平成 23 年度　特別研究報告書

自己組織化マップを使用した
キーボード入力による個人認証の改良

龍谷大学　理工学部　情報メディア学科

学籍番号　T080415　小松篤史

指導教員名　三好　力　教授

目次

内容梗概

ネットワーク上で本人かどうかを確認するためにパスワードを入力する方法が多く用いられている.しかし,パスワード入力は本人でなくても入力すればサービスを利用できてしまい,悪用される恐れもある.パスワード認証が抱える問題を解決するために,個人認証システムに自己組織化マップを導入し,セキュリティ能力の向上を図る技術が提案されている.しかしこの技術は,学習したマップを使用し続けるため,パスワード入力の慣れ等によって入力パターンが変化した時に対応することができない.本研究ではその問題点を解決するために,学習が終わったマップを再学習し,入力者の入力パターンが変わった場合でも精度を落とすことなく個人認証を行うことができる手法を提案する.実験では,パスワードを複数回入力することで入力の慣れによるパターン変化が現れるかどうか検証し,既存技術と提案手法とで比較を行い提案手法のほうが有効であることを証明する.結果は,入力回数を重ねると入力パターンが変化することを確認できた.また,既存技術では入力パターンが変化した入力では認証できないが,提案手法では認証が成功したので,提案手法が有効であると証明された.

1章　はじめに

今日,コンピューターは企業だけではなく一般家庭にも多く普及している.それは現代の生活に影響を与えており,ネットワーク上のサービスやアプリケーションによって利便性が向上している.そういったサービスを使用する際に,使用する本人かどうかを確認するためにパスワードを入力する方法が多く用いられている.しかし,パスワードを入力すれば認証するということは裏を返すとパスワードを入力さえすれば第三者でも本人になりすましてサービスを受けることができる.その第三者に悪意があった場合は,そのアカウントを悪用され被害を被ることも予想される.複数のアカウントで共通のパスワードを使用すると,パスワードの管理は容易であるが,もしそのパスワードが外部に流出したときに他のアカウントも被害を受ける恐れがある.この問題を解消するためにアカウントの数だけパスワードを用意すると,それらすべてを覚えておくことが難しくなり,パスワードを管理するためにメモなど目に見える形で保管してしまう.これではセキュリティ能力が低下してしまう.

パスワードを入力する認証以外にもICカード等を利用する認証や,身体的特徴を利用する生体認証が存在する.ICカードを利用した認証システムは,認証に物理的な鍵を必要とするので,パスワード認証のようにそれを記憶しなくてもよいというメリットがある.しかしICカードは内蔵されたICチップによって認証を行っているので,第三者の手に渡ってしまうと悪用されてしまう可能性がある.またICカードを紛失してしまうと,自身の認証が不可能になってしまうのも問題である.

生体認証は,個人の身体的特徴を利用するので,第三者がその特徴を盗むことは難しくセキュリティ能力は高いものとなっている.参照する特徴も指紋,声紋,網膜など多種に及ぶ.しかし,特徴を盗むことは難しいが特徴を偽装する方法は確立されてきているので,セキュリティ能力が完璧であると言いきれなくなっている.また,ICカードによる認証と同様に導入コストが非常に高くなってしまうので一般家庭には普及しにくい.

このようなそれぞれの認証システムが抱える問題を解決するために,個人認証システムに自己組織化マップを導入し,セキュリティ能力の向上を図る技術が提案されている.この方法は,キーの入力リズムを身体的特徴として扱う生体認証である.キーボードのキーの押された時間とキーが離された時間を取得し,それを学習データとして保存する.その学習データで学習を行いマップを作成し,そのマップと入力データを比較して本人であるか否かを判断する.そのマップは学習を終わった時点で二度と更新されることはない.とすると,もし入力者本人の入力パターンが変化した場合,そのパターンの変化に対応することができない.例えば,パスワード入力を一日一回以上行った場合,何回も同じパスワードを入力することで入力に慣れが生じる.このように入力パターンが変化すると,学習したマップでは本人が入力していても本人だと判断されない.よって認証精度が低下し,セキュリティ能力に問題が生じる.

本研究ではその問題点を解決するために,なんらかの規約を設けて学習が終わったマップを再学習し,入力者の入力パターンが変わった場合でも精度を落とすことなく個人認証を行うことができる手法を提案する.既存技術と提案手法を比較して提案手法が有効であることを証明する.また,入力パターンが時間経過によって変化するかどうか確認するために,用意した文字列を複数回入力することで入力の慣れによるパターン変化が現れるかどうかも確認する.

2 章　自己組織化マップの概要

2-1 自己組織化マップとは

自己組織化マップ（Self-Organization-Mapping：通称 SOM）とは,競合学習型ニューラルネットワークの一つであり,入力層と出力競合層の 2 層から成る表現モデルである.自己組織化マップは 1980 年代にコホネンによって開発され,多次元データの分類,解析を容易にする技術として知られている.

自己組織化マップの特徴は,n 次元ベクトル集団を学習することで 2 次元マップにそれらのベクトルの関係を写像することができることである.似ているベクトルは 2 次元マップ上の近い位置に配置され,そうでないベクトルは遠い位置に配置される.汎用性が高く,認識,予測,分類など様々な応用が可能である.

2-2 自己組織化マップの基本的アルゴリズム

コホネンは,人間の脳の情報処理の仕方を以下の式で表した.

$$m_i(t+1)=m_i(t)+h_{ci}(t)[x(t)-m_i(t)] \qquad (1.1)$$

この式は,次のような意味を持っている.神経細胞(ノード)i が時刻 t で処理している情報処理能力を $m_i(t)$ とする.外部から入力信号 x(t) が入ってきた時,細胞はこの入力信号を学習して次の時刻には入力信号により近い情報処理能力 $m_i(t+1)$ を持つようになる.このとき,x(t) が n 次元の入力ベクトルである場合,参照ベクトルとも呼ばれる $m_i(t)$ もまた同じ n 次元の要素を持つ.そして, $h_{ci}(t)$ は学習率係数を含めた近傍関数を意味する.ここで,t = 0,1,2・・・は離散時間座標である.出力競合層のベクトルは参照ベクトル $m_i(t)$ で表され,入力層の次元に合わせて n 個の要素を持っている.図に示すように,視覚的に出力を見るために,普通は 2 次元に祭列されている.

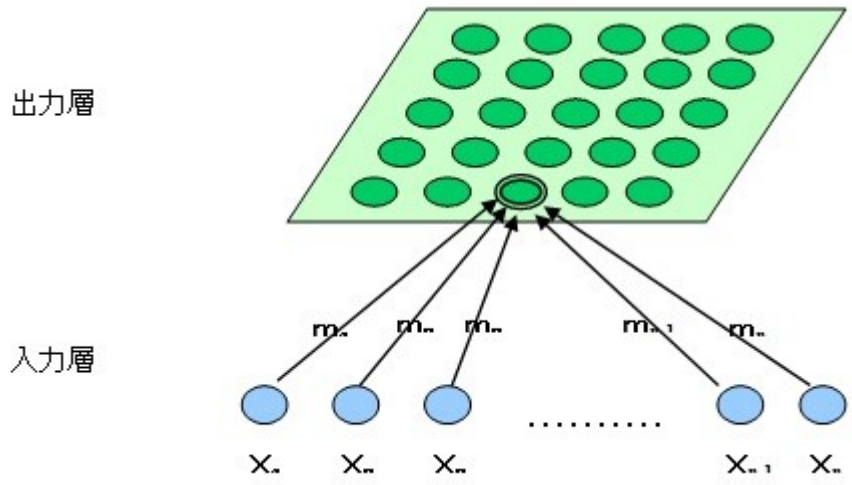

図 1　自己組織化マップの構造

自己組織化マップの学習は次のように行われる.入力ベクトル x(t)はある測度,例えばユークリッド距離 | $x(t)-m_i(t)$ | を最小にするノード i を探し,それに添え字 C をつける.

