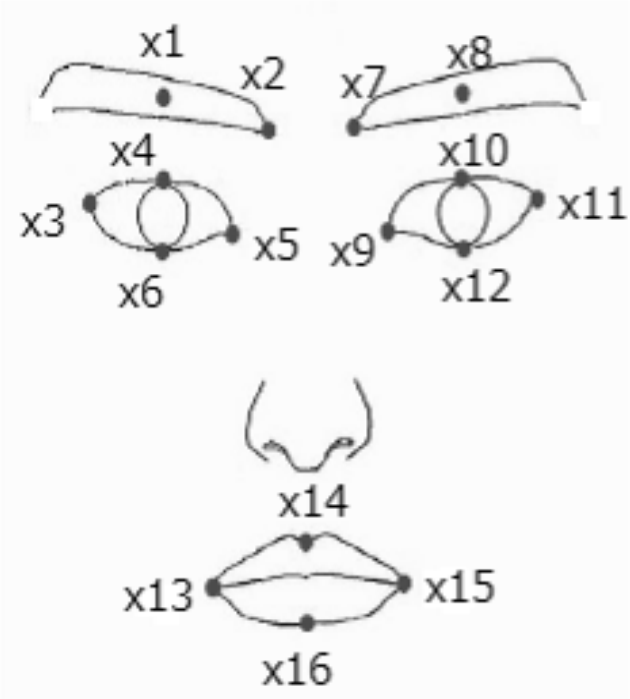

図 4.6　顔の特徴点

文献 [6] では抽出した特徴点から位置情報と呼ばれる情報と形状情報と呼ばれる情報を作成している．位置情報は特徴点を $X-Y$ 座標系の 2 次元座標で表す情報であり、形状情報は特徴点を $X-Y$ 座標系の 2 次元座標で表した情報を組み合わせ加工し、3 部位 (眉、目、口) の形状を表す情報である。文献 [6] では位置情報を用いての表情の識別も形状情報を用いての表情の識別もどちらも高精度に行えていた。よって本プロジェクトでは位置情報と形状情報のどちらを使用しても結果に差はでないと考えたので、形状情報を用いて実験を行うことにした。

眉、目、口の形状情報を求めるために、まず表 4.1 のように真顔の特徴点間の距離を計算する。

表 4.1　特徴点間の距離

1	x3 と x11 の距離
2	x4 と x6 の距離
3	x4 と x5 の距離
4	x6 と x5 の距離
5	x2 と x7 の距離
6	x1 と x4 の距離
7	x14 と x 16 の距離
8	x13 と x15 の距離
9	x13 と x14 の距離
10	x13 と x6 の距離

また表 4.1 で得られた距離を被験者の顔と PC のカメラとの距離によって生じる撮影された顔の大きさの違いを修正するために、どんな表情でも変化しないと考えられる左右の目頭の距離 (表 4.1 の 1) で割る.

例えば図 4.7 と図 4.8 のようにカメラと顔の距離が違うと同じ人でも特徴点間の距離を計算すると異なる結果が出てしまう. そこでいつも変わらない目頭間の距離で割ることによって、特徴点間の距離を目頭間の距離との比に直すことができる. そうすると顔とカメラとの距離が異なってしまっても正確な情報を作成することができる.

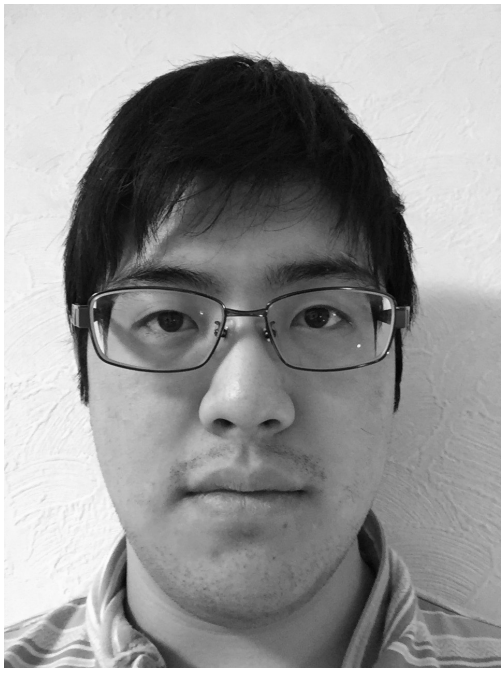

図 4.7　PC とカメラの距離が近いとき

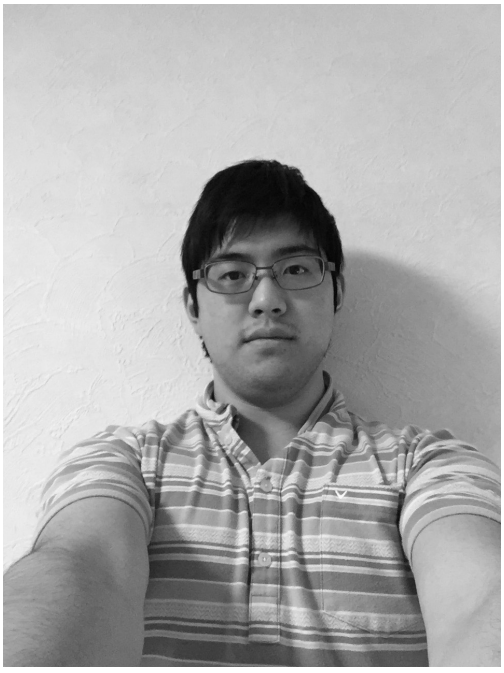

図 4.8　PC とカメラの距離が遠いとき

そして得られた真顔と笑顔のそれぞれの距離の差の絶対値を求め、それをニューラルネットワークの 10 個の入力とする (もう一つのユニットはしきい値用に常に 1 を入力として与える). 次ページ表 4.2 に笑顔のデータの例を示す.

表 4.2　笑顔の入力データの例

1	1
2	0.0
3	0.07479795
4	0.0026318133
5	0.0013212115
6	0.0152229965
7	0.014583319
8	0.12277149
9	0.14043921
10	0.054261774
11	0.13387135

なお、この入力データを作成する作業は前述の processing を用いた特徴点抽出のプログラムに組み込み、特徴点の抽出と同時に行えるようにした.

出力層のユニット数は怒り、笑顔、驚きの 3 表情を対応させて 3 つとし便宜上、上から怒り、笑顔、驚きを表すユニットとする. ニューラルネットワークの学習を行うときの教師信号の与え方は学習させるデータが表す表情と同じ表情を表すユニットのみを 1 とし、それ以外のユニットを 0 にする. 例を挙げると、笑顔のデータを入力層に与えた時は教師信号は上から (0,1,0) とする.

そして 15 人分の怒り、笑顔、驚きの入力データを作成したあと、ニューラルネットワークに学習させる. そこで得られたユニット間の結合の重みを用いて、実際に 15 人分の表情識別の実験を行った. ここの 15 人は学習に用いた 15 人分以外のデータを用いた. その実験結果の例を表 4.3 に示す.

