博士学位論文

リカレントニューラルネットワークによる
チャネルコンダクタンス時系列の自動推定法の開発

高橋 正明

第1章 序論

第1節　はじめに

神経科学の研究において、電気生理学実験を行っている実験家らによって数多くの様々なチャネルの電流・電圧特性及びチャネルの電圧依存的な動特性などが得られている。そのために電気生理学的な手法である、ホールセルクランプ法やパッチクランプ法などの技術が使われる。また神経科学の研究には計算機を用いた計算機神経科学研究がある。計算機神経科学研究で盛んに行われている、計算機シミュレーションでは神経細胞の膜特性を始め、様々なチャネルの特性に関するパラメータを精密に設定できる利点がある。例えば、神経細胞ネットワークを構築する際、神経細胞のチャネルを自由に設計し、神経細胞の数、シナプスの種類やその強度を自由に設定出来る。また、神経発火に対する薬品の効果をシミュレートすることも可能である。電気生理実験家は計算機シミュレーション結果を自分の実験にフィードバックすることができるだろう。例えば、神経細胞ネットワークを自由に設計し電気生理実験結果を説明するような条件を計算機上で探すことができる。そうすれば、実際のネットワーク上の神経細胞やシナプス強度などにたいして何らかの示唆を与えてくれるだろう。実験家はその情報をもとにして、それが正しいかどうかを電気生理実験によって明らかに出来るであろう。しかしながら、神経のシミュレーション方法を最初から学び実行するのは電気生理実験を行う実験家にとって敷居が高いものと思われる。より簡単に神経細胞の発火をシミュレート出来るシステムがあればシミュレーションの初学者でもすぐに計算する事が出来、自分の研究結果の解釈に役立つ可能性がある。

第2節　神経細胞

神経細胞とは情報処理・記憶のために特異に分化した細胞である。それは細胞核を持つ細胞体から数多くの突起を伸ばしている。その中で最も長く伸びているものが軸索と呼ばれ、他の突起は樹状突起と呼ばれる。軸索から他の神経細胞の樹状突起に結合を作り信号が伝達される。2 つの神経細胞間の結合部分をシナプスといい、信号を受け取った神経の作用の仕方によって興奮性と抑制性に分けられる。多くの神経細胞は他の神経細胞と結合して神経細胞ネットワークを形成する。

神経細胞の膜の内外には電位差があり、その電位は膜電位と呼ばれる。細胞膜には各種チャネルがあり、それぞれイオンの通し方に特性がある。主なものは Na^+チャネル、K^+チャネル、Ca^{2+}チャネル、Cl^-チャネル、カチオンチャネル等がある。それぞれのイオンには平衡電位が定義され、それは濃度勾配による拡散的な効果と電位勾配による電気化学的な効果

とがちょうどつりあい、正味のイオンの移動がなくなる時の電位である。神経細胞は電流注入やシナプス入力によって膜電位が急上昇し、その後、急下降する活動電位という現象を起こすことができる。以下にその典型的な例を説明する。電流注入などによって膜電位が上昇し、ある閾値を越えると電位依存性 Na^+チャネルが開く。Na^+は細胞外の方が内部に比べて濃度が高く、Na^+チャネルが開くと、内部に Na^+が流入する。すると、さらに膜電位が上昇（脱分極）する。膜電位が上昇するとより Na^+チャネルチャネルが膜電位によるポジティブフィードバックによって、膜電位は Na^+の平衡電位付近まで上昇する。その後、Na^+チャネルの不活性化や遅延整流性K^+チャネルの開口によるK^+の細胞外への流出によって元の静止膜電位以下に膜電位が下げられる。その後、不応期という刺激を与えても活動電位を起こすことができない期間を経てもとの静止膜電位に戻る。神経細胞は活動電位を生成することにより、シナプスを介して他の神経細胞に信号が送られる。

第3節　電気生理実験から神経細胞の計算機シミュレーションまでの一般的な流れ

神経細胞の計算機シミュレーションを行うためには、実際の神経細胞を用いて電気生理学実験を行い、様々な膜電位に膜電位を固定した時のチャネルコンダクタンスの時間変化を測定し、その変化を表現出来る計算式を見つけ計算できるようにする必要がある。この節では実験から神経シミュレーションまでの流れを説明する。

第1項　電位固定実験によるチャネルコンダクタンスの時間変化の測定

神経細胞のシミュレーションを行うためには、その細胞膜にある個々のチャネルのコンダクタンスダイナミクスを計算する必要がある。そのためにチャネルコンダクタンスを電位固定実験法により測定する。この実験では神経細胞の膜電位を任意に固定し、その電位でのコンダクタンスの時間変化を測定される。例としてこの実験により得られた、Hodgkin-Huxleyによるヤリイカ巨大軸索のNa^+チャネルコンダクタンス時間変化[1]をFigure 1-1 に示す。