$$|x(t)-m_c|=min|x(t)-m_i(t)| \qquad (1.2)$$

式(1.2)で決められた参照ベクトル $m_c(t)$ を持つノードを勝者ノードと呼ぶ.式(1.1)と式(1.2)での学習の状態を図 2 に示す.まず入力ベクトルが提示されると,その入力ベクトルに一番近いノードが勝者ノードとなる.そして勝者ノードの周囲のノード,図 2 の円で囲われている領域を近傍領域と定義する.その形は正方形でもよい.基本ノードの配列を六角形の形をした場合は周囲 6 個の六角形が近傍領域ということになる.その近傍内の全てのノードは入力ベクトルを学習し,式(1.1)に従って入力ベクトルの方向へ修正される.この学習を繰り返し行うのだが,このとき近傍領域の範囲は最初は広く指定しておき,学習が進行するにつれて近傍領域を狭めていく.

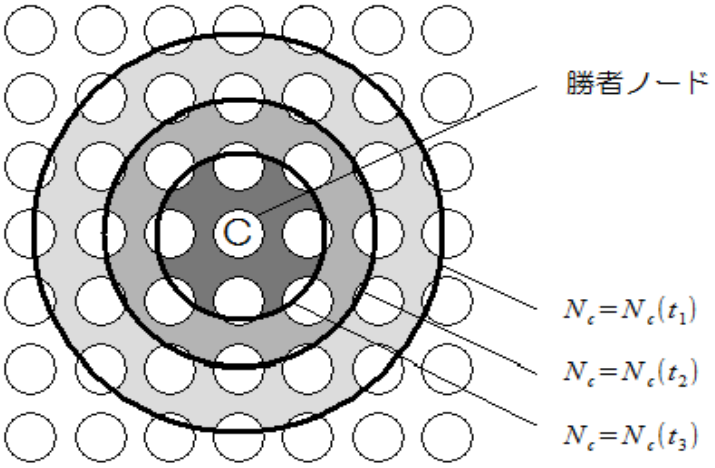

図 2　自己組織化マップの学習回数における近傍領域の変化

式(1.1)において, $h_{ci}(t)$ は学習率係数 $\alpha(t)$ と近傍関数 $h(d,t)$ により以下の式のように表現できる.

$$h_{ci}(t)=\alpha(t)*h(d,t) \qquad (1.3)$$

近傍領域は,学習される近傍領域を指定する関数である.近傍領域は,出力競合層の勝利ノードの近傍を意味し,学習によって参照ベクトルが更新される領域である.学習をする際,最初は近傍領域をその範囲を大きくとり,学習が進行するとともに徐々にその領域を狭める.近傍領域を減少させる関数には,線形型とステップ型が存在する.ここではよく使用されるガウス型近傍関数について説明する.

$$h(d,t)=\exp(-d^2/k(r(t))^2) \qquad (1.4)$$

ここで,d は式(1.5)で表せる,勝者ノードから参照ベクトルまでの距離, $k(r(t))$ は学習時間 t のときの近傍領域の最大距離で,ガウス関数の波幅の係数である.

$$d^2=(x-a_i)^2+(y-b_i)^2 \qquad (1.5)$$

ここで, (a_i, b_i) は勝者ノードの位置,(x,y)は半径 $r(t)$ により成る円形領域の内側にあるノードの位置を示す.

近傍領域の半径 $r(t)$ は,学習の初めでは大きく,学習が進行すると小さくなっていく.

半径 $r(t)$ の円領域に含まれる範囲のノード,参照ベクトルは,指数関数によって決められる重みを持って学習する.すなわち,勝者ノードに近いほどその類似性が大きくなるように学習され,勝者ノードから遠ざかるほど,その類似性が小さくなるように学習される.そして半径 $r(t)$ の範囲外のノードは学習が行われないことになる.

したがって,式(1.1)によって学習している間,近傍領域 N_c 内のノードに関しては $h_{ci}(t)=\alpha(t)*h(d,t)$,近傍領域 N_c 外のノードに関しては, $h_{ci}(t)=0$ である.つまり,近傍領域の外側の領域では学習は行われない.

以上より,近傍関係を以下の式で表す.

$$\begin{aligned} h_{ci}(t)&=\alpha(t)*h(d,t) && (i\in N_c)\\ h_{ci}(t)&=0 && (i\notin N_c) \end{aligned} \qquad (1.6)$$

このとき, $\alpha(t)$ の値を学習率係数と呼び, $0<\alpha(t)<1$ の値を持つ. $\alpha(t)$ と N_c の大きさはどちらも学習時間が経つにつれて普通,単調減少させる.例えば, $\alpha(t)$ であれば次の式で定義してもよい.

$$\alpha(t)=\alpha_0(1-t/T) \qquad (1.7)$$

ここで, a_0 は a の初期値であり,普通 0.2~0.5 の値を選ぶ.T は行われるべき学習での予定された全更新学習回数である.ただし,式(1.1)中,比例係数の $a(t)$ は学習の最初では大きな値をとるように設定し,学習の進行とともにその値を減少させるようにする.また,近傍領域 $N_c=N_c(t)$ も式(1.6)と同様に減らしていってもよい.つまり,

$$N_c(t)=N_c(0)(1-t/T) \qquad (1.8)$$

ここで, $N_c(0)$ は初期値である.

コホネンの自己組織化マップアルゴリズムを整理すると,以下のようある.

1. 出力層にノードを配置し,それぞれの持つ参照ベクトルを乱数で初期化する.
2. 入力ベクトルに最も近い(類似している)競合層での参照ベクトルを探し,これを勝者ノードとする.
3. この勝者ノード及びその近傍内のノードを式(1.1)に従って更新する.また,学習が進むにつれて近傍領域を狭め,学習率係数の値も式(1.4)のように減少させていく.

自己組織化マップの学習を行うために,学習パラメータを設定する必要がある.ここに主要な学習パラメータとその説明を示す.

学習回数:競合層に対して行う学習の回数を設定する.
学習係数:一回の学習でノードをどれだけ更新するか設定する.
近傍領域:学習に影響を与える範囲の初期値を設定する.

3 章　既存技術

コンピュータのセキュリティの問題は非常に重要なってくる.そこでその問題を解決するのが既存の技術である.主に知識認証, 所有物認証, 生体認証の３つがあげられる. それぞれの認証方法の特徴, 問題点を挙げる.

3-1 知識認証

知識認証は主にパスワード, ワンタイムパスワードが存在する. 今現在最も普及しているであろう認証システムである. 銀行の暗証番号やポータルサイトによく搭載されている.
サーバー側のプログラムが難解ではなく, 開発費用が他のシステムに比べて安価に抑えることができ, 認証を行う際にあらかじめ登録しておいた文字列を入力するだけなので非常に簡単である.
多くの場面で採用され, 便利とされているパスワード認証にも問題点は存在する. その問題点をいくつか列挙する.

・推測されやすいパスワードの設定
まず, はじめに登録したパスワードは忘れずに覚え続けなければならない. そのため殆どの人間が覚えやすい単語や自身の個人情報をパスワードを設定してしまうことが多い. 一種類の文字のみを使用したものであったり, 生年月日等の自身の情報, 4 文字程度の短いパスワードを設定しまいがちだが, こういったパスワードは基本的に設定しないほうがいいとされている. 辞書の単語や生年月日を採用したパスワードであれば, それを特定するための攻撃方法があり, ソフトウェアも存在する. よって単語や個人情報をパスワードに採用するのは望ましくない.