表 4.3　笑顔を識別した時の例

出力層	出力
怒り	0.00000
笑顔	0.99939
驚き	0.0133071

（※文責: 尾ヶ瀬拓哉）

4.4　表情識別における顔部位の重要性

4.4.1　先行研究

本プロジェクトではニューラルネットワークによる表情識別の際に重要となる顔の部位を調べることを第 2 の課題とした. 文献 [7] で実際に人が表情を識別する際に顔のどの部位が重要であるかが調べられている. そこでの実験の内容を説明する.

文献 [7] では表情認知において顔の上部と下部のどちらが手がかりとなりやすいか、それは顔全体とどのように異なるのか、また表情によって異なるのかを検討し、表情認知における顔の部位の

相対的重要性を論じている.

次に文献 [7] での実験の方法について説明する. 文献 [7] では実験被験者に顔の上部と下部に異なる表情を合成した顔画像を見せて基本 6 感情の表出の程度をそれぞれ 7 段階で評定してもらう. 非常によく表している場合は 7 を、全く表していない場合は 1 になるように数字を記入してもらう. また、ここの上部と下部が異なる顔の画像は片方が基本 6 感情を表す表情でもう片方は真顔の表情である. 例えば顔の上部が怒りで下部が真顔の表情の画像などである.

結果は、怒り、悲しみ、驚きの表情では顔の上部の影響が強く、恐怖、喜びの表情では下部の影響が強いことが認められ、表情によって影響の強い部位が異なることが示されていた.

4.4.2 本プロジェクトにおける顔部位の重要性を調べる実験

4.4.1 節の顔部位の重要性の先行研究を受けて、本プロジェクトではこの先行研究の実験をニューラルネットワークを用いて行ってみようと考えた. 具体的な方法としては、まず 4.3 節でのニューラルネットワークを用いての表情の識別実験での各表情の識別率を求める. その後入力データの顔の上部に関する部分と下部に関する部分を 0 にしたデータを先行研究における異なる表情を合成した画像とみなし、このデータで表情認識のテストを行い識別率を求める. そして先の実験との識別率を比較する. 表 4.3 と表 4.4 にそれぞれ顔の上部が笑顔のデータと下部が笑顔のデータを示す.

表 4.4 顔の上部が笑顔で下部が真顔のデータ

1	1
2	0.0
3	0.07479795
4	0.0026318133
5	0.0013212115
6	0.0152229965
7	0.014583319
8	0.0
9	0.0
10	0.0
11	0.0

表 4.5　顔の上部が真顔で下部が笑顔のデータ

1	1
2	0.0
3	0.0
4	0.0
5	0.0
6	0.0
7	0.0
8	0.12277149
9	0.14043921
10	0.054261774
11	0.13387135

もし表 4.4 のような顔の上部が笑顔で下部が真顔のデータを入力として与えたときの識別率がもし普通のデータを与えたときの識別率よりも下がった場合は、ニューラルネットワークによる笑顔の識別には顔の下部が重要ということになる. また表 4.5 のような顔の上部が真顔で下部が笑顔のデータを入力として与えた時の識別率が普通のデータを入力として与えたときの識別率よりも下がった場合は、逆に笑顔の識別には顔の上部が重要であるということになる. この実験を怒り、笑顔、驚きの 3 表情で行った.

（※文責: 尾ヶ瀬拓哉）

4.5　期末発表の際のデモプログラムの概要

本プロジェクトでは期末発表の際、Kinect(V1) を使用したリアルタイムで表情認識を行うデモンストレーションプログラム（main.cpp 参照）を使った。このプログラムは C++ を使用しており、また外部ライブラリとして Kinect for Windows と OpenCV という画像処理のオープンソースのライブラリを使用した。またニューラルネットワークの動作に関しては、C ＃を使用して書かれたバックプロパゲーションのプログラム (Program.c) を利用している。

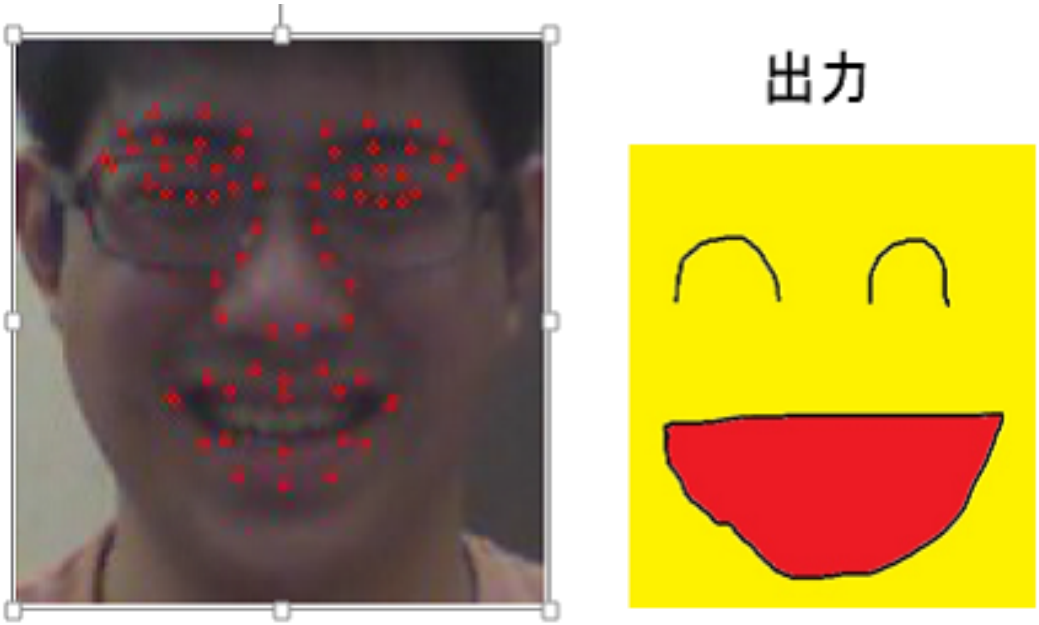

図 4.9　デモプログラム実行の様子

このプログラムは、Kinect の FaceTrackLib という顔の認証と顔の特徴点を抽出する機能を利用している（KinectSensor.h)。このライブラリ内の kinect.get2DPointCount という関数を利用することにより顔の特徴点を 118 個リアルタイムで取得することができる。また今回は Kinect V1 を使用したが、Kinect V2 を使用すれば頬の動きなども取得できる全特徴点 1347 個を取得できる。プログラムの流れは、まず初めに Kinect カメラに向かって真顔の状態でスペースキーを押し、基準となる真顔のデータを保存する。次に笑顔・怒り・驚きのいずれかの表情をカメラに向けてキーボードの 1 キーを押し、現表情のデータを取得する。表情のデータを取得すれば、そのデータをバックプロパゲーションのプログラム (program.c) に送り自動で出力が帰ってくる。その帰ってきた値を元に笑顔・怒り・驚きのいずれかの画像を出力するという流れになっている。

、

（※文責: 斉藤健士郎）

4.6　各人の担当課題の概要と役割

各プロジェクトメンバーの 4 章で示した課題解決のプロセスにおける担当した部分を以下で説明する.