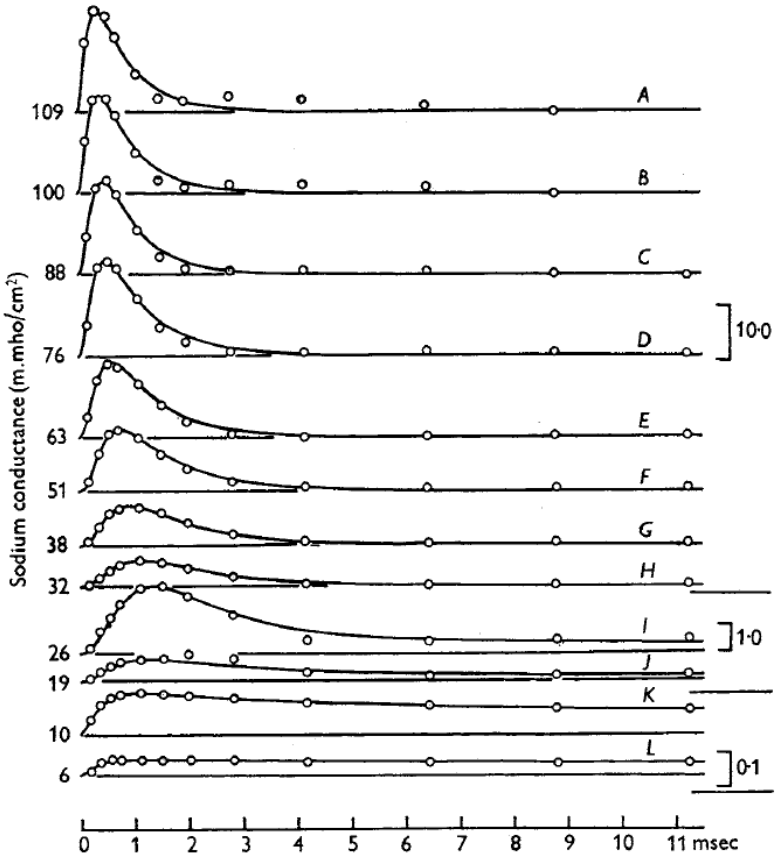

Figure 1-1　電位固定実験におけるヤリイカ巨大軸索の Na^+チャネルコンダクタンス時間変化
図の左側に固定した電位の値を示している。（静止電位を 0 mV としている）

第2項　　チャネルコンダクタンスダイナミクスの定式化

実験的に求めたチャネルコンダクタンスの時間変化をシミュレートするためには、時間変化を定式化することが必要である。Huxley はチャネルコンダクタンスの時間変化をゲートという概念を用いて定式化した。ゲートは、チャネル中のイオンが通る通路にある確率的に開閉する門のように考えられ、膜電位が上昇すると開く確率が上がる活性化ゲートと、逆に膜電位が下がると開く確率が増加する不活性化ゲートがある。Huxley は実験結果を説明するように Na^+チャネルは 3 つの活性化ゲートと 1 つの不活性化ゲートをもつと考え、Na^+チャネルコンダクタンスダイナミクス g_{Na} を定式化した。そして Na^+の平衡電位 V_{Na} と膜電位 V との差と g_{Na} をかけて Na^+チャネル電流を計算できる。 g_{Na} と I_{Na} を付録の第 1 節に示す。

第3項　　神経活動のシミュレーション

神経細胞膜はFigure 1-2のように抵抗とコンデンサーを並列回路として結合した等価回路として表現することができる。そして神経細胞膜の膜電位は膜にあるチャネルを通って流れるイオン電流とコンデンサーを流れる容量性電流と刺激電流の和を 0 と想定することで計算できる。