・記憶が難しいパスワードの設定
上記の問題を解決する方法として, 推測されにくい文字列を設定することが挙げられる. ローマ字と数字を不規則に組み込んだパスワードが最も理想的である. しかし特に規則性のないパスワードを記憶しておくことは非常に難しい. そのパスワード一つを記憶しておくことが可能だとしても, クレジットカード等のその他のパスワードも記憶していくとするとそれぞれのパスワードと混同してしまい記憶することが難しくなる. パスワードを忘れた時のためにメモ帳などに筆記しておくという方法がある. しかし他人でも見ることができるため, パスワードが漏洩する可能性がありセキュリティの低下につながる.

・共通のパスワードの設定
それぞれのアカウントに対して, 共通のパスワードを使用することでパスワードの管理は簡単なものとなる. しかし, そのパスワードが漏洩してしまうと全てのアカウントで不正されてしまう恐れがある. この問題を解決するために, アカウントの数だけパスワードを用意する方法がある. この方法であれば, もしパスワードが漏洩して不正利用されたとしても被害を抑えることができる. しかし, アカウントの数だけパスワードを用意すると, 管理が難しくなる. そうすると, 管理するためにメモ等に記入

して管理することになり,セキュリティの低下につながる.よって管理性とセキュリティ性を両立させることは難しい.

・パスワードの漏洩
　たとえ複雑なパスワードであっても,漏洩してしまうとセキュリティ機能は完全に失われる.パスワードは,通信回線の盗聴や,キーロガー等のキー入力を記録するソフトウェアによって漏洩するパターンが多い.またパスワードを一度盗まれてしまうと,本人が次にサービス等を利用する時まで被害にあったことに気付くことが難しい.

3-2 所有物認証

　所有物認証は主にICカード,USBトークン等が存在する.ICカードはCPUやメモリなどを構成したICチップを内包したカードであり,USBトークンは,USB内に著名情報を組み込んだいわばコンピュータの鍵のようなものである.パスワード認証の場合は,たとえばインターネット上のシステムの場合に世界中のどこからでも不正利用できてしまう.ICカードの等の認証デバイスを利用した認証の場合,ICカードを持ってないと認証できないので,セキュリティの強度は大幅に上がる.これらは企業や大学で個人認証ツールとして利用されており身分証明証などの用途としても併用できる強みがあるが,所有物認証にも問題点は存在する.所有物認証の問題点をいくつか列挙する.

・成りすましが容易
　ICカード,USBトークンともに物理的な鍵であるため,貸し借りが容易である.そのため悪意のある人間にICカードを貸してしまうと,それを悪用されてしまう恐れがある.また,ICをカードは物理的な物であるため,ICカードを紛失してしまう恐れがある.そして,それを拾った第三者が悪用する可能性もある.
・費用がかかる.
　ICカード,USBトークンともに費用が高額になる.ICカードは一枚あたりの費用が250円程度の金額となり,USBトークンは一つ15000円程度の金額である.企業等たくさんの人の分だけ用意するとなると,単価が安いとはいえどちらも非常に高額になる.

3-3 生体認証

　知識認証,所有物認証に変わり様々なところで使用されているのが生体認証である.生体認証は個人の特徴を使用することでセキュリティ能力を強固なものにしているが,生体認証にも欠点は存在する.生体認証が持つ欠点をそれぞれ挙げる.
・指紋認証の問題点
　人間の指に存在する指紋を使用する個人認証である.個人の特徴を使用するのでセキュリティ強度は高いが,欠点も存在する.そのいくつかの欠点を挙げる.
　専用のハードウェアが必要
　指紋によって個人認証を行うためには,指紋を読み取るためのハードウェアと,認識した指紋を記録,処理し,認証に必要な状態にするためのソフトウェアがそれぞれ必要となる.これらを用意すると,費用がかかるので気軽に導入することが難しい.

偽装方法の確立

生体認証の中で最も古いものにあたる指紋認証は指紋を偽装してセキュリティを突破する方法が存在している. 残留指紋をゼラチンに写し取り人口指を作成し, その人口指でセキュリティを突破することに成功している. こういった例があるので安全性に欠ける. また, 木工用ボンドを使用することでスライド式の指紋認証を突破ことが日本の学生によって提唱される. 実際に実システムに対して指に特殊なテープを張り, 指紋の変造を行った事件も発生している.

傷を負うと認証ができない

指紋認証には指紋を使用するので, 指に傷を負うと認証は不可能になる. これは修復可能な程度の軽傷ならば問題ない. しかし火傷などの指紋が変形するほどの怪我をであると, 永遠に認証する手段を失うことになる. 実際, ゴルフで手が荒れてしまい指紋による認証ができなくなった例も存在する.

・網膜認証の問題点

人間の目の部分に存在する網膜を用いた認証は, 網膜の終生不変性から高い認証精度を発揮するが, 網膜を使用するが故の欠点もある. その欠点をいくつか挙げる.

装置が非常に高価

網膜認証を行うために使用される装置は非常に高額であり, 容易に導入できるものではない. この問題は他の生体認証にもいえることである.

疾患の影響を受けやすい

網膜は非常に疾患の影響をうけやすい, 白内障や緑内障, 糖尿病や高血圧症によって網膜パターンが影響を受け認証されなくなった例も報告されている.

照明に影響される

赤外線を使用するため, 照明や他の光の影響を受けやすく, 環境が変化すると認証システムが機能しなくなる場合がある.

・虹彩認証の問題点

虹彩認証の認証能力は, コンタクトレンズを装着していても低下することがなく環境にも左右されにくいが決して万能ではない.

認証の手間

数メートル離れると認証ができなくなり, 対象者が頭を動かしていたりカメラに目を向けていない場合は認証することができない. 更に, 普段とらないような行動を取らなければならないため, 手間がかかる.

画質の問題

画像を用いる生体認証では, 画像の品質によって認証精度が大きく変化する. 認証精度を上げるためには鮮明な画像を用意すればいいが, それを用意するためのハードウェアにかかる費用は非常に大きい.

・筆跡認証の問題点

生体認証の中では上記で挙げたものほどコストが掛からず, 容易に導入することができる点は優秀である. 筆跡自体は模倣することが可能で信頼性は高くない. そこに書いている過程の筆圧を導入すると, セキュリティ能力が向上するが, 圧力測定器などのハードウェアが追加で必要となってくるため, コストの問題が発生してしまう.

・音声認証の問題点

音声を使用するため, 声帯や声を出すための器官に異常がある場合は認証が難しくなってくる. また, 録音機材によって認証精度が左右されてしまうのも問題である. そして良質な機材を揃えるとなると, コストの問題が発生する.

種類によって問題点は異なるが, 共通して以下のような問題がある.

・体の一部を必要とする認証システムの場合, その必要な部位が怪我や病気等で損傷すると認証が不可能になる. また対象者が成長した場合に, 身体的特徴が大きく変化してしまうと本人であった場合でも認証されない可能性がある.
・生体情報を認証に使用する場合, システム管理者に情報を公開することになる.
・専用のハードウェア・ソフトウェアを用意しなければならないため, 一般家庭で導入するには費用がかかる.