- 斉藤健士郎
  - 4.3 節での特徴点抽出のための processing によるプログラミング
  - 最終発表時に使用したデモの作成
- 尾ヶ瀬拓哉
  - 4.2 節でのバックプロパゲーションモデルのニューラルネットワークの実装
  - 4.3 節の表情識別の実験
  - 4.4 節の顔部位の重要性の実験
- 藤島真央

  - 4.3 節の表情識別の実験
  - 4.4 節の顔部位の重要性の実験
- 中谷奨吾
  - 4.2 節でのバックプロパゲーションモデルのニューラルネットワークの実装
- 齋藤達哉
  - 4.3 節の表情識別の実験
  - 4.4 節の顔部位の重要性の実験
- 大場和貴
  - 4.2 節でのバックプロパゲーションモデルのニューラルネットワークの実装

上記以外にも、プロジェクトメンバー全員でニューラルネットワークの基本的な部分の学習をし、さらに文献 [6] や [7] を読んで理解した.

（※文責: 尾ヶ瀬拓哉）

第 5 章　結果

5.1　プロジェクトの結果

5.1.1　前期の活動結果

前期では，プロジェクトの成果を出すための準備としてニューラルネットワークの数理と学習に関して理解を深めた．

その結果，ニューラルネットワークには大きく分けて階層型ネットワークモデルと相互結合型ネットワークモデルの二つに分類されていることが分かった．また，階層型ネットワークモデルの代表的なモデルとしてパーセプトロンが，相互結合型ネットワークの代表的なモデルとしてホップフィールドネットワークとボルツマンマシンがあり，それぞれのモデルについて数式の意味や法則性について理解できた．このホップフィールドネットワークとボルツマンマシンに関してそれぞれプログラムを作成し，双方のモデルの相違点を知ることができた．例えば，ホップフィールドネットワークについて，ニューロンの内部状態変化は決定論的なものであることが分かり，ボルツマンマシンについて，ニューロンの内部状態変化は確率的なものであるかということが分かった．

（※文責: 藤島真央）

5.1.2　後期の活動での検証結果

後期では，前期で勉強したニューラルネットワークを用いて人の認知過程について調べた．本プロジェクトでは，認知過程の中でも特に表情の識別に注目し，目や口などの顔の一部分の情報を取らない場合，表情の識別に影響するか検証を行った．

その結果，被験者全員の表情のデータをそれぞれ入力した場合に，ニューラルネットワークが正しく表情を 識別できた数を被験者の総数で割った値を識別率として定義すると，特徴点をすべて取った場合の各表情の識別率は驚きが 93 %，喜びが 87 %，怒りが 80 %，目の特徴点を取らない場合の各表情の識別率はは驚きが 93 %，喜びが 80 %，怒りが 47 %，口の特徴点を取ない場合の各表情の識別率はは驚きが 20 %，喜びが 27 %，怒りが 80 %となった．図 5.1 および表 5.1 は，特徴点をすべて取った場合，目の特徴点を取らない場合，口の特徴点を取らない場合における驚き，喜び，怒りの表情の識別率をそれぞれ表したものである．特徴点をすべて取った場合は驚き，喜び，怒りの表情をそれぞれ識別できたが，目の特徴点を取らない場合には怒りの識別率が，口の特徴点を取らない場合には驚きと喜びの識別率が大きく減少した．

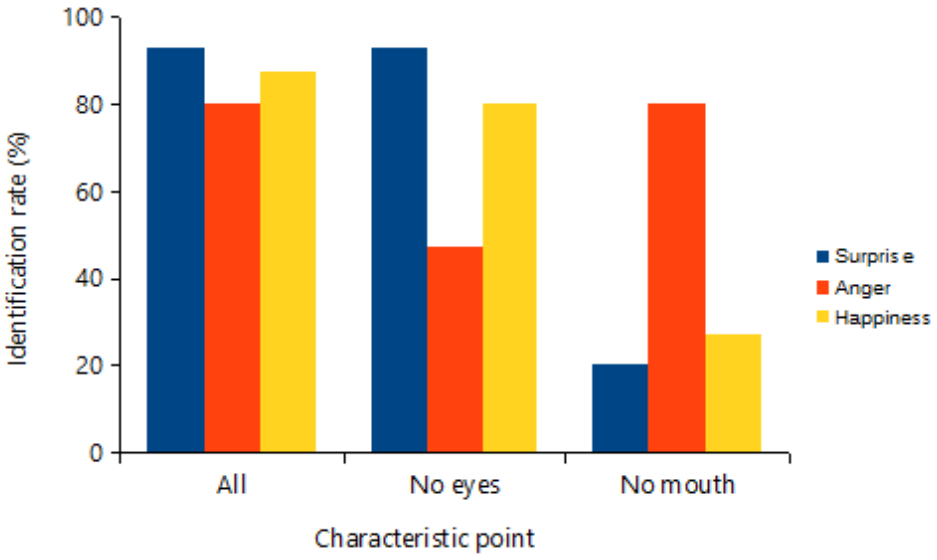

図 5.1　各表情の識別率

表 5.1　各表情の識別率

	Surprise	Anger	Happiness
All	93%	80%	87%
No eyes	93%	47%	80%
NO mouth	20%	80%	27%

このことから，怒りの識別には目の情報が重要であり，喜びや驚きの識別には口の情報が重要であるということ，すなわち表情ごとに重視される顔の部位の情報は異なることが分かった．驚きや喜びの表情が口の情報を重視するのは，口を大きく開いたり，口角が上がったりするためえあると考えられる．これに対して怒りの表情が目の情報を重視するのは，眉間に皺が寄ることによって目に力が入るためであると考えられる．

（※文責: 藤島真央）

5.1.3　先行研究との比較

本プロジェクトでは，ニューラルネットワークを用いて表情の識別にを行った文献 [6] の研究事例と，顔の部位が表情の識別にどのように影響するのか認知実験を行った文献 [7] の研究事例のそれぞれの特徴を踏まえて，顔の部位が表情の識別にどのように影響するのかニューラルネットワークを用いた検証を行った．その結果，「表情認知における顔部位の相対的重要性」においては驚きの表情は顔の上部が重要であるということが示されていたが、今回本プロジェクトでは驚きの表情は下部のほうが重要であるという結果が出た．本プロジェクトは前述の通り驚きの表情は口が大きく動くため口のほうが重要であるという考えだが、驚きの表情は口と同時に目も大きく開くことが多いので、どちらがより重要であるか更に調べる必要がある．

（※文責: 藤島真央）

5.2 成果の評価

5.2.1 前期の活動成果に関しての評価

前期では，ニューラルネットワークの数理と学習を理解するするための具体的な方法として，ニューラルネットワークのテキストを用いてプロジェクトメンバー全員で 分担して輪読を行った．また，ホップフィールドネットワークとボルツマンマシンの二つのモデルに関して簡単なプログラムを作成した．プログラムを動作させて視覚的にもそれぞれのネットワークモデルの仕組みについて理解しやすいようにするなどして，ニューラルネットワークに対する勉強方法を工夫した．十分な理解をすることを重視したため，学習の進捗状況は予定としていた相互結合型ネットワークの学習手法の内容までは勉強が進まなかったが， 相互結合型ネットワークモデルの数理の部分に関しては，状態変化規則や数式の意味，構造や動作の違いなど，十分な理解をすることができた．階層型ネットワークに関しては，数理と学習手法の両方を勉強し，理解することができた．