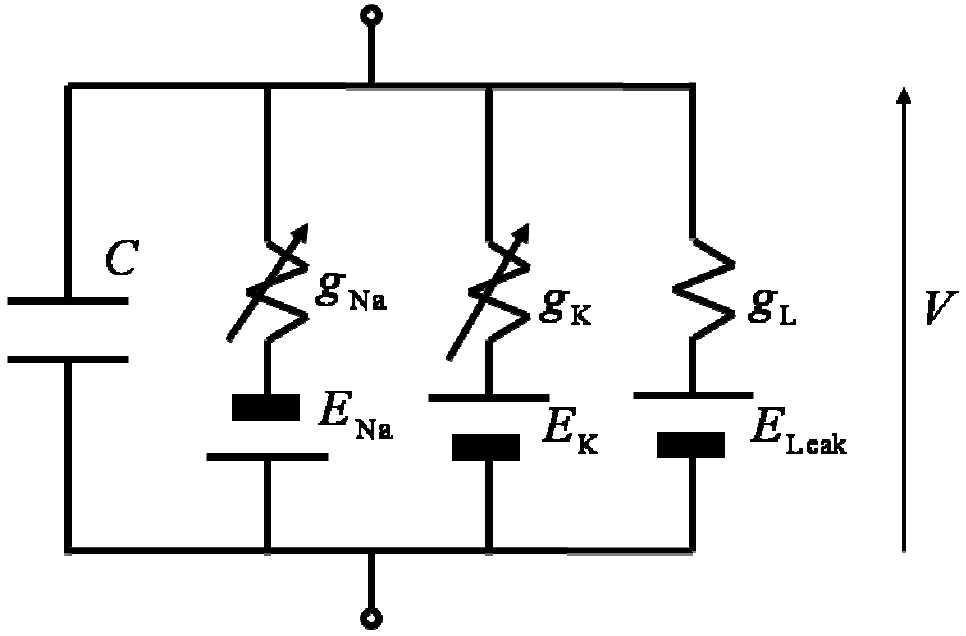

Figure 1-2 神経細胞膜の電気的活動を表す並列等価回路。 C は膜容量で、g_{Na}、g_Kとg_Lはそれぞれ Na^+チャネル、K^+チャネルおよびリークコンダクタンスである。

そして、回路は並列になっているので膜電位はチャネル電流の足し合わせで計算できる（付録の第 1 節）。以上の方法が実験からシミュレーションまでの一般的なプロトコールとなっている。

第4節　チャネルコンダクタンス定式化の既存の方法

チャネルコンダクタンスダイナミクスの定式化の方法として、ゲート変数を第 3 節第 2 項で示したように定式化する方法が考えられる。この方法では式中のα_mやβ_mなどの速度定数を膜電位V の関数として実験結果をうまく説明できるように定式化しなくてはならない。

一般的には、計算機神経科学者が、電気生理実験者が電位固定実験法により測定したチャネルコンダクタンスのデータから式を類推する。α_mやβ_m、最大コンダクタンスなどのパラメータを実験結果に合うように推定する。

またより簡便な方法として、固定膜電位に対するゲート変数の収束値と収束する時の時定数でゲート変数のダイナミクスを計算する方法もある。mゲートの場合を考えてみると、適切に電位固定実験を行うことで、収束値$m_\infty(V)$と時定数$\tau_m(V)$を実験的に求め、$m_\infty(V)$をボルツマン関数で、$\tau_m(V)$をガウシアン関数で近似する[2]。そしてmゲート変数のダイナミクスをそれら 2 つの関数を用いて表現する。詳細な式は付録の第 2 節に示した。他のゲートについても同じように適切に電位固定法を行い求める。しかしながら、この方法では、収束値と時定数の 2 つの関数の近似を行う必要があるためそれらを実験結果に合うようにパラメータフィッティングする必要がある。

さらに神経細胞の膜電位の時間変化とそのときの刺激電流値を用いてチャネルコンダクタンスのダイナミクスを推定する方法もある [3]。この方法では、まず神経細胞のいくつかのチャネルが想定され、そのチャネルダイナミックスの基本的な式を仮定し、そして式中にあるパラメータを刺激時の神経細胞の膜変化から推定する。この方法では仮想的なチャネルのコンダクタンスダイナミクスが神経細胞の神経活動から推定されるので、仮定したチャネルが実際に存在するかどうかは不明である。

第5節　RNN を用いたチャネルコンダクタンスダイナミクス自動推定

既存のチャネルコンダクタンスダイナミクス定式化の方法では、定式化の知識と経験が必要であり、初学者がシミュレーションに興味をもち、自分のデータを用いてチャネルコンダクタンスダイナミクスを再現し、シミュレーションしたいと思ったとしても、初学者にとっては敷居が高い。初学者が神経活動のシミュレーションに興味を持ち実際に計算しようと思った場合、計算機神経科学者が既に論文として発表している神経細胞膜の計算は既存のシミュレータなどで計算可能かもしれないが、自分で新しく見つけたチャネルを用いてシミュレーションしようと思った時、新しくチャネルを組み合わせて計算しようとする場合などは困難を伴うと考えられる。このような場合でも初学者がシミュレーションを行えるシステムの開発が必要なのではないか、初学者でもすぐに実験データを用いてシミュレーションできる仕組みが必要なのではないかと私は考えた。電気生理実験から神経活動の計算機シミュレーションを行う過程でもチャネルダイナミックスを定式化する過程は初学者にとって困難な過程である。従って、初学者にとっても簡便に神経活動のシミュレーションが可能なように、電気生理実験によって得られたチャネルコンダクタンスダイナ

ミクスのデータのみを与えるだけで、そのダイナミックスを再現ができるような自動推定法が必要であると考えた。

さらにチャネルダイナミクスの推定方法の開発に、ニューラルネットワークの分野で用いられているリカレントニューラルネットワーク（recurrent neural network; RNN）技術を用いることを考えた。その理由は、RNN は外部からの入力データを用いて時系列を学習し、そして再現することが可能であるからである[4]。また学習した時系列そのものだけでなく、そのダイナミックスも学習可能である[5]と考えられているからである。従って、膜電位を入力してコンダクタンスの時系列、つまり電位固定法の実験データを出力し、そのダイナミックスも推定出来、神経活動のシミュレーション計算に用いる事が出来ると考えられた。RNN による神経細胞のチャネルコンダクタンスダイナミクスを推定し、神経活動を計算する方法は、世界的にも初めてで、非常にユニークな方法である。