3-4 パスワードとSOMを用いた知識認証

上記の問題点を解決するために, キーボード入力を用いた個人認証方法が存在する. この認証方法には自己組織化マップを使用している. 具体的な手順として, まず使用者本人の文章入力のデータを採取する. このとき採取するのは, キーボードを打つ際のキーを押してから離すまでの時間と離してから次のキーが押されるまでの時間の 2 種類のデータである. この 2 種類のデータを自己組織化マップの学習データとして学習を行い, 作成されたマップをサンプルとして保存しておく. そして, プログラムを秘密裏に起動させておき, 実際に入力されたパターンとサンプルとの比較を行い, 本人であるかどうかを判定する. 入力されたパターンとサンプルのパターンに一定以上の差がみられた場合, 管理者が連絡する. この認証方法は個人の入力リズムという身体的特徴を使用した生体認証なので, セキュリティ能力は高い. またこの技術であれば生体認証の問題点である導入費用を安価に抑えることが可能である.

上記の既存技術では, 学習して作成されたマップは一度学習を終えてしまうと以降それをサンプルとして使い続ける. 使用し始めた頃は照合はうまくいくが, 一定の期間が経ち使用者の入力パターンが入力の慣れ等によって変化した場合に, 初期のマップとの照合がうまくいかない場合が発生する. 本人が入力しているのに本人でないと認識されてしまうと, セキュリティ技術として意味をなさなくなる.

4章 提案手法

3-4 の問題点を解決するため, 既存技術の改良を行う.

入力パターンが変化しても正しく認証を行うために, 一定期間ごとにデータを収集しておき, それを用いて一定期間後に再学習を行う手法を提案する. 以下にその概要を示す.

・自己組織化マップによって作成したマップをサンプルとし, 入力されたデータを比較をする. マップの初期値は乱数で決定し, それぞれのノードに値を代入する. 学習データは今後比較する本人の入力時間データを採取する. パスワード入力を行った際のキーが押された時間と, キーが離された時間を学習データとする. そして指定した回数だけ学習を行う.
・学習したマップと入力されたパスワードを比較し, 本人であると判断された入力データであれば本人の入力データとして保存しておく. 本人ではなかった場合の入力データは別の領域に保存しておく.

・一定数のデータが蓄積された時, 保存しておいた入力データを学習データとして使用しマップを再学習させる.

・最新の入力データで学習したマップとなるので, 使用者の変化した入力パターンであっても精度の高い認証を行うことができる.

普段から使っているパスワードで既に最適化されている入力パターンであれば改めて入力パターンが変化することは殆どない. しかし, 新しいパスワードを提供された時やパスワードを変更した場合であれば, 使用者はまだパスワードを打ち慣れていないので, 後々に入力パターンが変化する可能性がある.

5章 実験

5-1 実験方法

今回の実験では,既存技術と提案手法を使って検証を行い,提案手法が入力パターンの変化に対応できているかを確認する.

実験では,まず自己組織化マップの学習データとして使用する入力時間を採集するためのプログラムを作成する.採集する時間は,キーが押された時間とキーが離されて次のキーが押された時の時間の2種類である.採集は java のキーイベントの KeyPress と KeyRelease を使用し,最初に初めてキーが押された時に入力時間を取得する.次に離された時間,または押された時間を取得し,最初の入力時間との減算を行い,キーが押された時間と離された時間を算出する.算出された時間は変数に代入しておき,同様にキーイベントが発生するごとに計算を行い入力パターンを測定する.今回は 1/1000 秒の入力時間をデータとして採集を行う.

実験 1

まず最初に,入力パターンが時間の経過によって変化するのかを見る実験を行う.被験者にはあらかじめこちらで用意したパスワードを入力してもらい,入力された時間を測定し保存する.入力は日を明けて行ってもらい,月日が経過することによって入力パターンに変化が出ているか検討を行う.パスワードはランダムで文字と数字を並べた「ayzdc0e819」を採用した.不規則な文字列であれば,個人によって連続で打ち込まれる文字数な感覚等の特徴が現れやすくなるのではないかと考える.パスワード入力を 100 回行ってもらい,最初の入力と 100 回目の入力を比較する.

実験 2

既存技術と提案手法で実験を行い,比較・検討をする.実験1で採集した入力データを学習データとして使用する.提案手法では,1 回目から 20 回目のパスワード入力のデータを保存し,これを学習データとして再学習を行う.そして 21 回目の入力データをテストデータとし,マップ上で認証が行えているか検証を行う.同様の検証をあと 4 回行い,合計 100 回分のデータで 5 回の再学習を行う.既存技術では,あらかじめ学習したマップを使用し,提案手法で入力されたテストデータをこのマップに入力し,認証が行えているか検証を行う.最終的に学習しおわったマップで既存技術との比較を行う.

今回の実験では SOM-PAK[1]を使用する.SOM-PAK による出力は,ノードの色が白であれば,要素の特徴が近く,ノードの色が黒に近づくほど要素が離れている,と見ることができる.
以下に SOM-PAK の仕様と入力したデータを記す.

- din* , cout*:入力データと出力データの指定をそれぞれ行う.
- xdim,ydim:X 軸と Y 軸の長さを指定する.今回は xdim を 30,ydim を 20 と設定する.
- rlen:学習回数を指定する.今回は 10000 回の学習を行う.
- alpha:学習率係数を指定する.今回は 0.2 と設定する.
- radius:近傍領域を指定する.今回は 20 と設定する.
- rand:乱数の範囲を指定する.今回は 1000 と設定する.

5-2 実験 1 の結果

実験 1 の結果を図 3 に示す.