中間発表では，ニューラルネットワークの基本的な部分に関して十分間で 理解できるように図やグラフを用いて 説明を行った．しかし，発表評価シートの結果を見てみると，大半の学生は十分な理解を出来なかったとの回答が得られた．

夏季休暇期間においても認知心理学の本をプロジェクトメンバーで分担して読み，人の認知過程の研究事例に関して学習することができた．

（※文責: 藤島真央）

5.2.2 後期の活動成果に関しての評価

後期では，ニューラルネットワークの学習手法のひとつであるバックプロパゲーションを用いて表情を識別する装置の作成をプロジェクトメンバーで協力して行った．表情の識別の検証を行った結果，目の存在は重要であることを示唆する結果が得られたが，目は怒りの 1 表情分，口は驚きと喜びの 2 表情分であることから，諺のように目は口ほどにものを言う存在にはならないようである．後期の活動開始当初は，表情の識別について活動を行うことは決まっていたが，具体的な方針は定まっておらず，思うように活動は出来なかった．明確な活動方針が定まってからは役割を分担して活動を行うことができた．

後期末のプロジェクトの成果発表では，中間発表での発表評価シートに基づいて，複雑な式は省き，図や表を重視した，理解しやすい発表スライドとポスターの作成を行うことができた．同様に，発表評価シートから発表技術を改善することができた．

（※文責: 藤島真央）

5.2.3 総評

プロジェクトの活動の一年を通して，ニューラルネットワークに関する勉強会の進捗状況が悪く，ニューラルネットワークを用いた認知過程の実験の方針が明確に定まらず，プロジェクト開始当初に立てた活動計画の通りに実行することが出来なかった．特に，方針が明確に定まらないままプロジェクトが進行していたため，プロジェクトメンバーで分担した作業に関しては各々従事していたが，後期が始まってから約 2 ヶ月の間は納得のいく成果を得ることが出来なかった．

成果発表会に関しては，発表評価シートの反映および図や表の差分の使用によって，発表資料に関して後期の発表資料は前期の発表資料に比べて質の向上が見られた．しかし，発表技術に関してはどちらの発表会でも声量が小さかったり，ずっとスライドやパソコンの方を見ながら発表していたりと，発表技術の練習不足が目立ってしまった．

プロジェクトメンバーが 6 人と他のグループに比べるとやや少なく，メンバー一人当たりの作業の負担が懸念されたが，各メンバー同士で連携することによってプロジェクトを進行することができた．また，プログラミングやニューラルネットワークに関しての技術や理解度がメンバー間で差が生じていたが， プロジェクトメンバーで教えあって協力し，作業を分担することで作業の遅れを取り戻し，成果物が得ることができた．

（※文責: 藤島真央）

第 6 章　今後の課題と展望

6.1　今後の課題

本プロジェクトでは，顔の部位が表情の識別にどのように影響するのかニューラルネットワークを用いた検証を行った．今回は識別する表情が 3 表情であったため 5 章で述べたような結果となったが，識別する表情を恐怖，嫌悪，悲しみを加えた基本 6 表情とした場合には，目の情報を重視する表情が増え，「目は口ほどにものを言う」といった諺通りの存在になる可能性がある．驚き，喜び，怒りの 3 表情に関しても，表情の変化には個人差があるため，さまざまな表情を識別できるようにニューラルネットワークに学習に用いるデータの量を増やす必要がある．また，今回の検証では男子学生 15 名が被験者であったが，年齢，人種，性別によっては異なる結果を得られていた可能性もある．他にも識別率の精度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が必要である．

今回の検証では，顔の中でも特に変化が顕著である眉，目，口の 3 つの部位の特徴点を手動で抽出し，顔の上部（眉，目）の特徴量と顔の下部（口）の特徴量から各表情の識別率をそれぞれ比較するものであったが，顔の部位が表情の識別にどのように影響するのかを検証するのであれば，眉と目の特徴量も別にして検証を行う必要があり，これらを今後の検討すべき課題とする．

（※文責: 藤島真央）

6.2　展望

表情を識別する技術を応用した装置は既にいくつか存在している．例えば，文献 [8] では，顔面上を伝播する表面筋電位から得られる生体信号のパターンから表情を識別し，LED や振動デバイス通して表情の識別結果を提示することによって表情によるコミュニケーションが困難な人へ表情の知覚の支援の実現をしている．ソフトバンクグループが開発したコミュニケーションロボット「Pepper」では，人の表情や声を識別し，その人の今の感情を推測する機能が実現しており，その時の表情に応じた行動をすることが可能である．ソニーでは顔の部位から笑顔とその強弱（微笑みや大笑いなど，笑顔の程度）を検出し，適切なタイミングで撮影を行う「スマイルシャッター」機能を実現している．

今後，自動での表情の識別を高い精度で実現する技術は前提とした装置の開発や研究が当然のことながら求められる．

（※文責: 藤島真央）

付録 A　相互評価

- 斉藤健士郎
  - プロジェクトリーダーとして皆をまとめてくれて助かった。具体的にはスケジュール管理等をしてくれた。
  - 主にプログラム作成で活躍してくれた。
- 尾ヶ瀬拓哉
  - ニューラルネットワークの内容を一番理解していて、輪読の時大変助かった。
  - 中間発表の時に質問の受け答えをほとんどしてくれて助かった。
- 藤島真央
  - 中間発表用のポスターの作成を一人で手掛けてくれて非常に助かった。
  - スライド作成のとき、大半を作成してくれとても良いものを作ってくれた。
- 中谷奨吾
  - 色々な意見を出してくれ、自分が気づかなかったことに気付かせてくれた。
  - 輪読で、難しい所を頑張ってくれた。
- 齋藤達哉
  - 輪読で、新しい章に入る難しいところをよく頑張ってくれた。
  - 補助スライドの作成等、色々なところで頑張ってくれた。
- 大場和貴
  - 輪読の出番まで行かなかったが、補助用のスライドの作成等いろいろな所で頑張ってくれた。
  - スライド作成が大変だったと思うが頑張ってくれた。