RNN を用いたチャネルコンダクタンスダイナミクスの自動推定を用いた本論文の提案手法における、電位固定実験から神経活動シミュレーションまでの流れを以下に説明する。まず、神経細胞から電位固定実験を用いて保持膜電位におけるチャネルコンダクタンスの時間変化を測定する。次に、そのチャネルコンダクタンスの時間変化を固定した電位を RNN への入力データ、コンダクタンスを出力データとしてその時間変化（時系列）を学習する。そうするとチャネルコンダクタンスダイナミクスを再現できる RNN が生成され、それらを用いて神経活動を計算する。以上の流れを Figure 1-3 に示す。

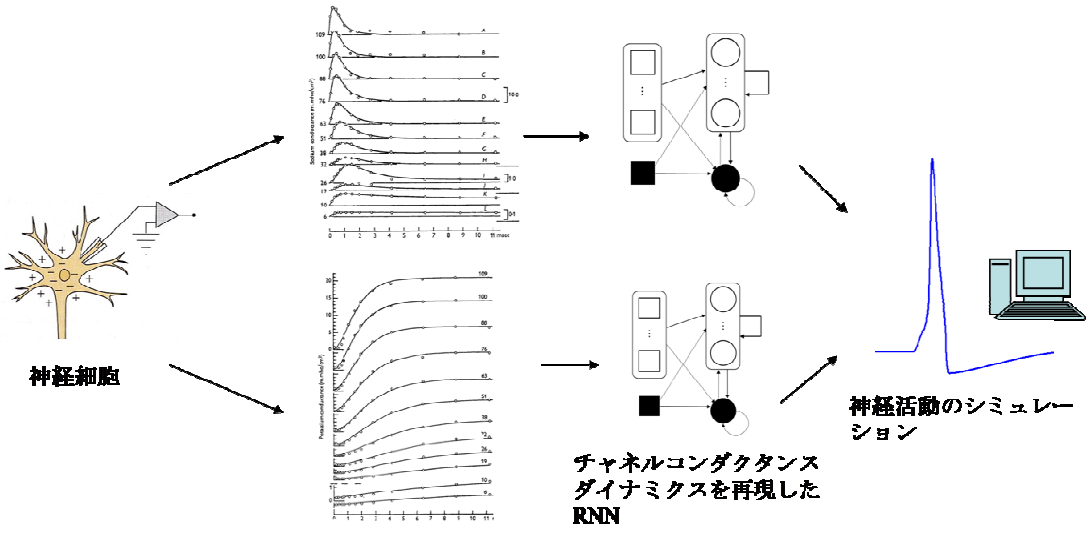

Figure 1-3　RNN を用いた場合の、実験からシミュレーションまでの流れ　神経細胞より記録されたチャネルコンダクタンスの時間変化を RNN でそれぞれ再現する。そして、それらを用いて神経活動をシミュレーションする。

第6節　HIPPOSTATION システム

脳・神経活動に関する研究データは日々世界各地で蓄積されている。脳・神経科学と計算機科学を結合させたニューロインフォマティクスとは膨大な研究データを収集し活用できるようにし脳のシステムを解明することを目的としている研究分野である[6]。具体的な研究活動としては、(1)ネットワークを用いて膨大な実験データを集約し研究者間で共有したり、(2) 実験により観測された脳の活動を再現する数理モデルを構築してデータベースに登録したり、(3)コンピュータで脳モデルを実現して情報処理装置としてその本質的特性を解明したりする。本論文は(2)の実験データを用いて神経細胞モデルを構築する研究に関連する。

電気生理学実験を行うためには専門の設備が必要であり、薬品代などランニングコストがかかる。また、それらを扱うための技術を習得する時間も必要となる。それに対し、計算機シミュレーションはコンピュータのみで行えるので、実験に比べてコストがあまりかからない。また、Graphical User Interface(GUI)がしっかりしていて使いやすいシミュレータならば技術習得の時間が短くて済む。従って、電気生理実験で得られた既存の知識・情報を蓄えていて使いやすいシミュレータは神経研究者にとって非常に有効であると考えられる。

神経科学分野で一般的によく使われているニューロシミュレータには NEURON[7], GENESIS[8]がある。しかし、それらはシミュレーションを行うためには特殊なプログラム言語を勉強しなければならなく、また基本的にコマンドを打ち込むことが必要となる。そのため、エキスパートでないと使いづらいシミュレータである。そこで、私が所属する研究室では、神経計算のためのパラメータや、その神経に関するテキストデータを含むデータをデータベースとして用意し、データベース(DB)と連動し、Web 上でシミュレーション出来るシミュレーションの開発を行っている。それは汎用的で使いやすいシミュレータを目指しており、HIPPOSTATION システムと名付けられている。当研究室の吉松によって、その基本が設計され[9]、私が、そのシステムの一部を改訂した[10],[11]。ここで DB を使うのは、チャネルダイナミクスなど生物学的データはとても膨大であり、多量のデータを DB で管理する必要があるからである。