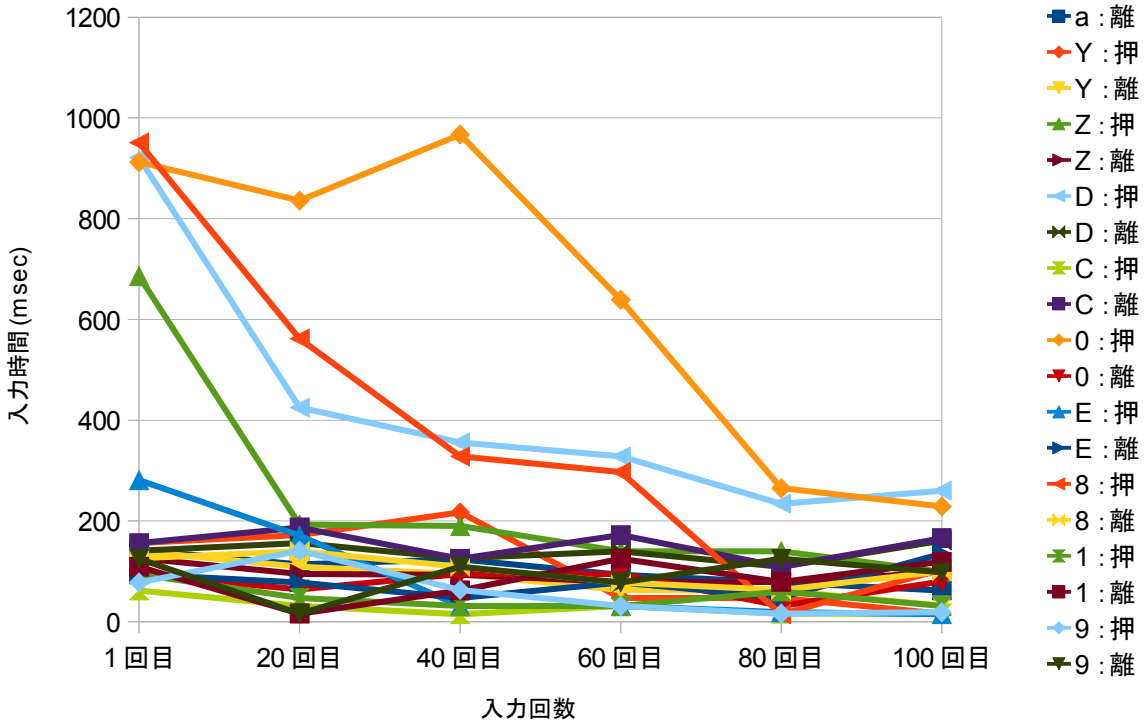

図 3：入力時間の変化率

図 3 は, ある被験者の 10 文字のパスワードを入力した場合の入力時間の変化率を表している. グラフ中の折れ線は, パスワード入力の際に採集した 19 個のデータである. ○は各文字を表し, 「○：押」はキーが押された時, 「○：離」は押されていたキーが離された時の時間を表している. また, X 軸は入力回数, Y 軸はパスワードを入力した際のそれぞれのキー入力の時間を msec で表している. これを見ると, 1 回目から 100 回目に進むにつれて, 特に「z」「d」「0」「8」の入力時間が短くなっていることがわかる. これは入力者がパスワードを覚える時に「ayz」「dc」「0e」「819」と分けていたため, 入力をする際に間が開いたものだと思われる. しかし回数をこなすにつれて入力速度が速くなり, 入力回数 100 回目では入力に時間がかかっていた 4 箇所の入力時間が短縮され, 入力も「ayz」「dc」「0e819」と変化しているのがグラフから見て取れる. よって入力パターンが変化していると言える. また, キーが押されている間の時間は 100 回目までの入力ではわずかな変化しか見られなかった.

5-3 実験 2 の結果

実験 2 の既存技術と提案手法の比較を以下の図に示す.

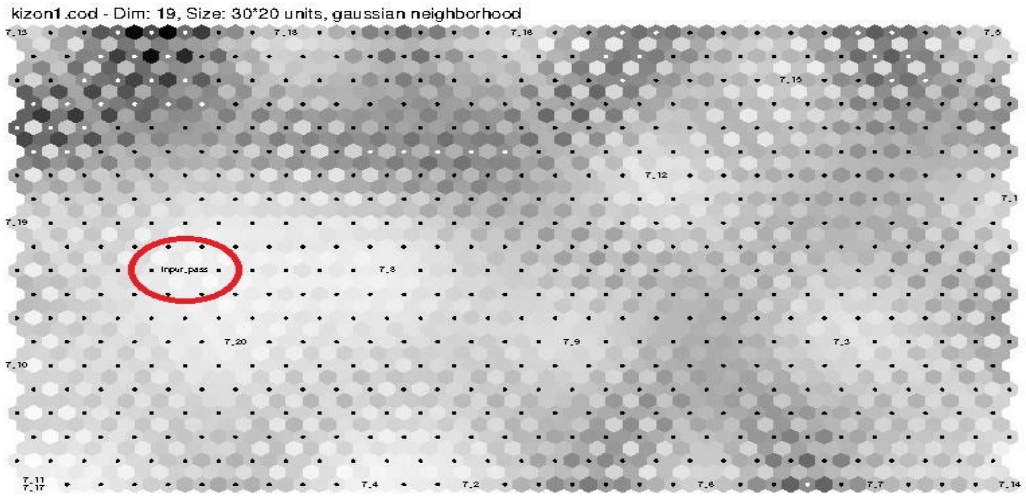

図 4-1：既存技術のマップにテストデータ(21 回目の入力)を入力した時の結果

これを見ると, 中央左にテストデータが出現しており, 周辺のノードが白いので正しく認証していると言える.

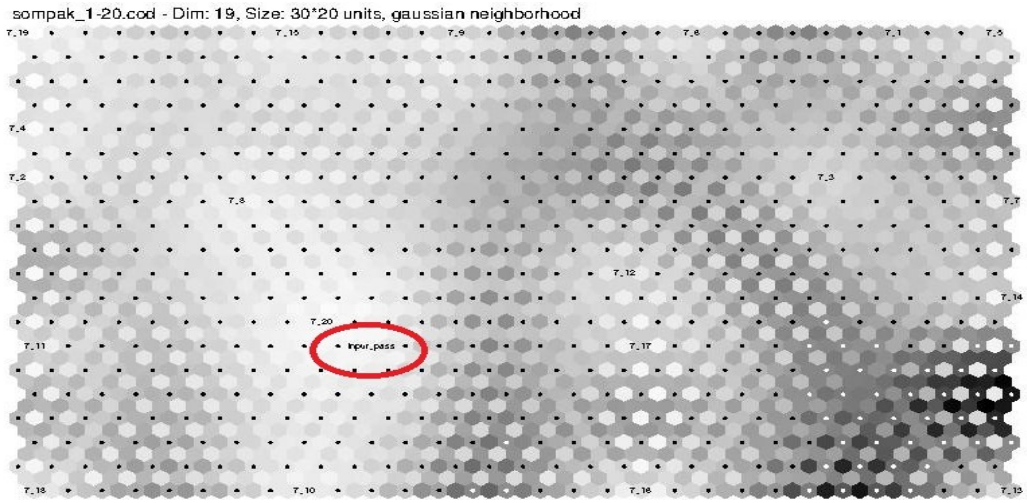

図 4-2：提案手法のマップにテストデータ(21 回目の入力)を入力した時の結果

これを見ると, 中央付近にテストデータが出現しており, 周辺のノードが白いので正しく認証していると言える.

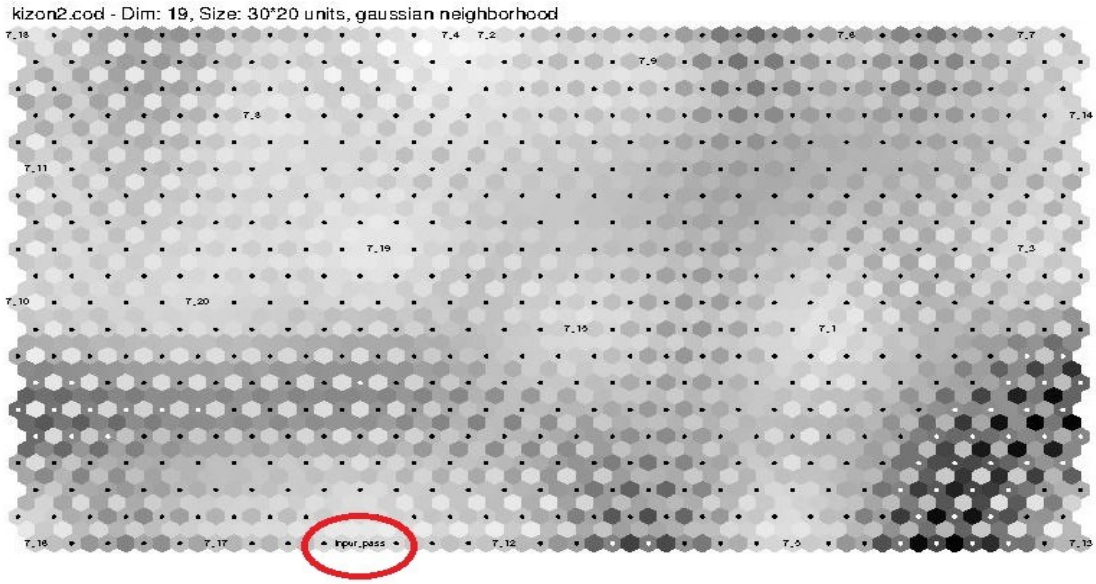

図 5-1：既存技術のマップにテストデータ(41 回目の入力)を入力した時の結果

これを見ると,左下にテストデータが出現しており,周辺のノードが白いので正しく認証していると言える.