付録 B　ソースコード

KinectSensor.h

```
#pragma once
// NuiApi.h の前に Windows.h をインクルードする
#include <Windows.h>
#include <NuiApi.h>
#include <FaceTrackLib.h>

#include <iostream>
#include <sstream>
#include <vector>



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

const NUI_IMAGE_RESOLUTION CAMERA_RESOLUTION = NUI_IMAGE_RESOLUTION_640x480;

class KinectSensor
{
private:

    INuiSensor* kinect;
    HANDLE imageStreamHandle;
    HANDLE depthStreamHandle;
    HANDLE streamEvent;

    DWORD width;
    DWORD height;

    IFTFaceTracker* pFT;                // 顔追跡する人
    FT_CAMERA_CONFIG videoCameraConfig; // RGB カメラの設定
    FT_CAMERA_CONFIG depthCameraConfig; // 距離カメラの設定
```

```
    IFTResult* pFTResult;               // 顔追跡結果

    IFTImage* pColorFrame;              // 顔追跡用の RGB データ
    IFTImage* pDepthFrame;              // 顔追跡用の距離データ

    FT_SENSOR_DATA sensorData;          // Kinect センサーデータ

    std::vector<unsigned char> colorCameraFrameBuffer;
    std::vector<unsigned char> depthCameraFrameBuffer;

    bool isFaceTracked;
    RECT faceRect;

public:


    KinectSensor()
        : pFT( 0 )
        , pFTResult( 0 )
        , isFaceTracked( false )
    {
    }

    ~KinectSensor()
    {
        // 終了処理
        if ( kinect != 0 ) {
            kinect->NuiShutdown();
            kinect->Release();

            pFTResult->Release();
            pColorFrame->Release();
            pDepthFrame->Release();
            pFT->Release();
        }
    }

    void initialize()
    {
        createInstance();

        // Kinect の設定を初期化する
```

```
        ERROR_CHECK( kinect->NuiInitialize(
            NUI_INITIALIZE_FLAG_USES_COLOR |
            NUI_INITIALIZE_FLAG_USES_DEPTH_AND_PLAYER_INDEX |
            NUI_INITIALIZE_FLAG_USES_SKELETON ) );

        // RGB カメラを初期化する
        ERROR_CHECK( kinect->NuiImageStreamOpen( NUI_IMAGE_TYPE_COLOR, CAMERA_RESOLUTION,
            0, 2, 0, &imageStreamHandle ) );

        // 距離カメラを初期化する
        ERROR_CHECK( kinect->NuiImageStreamOpen( NUI_IMAGE_TYPE_DEPTH_AND_PLAYER_INDEX,
CAMERA_RESOLUTION,  0, 2, 0, &depthStreamHandle ) );

        // Near モード
        ERROR_CHECK( kinect->NuiImageStreamSetImageFrameFlags(
            depthStreamHandle, NUI_IMAGE_STREAM_FLAG_ENABLE_NEAR_MODE ) );


        // スケルトンを初期化する
        ERROR_CHECK( kinect->NuiSkeletonTrackingEnable( 0,
            NUI_SKELETON_TRACKING_FLAG_ENABLE_IN_NEAR_RANGE |
            NUI_SKELETON_TRACKING_FLAG_ENABLE_SEATED_SUPPORT ) );

        // フレーム更新イベントのハンドルを作成する
        streamEvent = ::CreateEvent( 0, TRUE, FALSE, 0 );
        ERROR_CHECK( kinect->NuiSetFrameEndEvent( streamEvent, 0 ) );

        // 指定した解像度の、画面サイズを取得する
        ::NuiImageResolutionToSize(CAMERA_RESOLUTION, width, height );

        // 顔認識の初期化
        initializeFaceTracker();
    }

    void initializeFaceTracker()
    {
        // FaceTracker のインスタンスを生成する
        pFT = ::FTCreateFaceTracker();
        if( !pFT ) {
            throw std::runtime_error( "ERROR:FTCreateFaceTracker" );
        }
```

```
        // RGB カメラおよび距離カメラの設定を行う
        videoCameraConfig.Width = width;
        videoCameraConfig.Height = height;
        videoCameraConfig.FocalLength =
                    NUI_CAMERA_COLOR_NOMINAL_FOCAL_LENGTH_IN_PIXELS * (width / 640.0f);

        depthCameraConfig.Width = width;
        depthCameraConfig.Height = height;
        depthCameraConfig.FocalLength =
                    NUI_CAMERA_DEPTH_NOMINAL_FOCAL_LENGTH_IN_PIXELS * (width / 320.0f);

        // FaceTracker を初期化する
        HRESULT hr = pFT->Initialize( &videoCameraConfig, &depthCameraConfig, 0, 0) ;
        if( FAILED(hr) ) {
            throw std::runtime_error( "ERROR:Initialize" );
        }

        // FaceTracker の結果を格納先を生成する
        hr = pFT->CreateFTResult( &pFTResult );
        if(FAILED(hr)) {
            throw std::runtime_error( "ERROR:CreateFTResult" );
        }

        // FaceTracker で利用する RGB および距離データのインスタンスを生成する
        pColorFrame = FTCreateImage();
        pDepthFrame = FTCreateImage();
        if( !pColorFrame || !pDepthFrame ) {
            throw std::runtime_error( "ERROR:FTCreateImage" );
        }

        // RGB および距離データのバッファサイズを設定する
        // RGB は 1pixel あたり 4 バイト。距離は 1pixel あたり 2 バイト
        colorCameraFrameBuffer.resize( width*4 * height );
        depthCameraFrameBuffer.resize( width*2 * height );

        // フレームデータにバッファを設定する
        pColorFrame->Attach(width, height, &colorCameraFrameBuffer[0],
FTIMAGEFORMAT_UINT8_B8G8R8X8, width*4);
        pDepthFrame->Attach(width, height, &depthCameraFrameBuffer[0],
FTIMAGEFORMAT_UINT16_D13P3, width*2);

        // センサーデータを作成する
```

```
        sensorData.pVideoFrame = pColorFrame;
        sensorData.pDepthFrame = pDepthFrame;
        sensorData.ZoomFactor = 1.0f; //  1.0
        sensorData.ViewOffset.x = 0; //(0,0)
        sensorData.ViewOffset.y = 0; //(0,0)
    }

    void update()
    {
        // データの更新を待つ
        DWORD ret = ::WaitForSingleObject( streamEvent, INFINITE );
        ::ResetEvent( streamEvent );

        copyStreamData( imageStreamHandle, colorCameraFrameBuffer );
        copyStreamData( depthStreamHandle, depthCameraFrameBuffer );
        faceTracking();
    }

    const std::vector<unsigned char>& getColorFrameData() const
    {
        return colorCameraFrameBuffer;
    }

    const std::vector<unsigned char>& getDepthFrameData() const
    {
        return depthCameraFrameBuffer;
    }

    bool IsFaceTracked() const
    {
        return isFaceTracked;
    }

    const RECT& FaceRect()
    {
        return faceRect;
    }

    DWORD getWidth() const
    {
        return width;
    }
```

```
    DWORD getHeight() const
    {
        return height;
    }

private:

    void createInstance()
    {
        // 接続されている Kinect の数を取得する
        int count = 0;
        ERROR_CHECK( ::NuiGetSensorCount( &count ) );
        if ( count == 0 ) {
            throw std::runtime_error( "Kinect を接続してください" );
        }

        // 最初の Kinect のインスタンスを作成する
        ERROR_CHECK( ::NuiCreateSensorByIndex( 0, &kinect ) );

        // Kinect の状態を取得する
        HRESULT status = kinect->NuiStatus();
        if ( status != S_OK ) {
            throw std::runtime_error( "Kinect が利用可能ではありません" );
        }
    }

    void copyStreamData( HANDLE streamHandle, std::vector<unsigned char>& buffer )
    {
        // RGB カメラのフレームデータを取得する
        NUI_IMAGE_FRAME frame = { 0 };
        ERROR_CHECK( kinect->NuiImageStreamGetNextFrame( streamHandle, INFINITE, &frame ) )

        // 画像データを取得する
        NUI_LOCKED_RECT data;
        frame.pFrameTexture->LockRect( 0, &data, 0, 0 );

        // データをコピーする
        if ( data.size != buffer.size() ) {
            buffer.resize( data.size );
        }
```

```
        std::copy( data.pBits, data.pBits + data.size, buffer.begin() );

        // フレームデータを解放する
        ERROR_CHECK( kinect->NuiImageStreamReleaseFrame( streamHandle, &frame ) );
    }

    std::vector<unsigned char> copyStreamData( HANDLE streamHandle )
    {
        std::vector<unsigned char> buffer;
        copyStreamData( streamHandle, buffer );
        return buffer;
    }

private:

    // 顔追跡 (内部用)
    HRESULT faceTracking_p()
    {
        // 追跡中、未追跡によって処理が変わる
        if(!isFaceTracked) {
            // FaceTraking を開始する
            return pFT->StartTracking(&sensorData, NULL, NULL, pFTResult);
        }
        else {
            // FaceTraking を継続する
            return pFT->ContinueTracking(&sensorData, NULL, pFTResult);
        }
    }

private:

    FT_VECTOR2D* points;
    UINT pointCount;

public:

    FT_VECTOR2D* get2DPoints()const
    {
        return points;
    }

    UINT get2DPointCount() const
```

```
    {
        return pointCount;
    }

    // 顔追跡（公開用）
    void faceTracking()
    {
        // 顔追跡
        HRESULT hr = faceTracking_p();
        // 顔を見つけたので、追跡状態へ遷移
        if(SUCCEEDED(hr) && SUCCEEDED(pFTResult->GetStatus())) {
            ::OutputDebugString( "FaceTracking Success\n" );
            isFaceTracked = true;

            // 顔の領域を取得する
            pFTResult->GetFaceRect( &faceRect );

            // 2D の座標を取得する
            points = 0;
            pointCount = 0;
            hr = pFTResult->Get2DShapePoints( &points, &pointCount );
            if( FAILED(hr) ) {
                ::OutputDebugString( "Get2DShapePoints Failed\n" );
            }
        }
        // 顔を見失ったので、未追跡状態のまま
        else {
            ::OutputDebugString( "FaceTracking failed\n" );
            isFaceTracked = false;
        }
    }
};
```