容易にシミュレーションが行えることは初学者に必要なことだと思われる。神経細胞の設計、チャネルの細胞膜への埋め込みをドラッグアンドドロップなど GUI 操作で行え、パラメータを設定しシミュレーションを対話的に行うことができれば、膜電位波形に対するチャネルの役割やパラメータの影響などを容易に短期間で学べると思われる。修士論文では、その部分の仕事を行った[10],[11]。

最終的な HIPPOSTATION システムは、DB サーバ、計算サーバも兼ねている Web サーバ、システムを使うクライアントで構成されている (Figure 1-4)。シミュレーションを行うには、

まず、クライアントがブラウザソフトを用いて DB からチャネルのパラメータや計算式など、必要なデータをダウンロードし神経細胞や神経細胞ネットワークを設計する。そして、Web サーバ上で計算が実行され、そして結果が表示される。

電気生理実験から新しいデータが得られれば、その新しいデータをアップロードしデータを増やしていける。チャネルを 1 つの単位として登録できるシステムを考えており、具体的にはチャネルコンダクタンスを表現できる式やパラメータを DB に登録する事になる。現在、このような式やパラメータは、前述のように計算機神経科学者により求めなければならないか、既存の方法で近似的に解くしかない。私はこの部分を初学者でも利用可能なように自動的に定式化が推定出来る仕組みを開発したいと考えている。

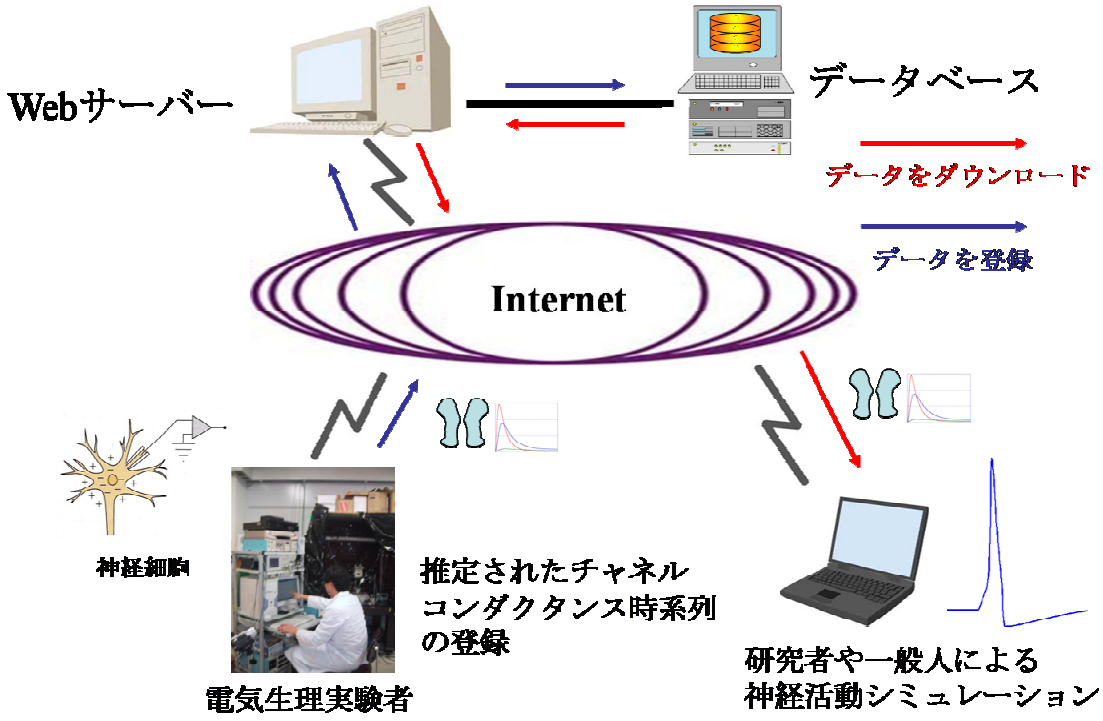

Figure 1-4. 目標となる HIPPOSTATION システム

GUI による直感的で使いやすいシミュレータがあり、実験により得られたチャネルコンダクタンスデータをすぐに神経細胞の計算機シミュレーションに反映できれば、実験家も楽にシミュレーションが行え、チャネルの神経細胞の活動性への影響などすぐに解釈が可能になると考えられる。また、そのような GUI によって簡単にシミュレーションができるのなら、初学者にとっての e-learning にも応用が可能である。