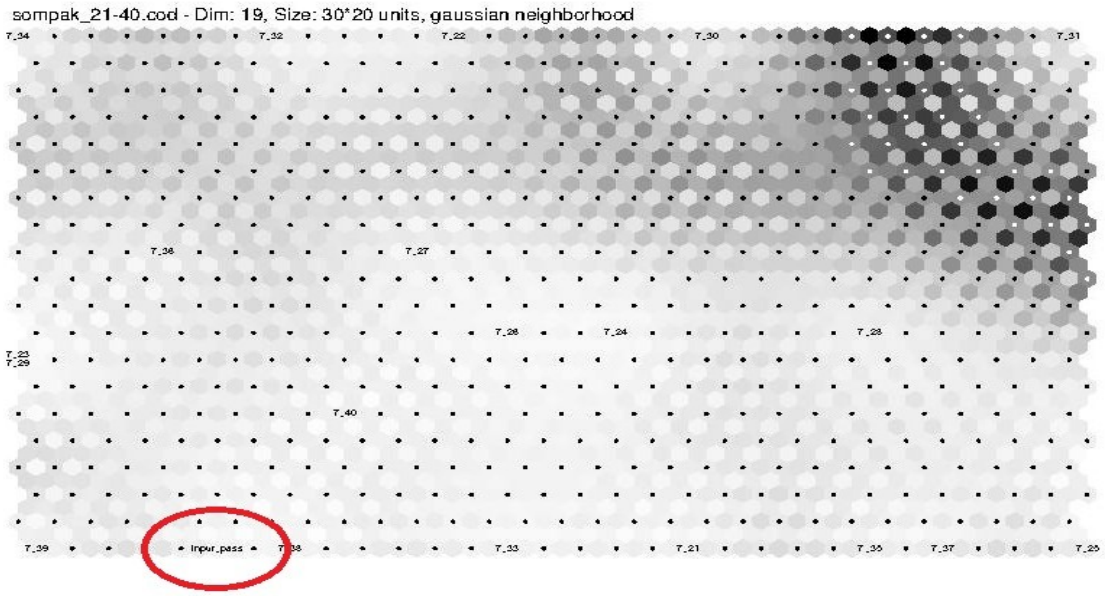

図 5-2：提案手法のマップにテストデータ(41 回目の入力)を入力した時の結果

これを見ると,左下にテストデータが出現しており,周辺のノードが白いので正しく認証していると言える.

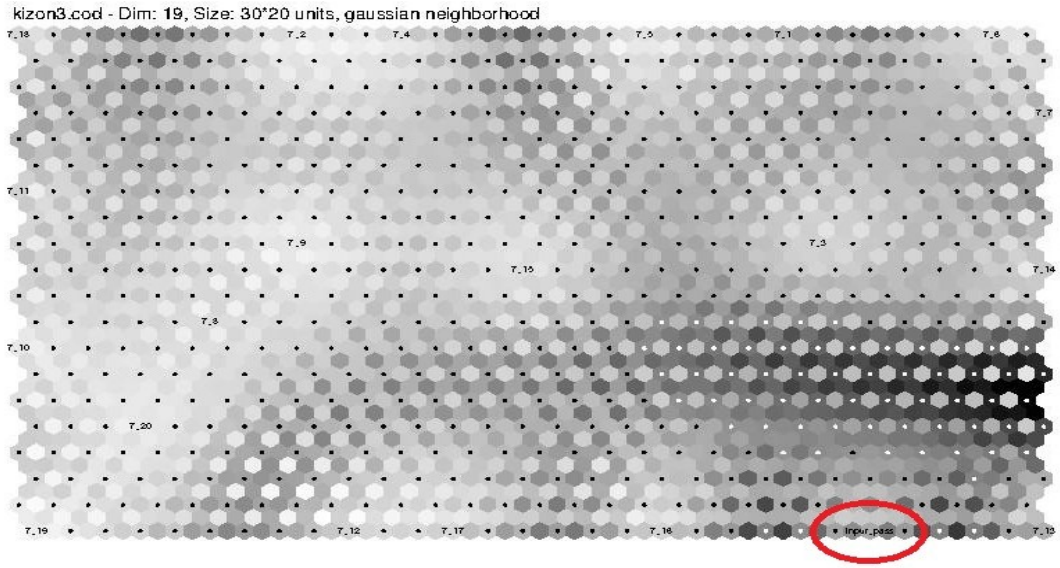

図 6-1：既存技術のマップにテストデータ(61 回目の入力)を入力した時の結果

これを見ると, 右下にテストデータが出現しており, 周辺のノードが黒いので認証ができていないと言える.

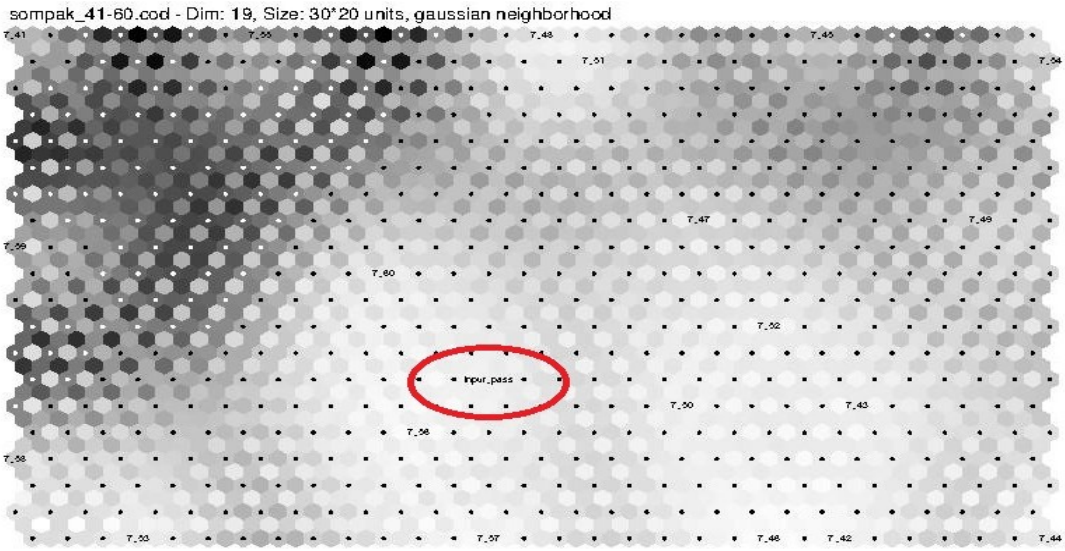

図 6-2：提案手法のマップにテストデータ(61 回目の入力)を入力した時の結果

これを見ると, 中央付近にテストデータが出現しており, 周辺のノードが白いので正しく認証していると言える.

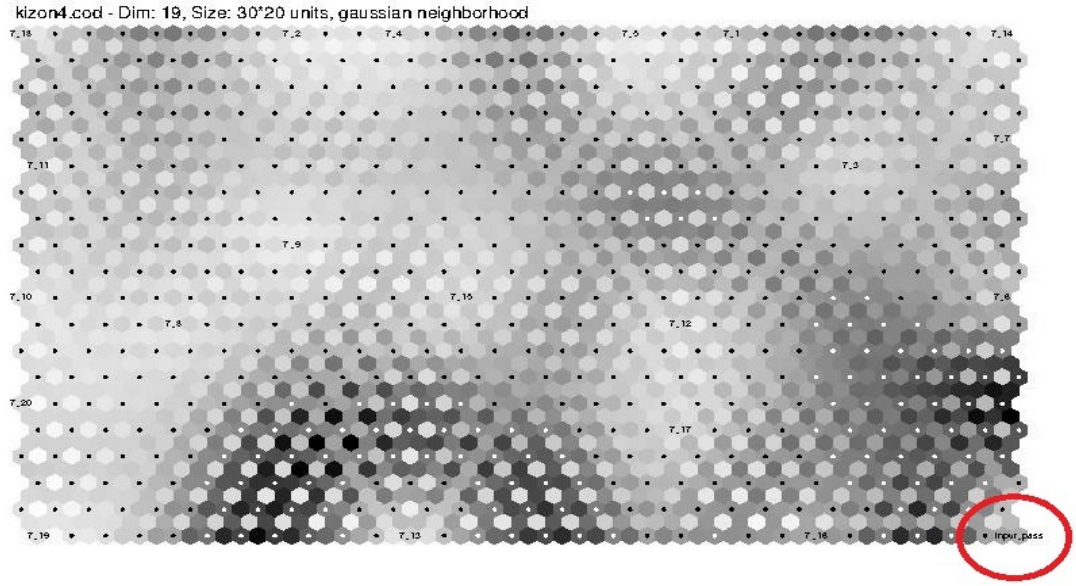

図 7-1：既存技術のマップにテストデータ(81 回目の入力)を入力した時の結果

これを見ると, 右下にテストデータが出現しており, 周辺のノードが黒いので認証ができていないと言える.