main.cpp

```
// NuiApi.h の前に Windows.h をインクルードする
#include <Windows.h>
#include <NuiApi.h>
#include <fstream>

#include "KinectSensor.h"
#include "kansu.h"
#include <opencv2/opencv.hpp>

float kao[10];//リアルタイムで特徴点を保存しておく

float magao[10];//真顔の時の特徴点を保存用
float egao[10];//表情をしている時の特徴点を保存用

//////キー動作のフラグ用
/**/bool keyon=false,keyegao=false,keykanashimi=false;
/**/bool keyreturn=false;
/**/bool magaokey=false, egaokey=false;
/**/bool key_get =false;
////////////////////////////

int ret=0;//
int file_time=0;//

//メイン文
int main()
{
  std::ofstream egao_file[100];
  egao_file[0].open("file\\egaosa.txt",std::ios::out);
  egao_file[0] << 1.0<< std::endl;
  //書き込み用ファイルを作る（開く）

    try {
        KinectSensor kinect;
        kinect.initialize();

        while ( 1 ) {
            kinect.update();
            cv::Mat image( kinect.getHeight(), kinect.getWidth(), CV_8UC4,
                           (void*)&kinect.getColorFrameData()[0] );//キネクト
の画像データをもらう
```

```
cv::Mat image_egao = cv::imread("file/kao/egao.jpg");
cv::Mat image_ikari = cv::imread("file/kao/ikari.jpg");
cv::Mat image_odoroki = cv::imread("file/kao/odoroki.jpg");

if(kinect.IsFaceTracked() ==false){//もし顔を確認できていなかったら
cv::destroyWindow("gazou");//画像をケス
}

            if ( kinect.IsFaceTracked() ) {

//ニューラルネットワークのプログラムを渡す
if(key_get)
ret = system("C:/Users/b1013152/Desktop/test/Console/ConsoleApplication1/ConsoleApplicationi



const FT_VECTOR2D* points = kinect.get2DPoints();
for ( int i = 0; i <68; ++i ) { //i<kinect.get2DPointCount() で全特徴点,68 で輪
郭なし
cv::circle( image, cv::Point(points[i].x, points[i].y), 1, cv::Scalar( 0, 0, 255 ) );
}

//顔を拡大する
cv::Mat out;
cv::resize( cv::Mat( image, cv::Rect(
cv::Point( kinect.FaceRect().left - 10, kinect.FaceRect().top - 10),
cv::Point( kinect.FaceRect().right + 10, kinect.FaceRect().bottom + 10)) ),
out, cv::Size(), 5, 5 );
//cv::imshow( "Face", out );

if(ret==3) cv::imshow("顔",image_ikari);//ここで顔みせる
if(ret==4) cv::imshow("顔",image_egao);//ここで顔みせる
if(ret==5) cv::imshow("顔",image_odoroki);//ここで顔みせる

//////////////必要な特徴点のデータの差を渡す
kao[0]=sa(points[4],points[12]);
kao[1]=sa(points[2],points[6]);
kao[2]=sa(points[2],points[0]);
kao[3]=sa(points[6],points[0]);
kao[4]=sa(points[16],points[26]);
kao[5]=sa(points[23],points[2]);
kao[6]=sa(points[51],points[57]);
```

```
kao[7]=sa(points[48],points[54]);
kao[8]=sa(points[48],points[51]);
kao[9]=sa(points[48],points[57]);
//////////////

float menagasa = kao[0];//目の間の長さを別に保存する
for(int i=0;i<10;i++)egao[i] = kao[i]/menagasa;

            }

cv::imshow("KinectSample", image );//画像を見せる


            int c = cvWaitKey( 2 );
            if (c == '\x1b') break;



/////真顔の特徴点のを計算
if ( keyon==true && GetAsyncKeyState(VK_SPACE)==false){
float menagasa = kao[0];
for(int i=0;i<10;i++)magao[i] = kao[i]/menagasa;
}keyon=false;
if ( GetAsyncKeyState(VK_SPACE) ){keyon=true;}

//表情の特徴点を計算
if ( keykanashimi==true && GetAsyncKeyState(0x31)==false){

if(egao_file[0].is_open() ==false){
file_time++;
 egao_file[file_time].open("file\\egaosa.txt",std::ios::out);
 egao_file[file_time] << 1.0<< std::endl;
}

}keykanashimi=false;
if ( GetAsyncKeyState(0x31) ){keykanashimi=true;}

key_get=false;
//真顔と各々の表情の差を計算，ファイルに出力
if ( keyreturn==true && GetAsyncKeyState(VK_RETURN)==false){

for(int i=0;i<10;i++){
```

```
egao_file[file_time] << abs(magao[i] - egao[i])<< std::endl;
}
egao_file[file_time].close(); //ファイルに２重に書き込みしないように一回で閉じる
key_get=true;

}keyreturn=false;
if ( GetAsyncKeyState(VK_RETURN) ){keyreturn=true;}

        }
    }
    catch ( std::exception& ex ) {
        std::cout << ex.what() << std::endl;
    }

}
```