第7節　目的

この研究の目的は電位固定実験により測定できるチャネルコンダクタンスデータを用いてチャネルダイナミクスを推定できるRNNを用いた推定方法を開発することである。そのために、まずRNNを用いて代表的な神経活動の再現が行えるかどうかを確認した。様々なタイプのRNNがあるが、その中で、今回の推定手法に、どのRNNが最適であるか検討した。またRNNの学習に適する電位固定データの検討も行い、膜電位固定法で得られた時間変化データからRNNを用いてチャネルコンダクタンスダイナミックスを自動推定し、神経活動をシミュレーションする最適な方法を開発した。

本論文では例としてヤリイカ巨大軸索のコンダクタンス、及び神経活動を使用した。その神経をSquid Giant(SG)神経と呼ぶ。また、RNNを用いた神経をRNN神経と呼ぶ。しかしながら、実際の膜転位固定法による電気生理実験データを持っていないので、SG神経のデータはHodgkin-Huxley (HH) モデル[1]を計算して得た結果を用いた。すなわち、HHモデルから得られたデータを電気生理実験より得られたデータとみなした。本論文ではRNNを用いたHHモデルを構築しているわけではないので、ご注意して頂きたい。

第2章 方法

第1節　ヤリイカ巨大軸索

ヤリイカの巨大軸索の神経細胞膜電位は Na^+, K^+, リーク電流, 容量性電流を計算することで得られる。Na^+と K^+電流は各々の最大イオンコンダクタンス、ゲート変数と、各イオンの平衡電位と膜電位との差を掛け合わせることで計算できる。Na^+電流は活性化ゲートと不活性化ゲートの 2 つの変数を持ち、それぞれ m と h と呼ばれる。K^+電流は活性化ゲート n のみ持っている。ゲート変数は時間により変化し、膜電位によっても変化する。活性化ゲート変数の定常（収束）値は膜電位と共に上昇する傾向を持つ。逆に、不活性化ゲート変数の定常値は電位と共に減少していく。このヤリイカ巨大軸索の神経活動は HH モデルで計算することが出来、モデルの詳細は付録に示した。

第2節　RNN 神経によるチャネルコンダクタンス計算

RNN を用いてチャネルコンダクタンスの計算を行えるようにするには、電位固定実験によって得られるチャネルコンダクタンスの時間変化を学習する必要がある(Figure 2-1)。入力として膜電位、学習する出力としてコンダクタンスの値とした。チャネルコンダクタンスダイナミックスを学習している RNN を ion-RNN(ion は Na^+もしくは K^+, RNN は TDNN, Elman もしくは FRNN)と呼ぶ。