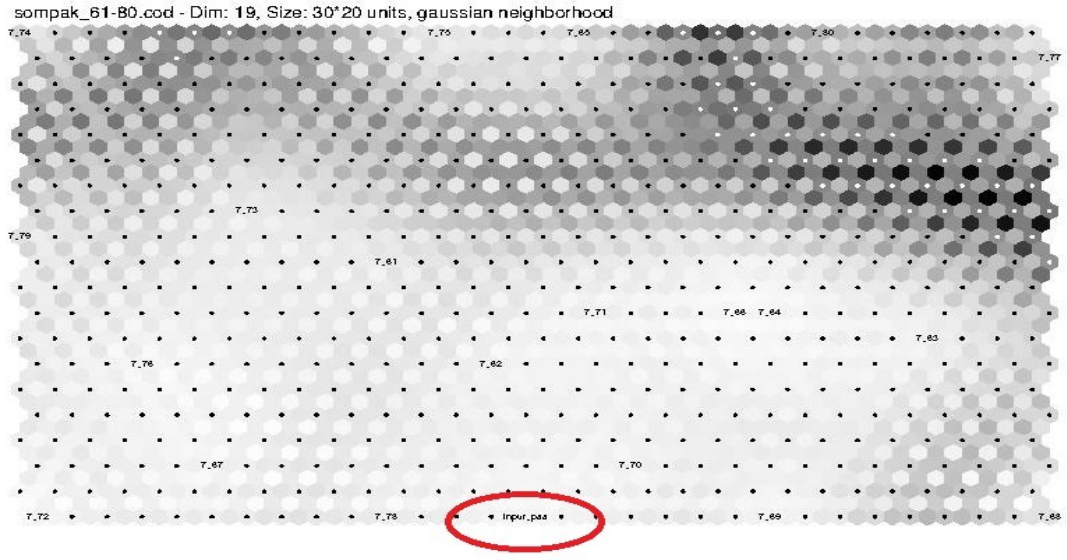

図 7-2：提案手法のマップにテストデータ(81 回目の入力)を入力した時の結果

これを見ると, 中央付近にテストデータが出現しており, 周辺のノードが白いので正しく認証していると言える.

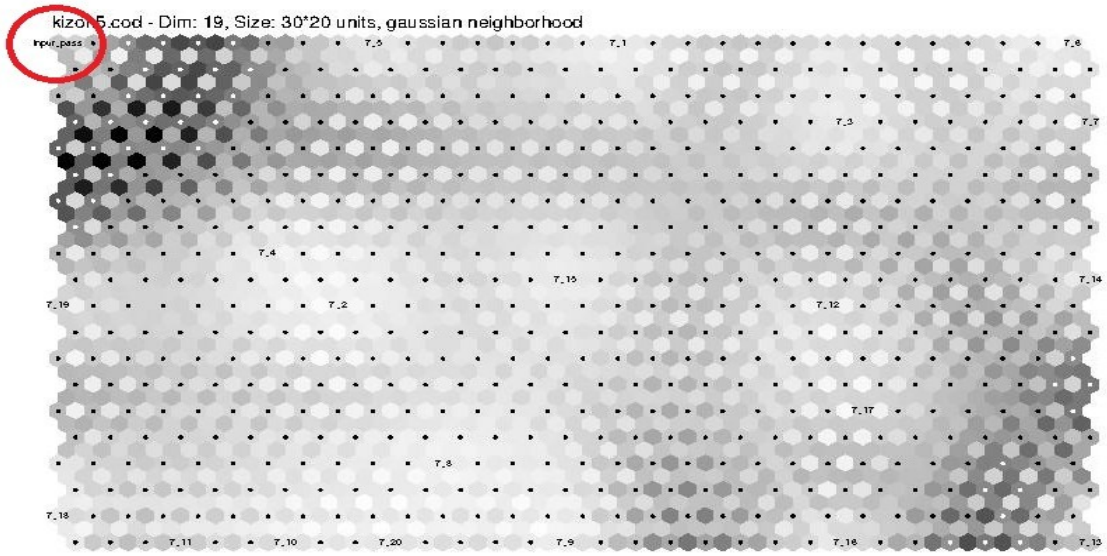

図 8-1：既存技術のマップにテストデータ(101 回目の入力)を入力した時の結果

これを見ると, 右下にテストデータが出現しており, 周辺のノードが黒いので認証ができていないと言える.

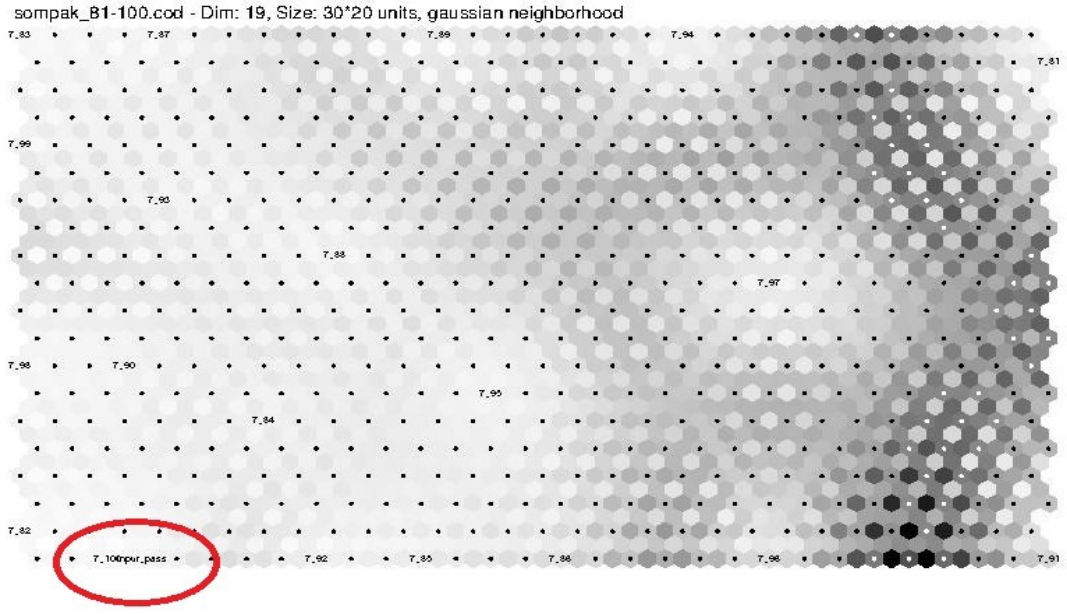

図 8-2：提案手法のマップにテストデータ(101 回目の入力)を入力した時の結果

これを見ると, 右下にテストデータが出現しており, 周辺のノードが白いので認証ができていると言える.