ConsoleApplication1 Program.c

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
namespace NN
{
    abstract class NNBase
    {
        public int gakusyu_kosu;  //学習に使うデータの個数
        public int test_kosu;     //テストに使うデータの個数
        public int S_unit_kosu;   //S 層のユニット個数
        public int A_unit_kosu;   //A 層のユニットの個数
        public int R_unit_kosu;   //R 層のユニットの個数
        public double alpha;      //結合の修正量を求める際に使う定数
        public double eta;        //結合の修正量を求める際に使う定数

        public double[,] output_S;  //S 層の出力
        public double[] output_A;   //A 層の出力
        public double[] output_R;   //R 層の出力
        public double[,] w_S_to_A;  //S 層から A 層への重み
        public double[,] w_A_to_R;  //A 層から R 層への重み

        public double[,] deltaw_S_to_A;  //S 層から A 層への重みの修正量
        public double[,] deltaw_A_to_R;  //A 層から R 層への重みの修正量

        public double[,] teacher_signal; //教師信号

        public NNBase(int gakusyu_kosu, int test_kosu, int S_unit_kosu,
int A_unit_kosu, int R_unit_kosu, double alpha, double eta)
        {
            this.gakusyu_kosu = gakusyu_kosu;
            this.test_kosu = test_kosu;
            this.S_unit_kosu = S_unit_kosu;
            this.A_unit_kosu = A_unit_kosu;
            this.R_unit_kosu = R_unit_kosu;

            output_S = new double[gakusyu_kosu + test_kosu, S_unit_kosu];
```

```
//S 層の出力は入力の総数と S 層のユニットの個数の 2 次元配列
            output_A = new double[A_unit_kosu];
            output_R = new double[R_unit_kosu];

            w_S_to_A = new double[A_unit_kosu, S_unit_kosu];
            w_A_to_R = new double[R_unit_kosu, A_unit_kosu];

            deltaw_S_to_A = new double[A_unit_kosu, S_unit_kosu];
            deltaw_A_to_R = new double[R_unit_kosu, A_unit_kosu];

            teacher_signal = new double[gakusyu_kosu + test_kosu, R_unit_kosu];

            this.alpha = alpha;
            this.eta = eta;

        }
        public abstract void Initialize1();
        public abstract void Initialize2();
        public abstract void FP(int dataIndex);
        public abstract void BP(int dataIndex);
    }
    class NNCore : NNBase
    {
        public NNCore(int a1, int a2, int a3, int a4, int a5, double a6 = 0.9,
double a7 = 0.01) : base(a1, a2, a3, a4, a5, alpha: a6, eta: a7) { }

        public override void Initialize1()
        {
            Random Rnd = new Random();

            //S 層から A 層への重みを初期化
            for (int i = 0; i < S_unit_kosu; i++)
            {
                for (int j = 1; j < A_unit_kosu; j++)
                {
                 w_S_to_A[j, i] = (Math.Sign(Rnd.NextDouble() - 0.5) * Rnd.NextDouble()) /
                }
            }

            //A 層から R 層への重みを初期化
        for (int i = 0; i < A_unit_kosu; i++)
          {
```

```
            for (int j = 0; j < R_unit_kosu; j++)
             {
               w_A_to_R[j, i] = (Math.Sign(Rnd.NextDouble() - 0.5) * Rnd.NextDouble()) / 10;
             }
          }

      }

        public override void Initialize2()
        {
            //S 層から A 層の重み
            using (StreamReader r = new StreamReader(@"weight\P_s_to_a.txt"))
            {

                for (int i = 0; i < S_unit_kosu; i++)
                {
                    for (int j = 1; j < A_unit_kosu; j++)
                    {
                        w_S_to_A[j, i] = double.Parse(r.ReadLine());
                    }

                }
            }

            //A 層から R 層への重みを初期化
            using (StreamReader r = new StreamReader(@"weight\P_a_to_r.txt"))
            {

                for (int i = 0; i < A_unit_kosu; i++)
                {
                    for (int j = 0; j < R_unit_kosu; j++)
                    {
                        w_A_to_R[j, i] = double.Parse(r.ReadLine());
                    }

                }
            }


        }
        public override void FP(int dataIndex)
        {
```

```
            double sum = 0.0;

            //S 層から A 層への信号伝播
            for (int i = 1; i < A_unit_kosu; i++)
            {
                sum = 0.0;
                for (int j = 0; j < S_unit_kosu; j++)
                {
                    sum = sum + w_S_to_A[i, j] * output_S[dataIndex, j];
                }
                output_A[i] = sigmoid(sum);
            }
            output_A[0] = 1;
            //A 層から R 層への信号伝播
            for (int i = 0; i < R_unit_kosu; i++)
            {
                sum = 0.0;
                for (int j = 0; j < A_unit_kosu; j++)
                {
                    sum = sum + w_A_to_R[i, j] * output_A[j];
                }
                output_R[i] = sigmoid(sum);
            }

        }

        public override void BP(int dataIndex)
        {
            double sum = 0.0;
            double[] teacher_signal_R_to_A = new double[R_unit_kosu];
            double[] learn_signal_R_to_A = new double[A_unit_kosu];

            //R 層の教師信号をすべてのユニットについて求める
            for (int i = 0; i < R_unit_kosu; i++)
            {
                teacher_signal_R_to_A[i] = 2 * (output_R[i] - teacher_signal[dataIndex,i])
*output_R[i] * (1.0 - output_R[i]);

            }
            //R 層から A 層の重みの変化量を求める。
            for (int i = 0; i < A_unit_kosu; i++)
            {
```

```
                sum = 0.0;
                for (int j = 0; j < R_unit_kosu; j++)
                {
                    deltaw_A_to_R[j, i] = -eta * teacher_signal_R_to_A[j] * output_A[i]
 + alpha * deltaw_A_to_R[j, i];
                    w_A_to_R[j, i] += deltaw_A_to_R[j, i];
                    sum += teacher_signal_R_to_A[j] * w_A_to_R[j, i];
                }
                learn_signal_R_to_A[i] = output_A[i] * (1 - output_A[i]) * sum;
            }
            //A 層から S 層への重みの変化量を求める。
            for (int i = 0; i < S_unit_kosu; i++)
            {
                for (int j = 1; j < A_unit_kosu; j++)
                {
                    deltaw_S_to_A[j, i] = -eta * learn_signal_R_to_A[j]
 * output_S[dataIndex, i] + alpha * deltaw_S_to_A[j, i];

                    w_S_to_A[j, i] += deltaw_S_to_A[j, i];
                }
            }
        }

        public double sigmoid(double x)
        {
            return 1.0 / (1.0 + Math.Exp(-x));
        }