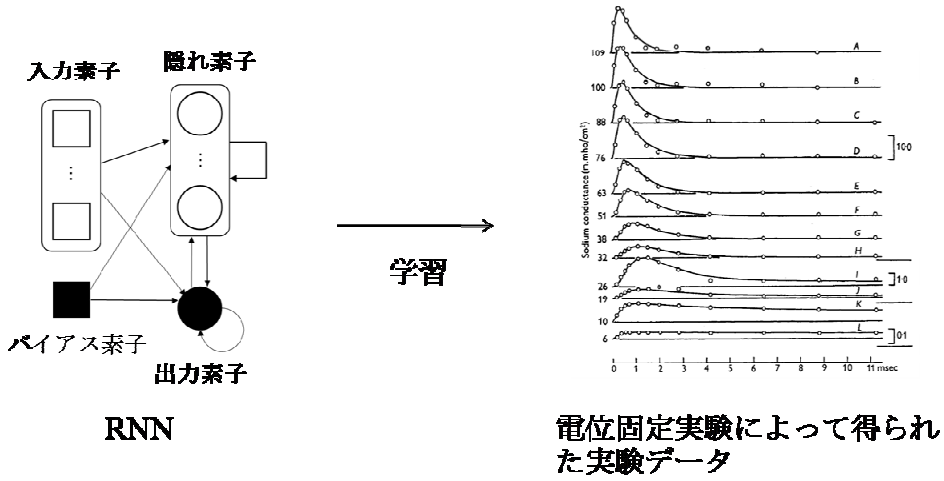

Figure 2-1 RNN による電位固定データの学習

従って、RNN 神経でのチャネルコンダクタンス計算は膜電位$V(t)$等を用いて、コンダクタ

ンスの値 $g(t)$ を出力する仕様となった（Figure 2-2）。

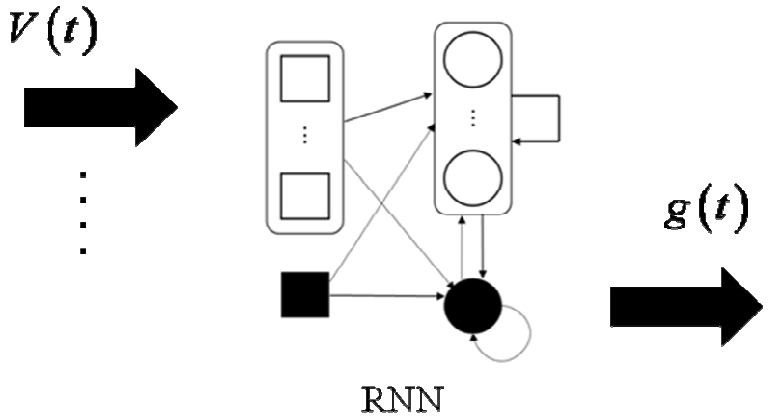

Figure 2-2. RNN によるコンダクタンスの計算 現在の時間の膜電位などを用いてコンダクタンスの値を出力する。

SG 神経と RNN 神経の計算フロー図を Figure 2-3 に示す。

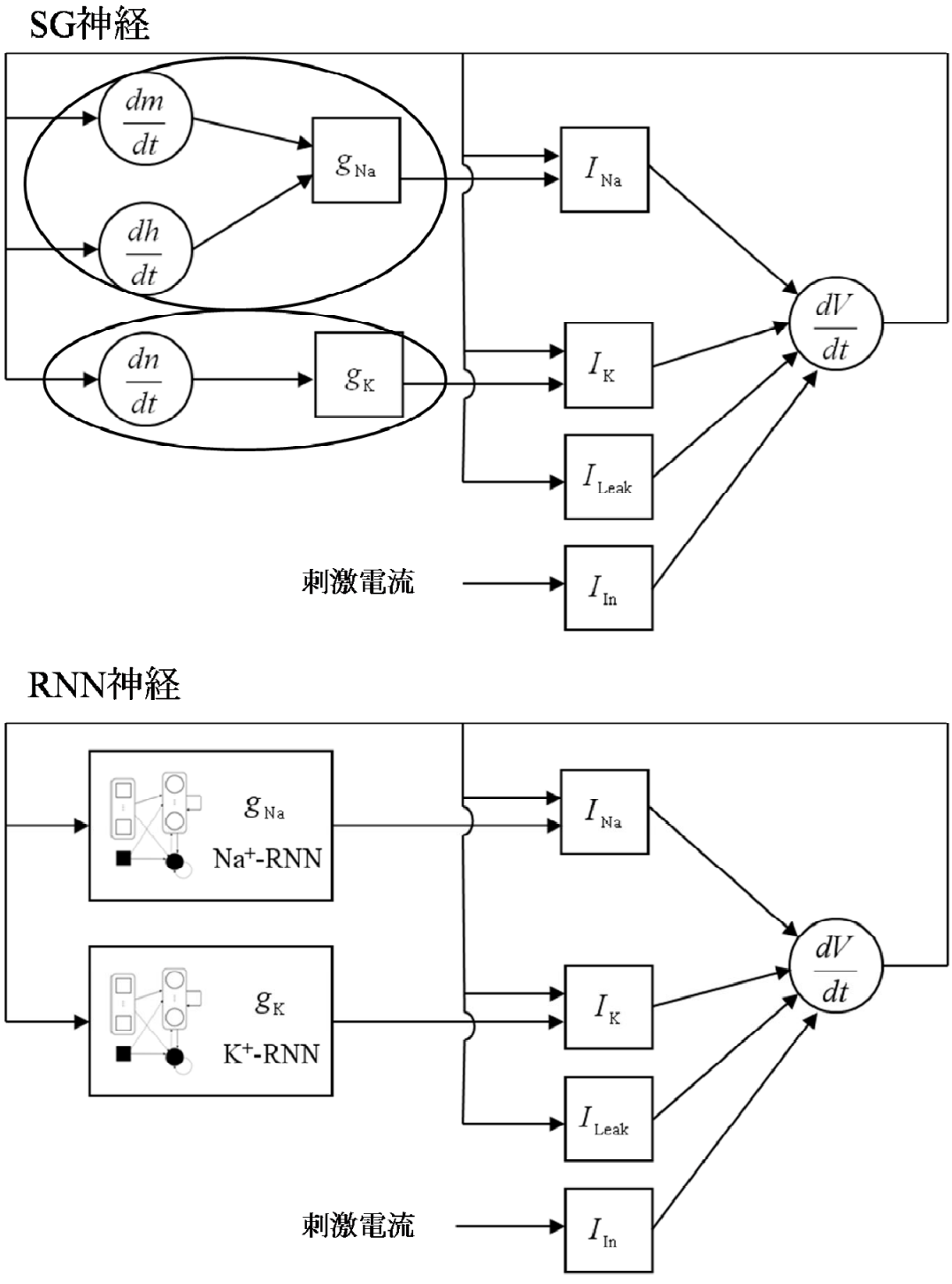

Figure 2-3. SG 神経と RNN 神経の計算フロー SG 神経、RNN 神経共に 1 周する毎に 1 タイムステップが経過する。□や○で示されている変数はその場所でその変数が計算されることを示している。但し○は微分方程式を用いている事を示している。RNN 神経の場合は、長円部分で囲まれている各イオンチャネルコンダクタンスの計算部が RNN と置き換えられる。