実験 2 の結果をまとめると,以下となる.

図 4-1, 4-2 の結果を見ると, マップを作成する際に採集した入力データと特徴の差がなかったため, 既存技術, 提案手法ともにテストデータが本人と判断できるノード付近に現れている. パスワード入力を重ね再学習を行っていくと, 図 6-1 の既存技術のマップ以後は, テストデータが黒いノードに囲まれた付近に現れるようになったので, 本人と判断することができなくなっている. しかし提案手法のマップでは再学習を行っているので, 図 6-2 のようにテストデータが白いノード付近に現れるので, 本人と判断できている. その後 5 回の再学習を行った場合でも, 提案手法のマップではテストデータが白いノード付近に現れているので, 提案手法は有効であることがわかる.

6章 まとめ

今回はパスワード入力による認証システムが持つ問題点を解決する自己組織化マップを用いた手法の,入力パターンの変化によって認証精度が落ちる問題点を解決するために,入力データを使った再学習を行う手法を提案した.それにより,入力パターンが変化した場合でも認証精度を落とすことなく認証を行うことが可能となった.入力パターンが入力の慣れによって変化するかを見るための実験 1 では,1 回目の入力の結果から 100 回目の入力の結果を見る限り,入力回数を重ねると入力パターンが変化することがわかった.また実験 2 では,入力パターンが変化した場合でも提案手法では認証精度を落とすことなく個人認証を行うことができたので,既存技術よりも有効であると言える.

今後の課題として,実験 1 より,押された時の時間による特徴が多く見られたが,離された時の時間の特徴はどれも類似した時間であった.なので,パスワード文字数を増やさずに入力を増やすため,パスワードに大文字アルファベットを採用し,shift キーによる時間も考慮すると,より多くの入力特徴が現れて認証精度が向上するのではないかと考える.また誤字をしたときのバックスペース入力は対応できていないので,その入力もデータとして採用するとより認証精度が上がると考える.

謝辞

本研究を進めるにあたり,様々なご指導を頂きました三好力教授に深謝いたします.
また,発表を通じ,多方面からの示唆をしてくださった三好研究室の皆様にも感謝いたします.

参考文献

1. HowToUseSOM_PAK
   http://www.forest.eis.ynu.ac.jp/~toshy/som/somuse.html
2. 認証方法いろいろネットマークス
   https://www.netmarks.co.jp/product/netmarks/column_ninsho_houshiki.html
3. BIOMETRICS
   http://www.jaisa.jp/action/group/bio/shoukai.html
4. 徳高平蔵/大北正昭/藤村喜久朗 「自己組織化マップとその応用」
5. 徳高平蔵/大藪又茂/堀尾恵一/藤村喜久朗/大北正昭 監修「自己組織化マップ」
6. 大北正昭/藤村喜久朗/徳高平蔵/権田英功 編「自己組織化マップとそのツール」
7. 香島健 著 平成 21 年度特別研究報告書「SOM を用いたキーボード入力による個人認証」
8. マーク M.ヴァン・フッレ著 徳高平蔵/藤村喜久朗 監訳「自己組織化マップ-理論・設計・応用」

付録

```
public class pass_watch extends Frame implements
KeyListener{

        private int             status = 0;
        private Date    startTime;
        private long diff;
        private long diffx;
        int data_count = 1;

        public static void main(String [] args){

        new pass_watch();
        }
        pass_watch(){

JFrame frame = new JFrame();
JLabel label = new JLabel("ayzdc0e8l9",JLabel.CENTER);
label.setFont(new Font("Arial", Font.PLAIN, 20));
        frame.add(label);
        frame.setSize(300,200);
        frame.setLocation(900,300);
        frame.addKeyListener(this);
        setSize(100,200);
        frame.setVisible(true);

                }
public void keyPressed(KeyEvent e) {

String outputFileName = "c:\\som_data_file\\実験
1_outputdata\\som_pak_data.dat";
 // ファイルオブジェクトの生成
File outputFile = new File(outputFileName);
try{
// 出力ストリームの生成
 FileOutputStream fos = new
FileOutputStream(outputFile,true);
 OutputStreamWriter osw = new OutputStreamWriter(fos);
 PrintWriter pw = new PrintWriter(osw);

        if(status == 9){
        status = 0;
                                }
 else if(status == 0){
if(e.getKeyCode() == KeyEvent.VK_ENTER){
        pw.println("");
        status = 0;
        }
elseif(e.getKeyCode() != KeyEvent.VK_ENTER){
        startTime   = new Date();
        status = 1;
                }
        }

else if(status == 1 ){
if(e.getKeyCode() == KeyEvent.VK_ENTER){

        pw.print("");
        status = 0;
                                        }       else
if(e.getKeyCode() != KeyEvent.VK_ENTER){

        Date enterTime     = new Date();
        long diff_millisec = enterTime.getTime() -
startTime.getTime();
        diff = diff_millisec;
        diffx = enterTime.getTime();
pw.print("押[" + e.getKeyChar() + "](" + diff + ")");

        status = 2;
                }
        }

else if(status == 2 ){
if(e.getKeyCode() == KeyEvent.VK_ENTER){

        pw.println("");
        status = 0;
                }
else if(e.getKeyCode() != KeyEvent.VK_ENTER){

        Date enterTime     = new Date();
        long diff_millisec = enterTime.getTime() - diffx;
        diffx = enterTime.getTime();
        diff   = diff_millisec;
pw.print("押[" + e.getKeyChar() + "](" + diff + ")");
                                        }
                                }
                        // 後始末
                        pw.close();
// エラーがあった場合は,スタックトレースを出力
                } catch(Exception a) {
                 a.printStackTrace();
                }
        }

public void keyReleased(KeyEvent e) {
String outputFileName = "c:\\som_data_file\\実験
1_outputdata\\som_pak_data.dat";
 // ファイルオブジェクトの生成
 File outputFile = new File(outputFileName);

         try{
        // 出力ストリームの生成
 FileOutputStream fos = new
FileOutputStream(outputFile,true);
OutputStreamWriter osw = new OutputStreamWriter(fos);
PrintWriter pw = new PrintWriter(osw);

        if(status == 0){
        if(e.getKeyCode() == KeyEvent.VK_ENTER){
        status = 0;
                                }
else if(e.getKeyCode() != KeyEvent.VK_ENTER){

        startTime   = new Date();
pw.print("離[" + e.getKeyChar() + "](" + diff + ")");
        status = 1;
                                }
                        }
```

```
else if(status == 1 ){
if(e.getKeyCode() == KeyEvent.VK_ENTER){
            status = 0;
                                                }
else if(e.getKeyCode() != KeyEvent.VK_ENTER){

            Date enterTime     = new Date();
            long diff_millisec = enterTime.getTime() -
startTime.getTime();
            diff = diff_millisec;
            diffx = enterTime.getTime();
pw.print("離[" + e.getKeyChar() + "](" + diff + ")");

            status = 2; //status2 へ移行
                                    }
                        }

else if(status == 2 ){ // 測定開始
if(e.getKeyCode() == KeyEvent.VK_ENTER){

            status = 0;
                                                }
else if(e.getKeyCode() != KeyEvent.VK_ENTER){

            Date enterTime     = new Date();
            long diff_millisec = enterTime.getTime() - diffx;
            diffx = enterTime.getTime();
            diff   = diff_millisec;
pw.print("離[" + e.getKeyChar() + "](" + diff + ")");

                                    }
                        }
             pw.close();

 // エラーがあった場合は,スタックトレースを出力
                            } catch(Exception a) {
                              a.printStackTrace();
                                }
                        }
            public void keyTyped(KeyEvent e) {

            System.out.print(e.getKeyChar());
                        }
            }
```