    }

    class hyouzyou
    {
        static int Main(string[] args)//メイン
        {
            //Console.WriteLine("学習:1, 表情認識:2");
            //int a = int.Parse(Console.ReadLine());
            switch (args[0])
            {
                case "1":
                    gakusyu();
                    return 1;
                case "2":
```

```
                    int a = kekka();
                    return a;

                default:
                    Console.WriteLine("おしまーい");
                    return 19;
            }

        }

        public static string[] GetFilesMostDeep(string stRootPath, string stPattern)
        {
            System.Collections.Specialized.StringCollection hStringCollection = (
                new System.Collections.Specialized.StringCollection()
            );

            // このディレクトリ内のすべてのファイルを検索する
            foreach (string stFilePath in System.IO.Directory.GetFiles(stRootPath, stPatter
            {
                hStringCollection.Add(stFilePath);
            }

            // StringCollection を 1 次元の String 配列にして返す
            string[] stReturns = new string[hStringCollection.Count];
            hStringCollection.CopyTo(stReturns, 0);

            return stReturns;
        }

        static void gakusyu()
        {
            NNCore core = new NNCore(30/*学習のデータ数*/, 0/*テスト用のデータ数
*/, 11/*入力層のユニットの個数*/,
                300/*中間層のユニットの個数*/, 3/*出力層のユニットの個数*/);

            //学習用データ入力＋出力（教師）
            //怒りの顔の学習
            string[] stFilePathesAngry
= GetFilesMostDeep(@"gakusyukao\processingkao\ikari", "ikari*.txt");

            for (int u = 0; u < 10; u++)
            {
```

```
                using (StreamReader r = new StreamReader(stFilePathesAngry[u]))
                {
                    for (int i = 0; i < core.S_unit_kosu; i++)
                    {
                        core.output_S[u, i] = double.Parse(r.ReadLine());
                    }
                    core.teacher_signal[u, 0] = 1.0;
                    core.teacher_signal[u, 1] = 0.0;
                    core.teacher_signal[u, 2] = 0.0;

                }
            }


            //笑顔の顔の学習
            string[] stFilePathesSmile
 = GetFilesMostDeep(@"gakusyukao\processingkao\egao", "egao*.txt");

            for (int u = 0; u < 10; u++)
            {
                using (StreamReader r = new StreamReader(stFilePathesSmile[u]))
                {
                    for (int i = 0; i < core.S_unit_kosu; i++)
                    {
                        core.output_S[u + 10, i] = double.Parse(r.ReadLine());
                    }
                    core.teacher_signal[u + 10, 0] = 0.0;
                    core.teacher_signal[u + 10, 1] = 1.0;
                    core.teacher_signal[u + 10, 2] = 0.0;

                }
            }
            //驚きの顔の学習
            string[] stFilePathesSurprise
= GetFilesMostDeep(@"gakusyukao\processingkao\odoroki", "odoroki*.txt");

            for (int u = 0; u < 10; u++)
            {
                using (StreamReader r = new StreamReader(stFilePathesSurprise[u]))
                {
                    for (int i = 0; i < core.S_unit_kosu; i++)
                    {
```

```
                core.output_S[u + 20, i] = double.Parse(r.ReadLine());
            }
            core.teacher_signal[u + 20, 0] = 0.0;
            core.teacher_signal[u + 20, 1] = 0.0;
            core.teacher_signal[u + 20, 2] = 1.0;

        }
    }



    //ニューラルネットワークを初期化
    core.Initialize1();

    //backprop による学習
    for (int j = 0; j < 100000; j++)
    {
        for (int i = 0; i < core.gakusyu_kosu; i++)
        {
            core.FP(i);
            core.BP(i);
            Console.WriteLine(j);
        }
    }



    Console.WriteLine("学習が終わったゾ");
    Console.WriteLine("学習した重みを保存する:1");
    Console.WriteLine("保存はしない:1 以外");



    int b = int.Parse(Console.ReadLine());
    switch (b)
    {
        case 1:
            using (StreamWriter w = new StreamWriter(@"weight\P_s_to_a.txt"))
            {
                for (int i = 0; i < core.S_unit_kosu; i++)
                {
```

```
                        for (int j = 1; j < core.A_unit_kosu; j++)
                        {
                            w.WriteLine(core.w_S_to_A[j, i]);
                        }
                    }
                }

                using (StreamWriter w = new StreamWriter(@"weight\P_a_to_r.txt"))
                {
                    for (int i = 0; i < core.A_unit_kosu; i++)
                    {
                        for (int j = 0; j < core.R_unit_kosu; j++)
                        {
                            w.WriteLine(core.w_A_to_R[j, i]);
                        }
                    }
                }

                break;
            default:
                break;


        }

    }

    static int kekka()
    {
        NNCore core = new NNCore(0/*学習のデータ数*/, 10/*テスト用のデータ数
*/, 11/*入力層のユニットの個数*/, 300/*中間層のユニットの個数*/, 3/*出力層のユニット
の個数*/);

        //学習した重みを適用
        core.Initialize2();

        //入力データを挿入！
        using (StreamReader r
 = new StreamReader(@"C:\Users\b1013152\Desktop\test\kinect_test\Release\file\egaosa.txt"))
        //"C:\Users\b1013152\Desktop\test\kinect_test\kinect_test\file\egaosa.txt"
        {
```

```
                for (int i = 0; i < core.S_unit_kosu; i++)
                {
                    core.output_S[0, i] = double.Parse(r.ReadLine());
                }

            }



            Console.WriteLine("--------------------");
            Console.WriteLine("表情");
            core.FP(0);
            for (int k = 0; k < core.R_unit_kosu; k++)
            {
                Console.WriteLine("{0:f20}", core.output_R[k]);
            }

            if (core.output_R[0] >= core.output_R[1] && core.output_R[0] >= core.output_R[2]
                return 3;

            if (core.output_R[1] >= core.output_R[0] && core.output_R[1] >= core.output_R[2]
                return 4;

            if (core.output_R[2] >= core.output_R[1] && core.output_R[2] >= core.output_R[0]
                return 5;

            return 100;


        }
    }
}
```

参考文献

[1] 麻生英樹(あそうひでき). 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 1988.
[2] 甘利俊一(あまりしゅんいち). 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] 伊藤尚枝(いとうひさえ). 認知過程のシミュレーション入門. 北樹出版 2005.
[4] 中野馨(なかのかおる). ニューロコンピュータの基礎. コロナ社 1990.
[5] McLeod, P. Plunkett, K. Rolls,E,T. 認知過程のコネクショニスト・モデル. 北樹出版 2005.
[6] 小林宏(こばやしひろし),原文雄(はらふみお). 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 Vol.29, pp.112-118, 1993.
[7] 伊藤美加(いとうみか),吉川左紀子(よしかわさきこ). 表情認知における顔部位の相対的重要性. 人間環境学研究、Vol.9, pp89-95, 2011.
[8] 高野裕士(たかのひろし),鈴木健嗣(すずきけんじ). 装着型表情識別インタフェースによる表情知覚支援への応用. 電子情報通信学会論文誌 D、Vol.J99-D No.1, pp.67-75, 2016.