第3節　実験に使用するリカレントニューラルネットワーク

神経細胞の本質的な特徴を損なわないで人工的に模擬した素子を考える。その特徴とは神経細胞が他の神経細胞から信号を受け取り膜電位が変化し、それが閾値を超えたら発火して他の神経細胞に信号を送ることである。すなわちその素子は多入力 1 出力の素子となるので、マカローピッツ型素子[12]を用いた。また素子間には生体のシナプス結合に類似した結合を持つ。このような素子が多数結合してニューラルネットワーク(neural network; NN)が形成される。代表的な構造としてフィードフォワード型がある。これは素子を層としてまとめたものを複数層用いたものである。素子間の結合重みを学習アルゴリズムによって調節することで、任意の連続関数の近似[13]、パターン認識[14]、音声処理[15]などに応用されている。さらに再帰的な結合を加えることで時系列データを学習し再現することが可能となる[4],[5]。そのような時系列データを再現できる RNN として時間遅れ NN(time delayed neural network; TDNN) [16]、Elman 型 RNN[17]、完全結合型 RNN(fully connected recurrent neural network; FRNN)[18]等があり、本論文ではこの 3 種類のネットワークを用いた。

第1項　　時間遅れ NN　(Time delayed neural network; TDNN)

時間遅れ TDNN[16]とはいくらかの長さの履歴を用いて次の値を計算する RNN である。Figure 2-4 に図を示した。

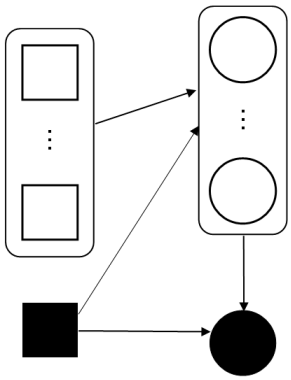

Figure 2-4. 時間遅れ NN □が入力、■がバイアスを示す。また、○が隠れ素子を、●が出力素子を示す。長四角は素子の集合を示す。矢印で結合を示されており、全結合している。

TDNN は、入力層、隠れ層、出力層からなるネットワークである。このネットワークは入力の長さによって履歴をどれだけ長く使うかが決まる。すなわち、必要な長さを試行錯誤で決めなくてはならない。各層は素子の集合から成る。入力層素子から隠れ層素子、隠れ層素子から出力層素子へのフィードフォワードな結合がある。また、隠れ層素子と出力層素子が様々な閾値を持てるように、バイアス素子を追加した。バイアス素子とは一定の出力を常に出力する素子である。本論文出用いるバイアス素子の出力値は-1 とした。なお、出力層素子は 1 つとした。各層素子の出力は以下の式により定義される。

$$y_j(t) = f\left(s_j(t)\right)$$

$$s_j(t) = \sum_{i \in I \cup B} w_{ji} x_i(t)$$

$$z(t) = f\left(u(t)\right)$$

$$u(t) = \sum_{j \in H \cup B} v_j y_j(t)$$

t は時間を示す。時間 t での入力層素子、隠れ層素子、出力素子はそれぞれ $x_i(t)$，$y_j(t)$，$z(t)$ で表わされる。なお、バイアス素子は入力層、隠れ層に 1 つずつ入っている。$s_j(t)$ は隠れ層素子のネット値、$u(t)$ は出力素子のネット値である。I は入力素子を、H は隠れ素子集合を表す。B はバイアス素子を表す。w_{ji} は素子 i から j への結合重みを示す。v_j は隠れ素子 j から出力素子への結合重みを示す。出力素子は 1 つなのでこの素子への重みはそのように表せる。$f(\bullet)$ は隠れ層素子と出力層素子で使われる出力関数で、以下の式で示されるシグモイド関数を用いた。

$$f(\theta) = \frac{1}{1 + \exp(-\theta)}$$

TDNN を学習させるために誤差逆伝播法[19]を用いた。時間 0 から T までの教師信号を学習した。教師信号と TDNN の出力との誤差 E は以下のように定義される。

$$E = \frac{1}{2} \sum_{t=0}^{T} \left(d(t) - z(t)\right)^2$$

$d(t)$ と $z(t)$ は教師信号と出力素子の出力をそれぞれ示す。このアルゴリズムでは誤差を小さくするために結合重みを更新する。更新量 Δw_{ji}、および Δv_j は以下の式を用いて求められる。

$$\Delta v_j = \eta \delta^o y_j(t)$$

$$\Delta w_{ji} = \eta \delta_j^h x_i(t)$$

ここで、η は学習率と呼ばれ、正の定数である。δ^o と δ_j^h は以下の式で計算できる。

$$\delta^o = \left(d(t) - z(t)\right) f'\left(u(t)\right)$$

$$\delta_j^h = f'\left(s_j(t)\right) v_j \delta^o$$

誤差Eが基準値に達するか、指定された回数計算するまで、各更新量を用いて結合重みを更新した。

第2項　　Elman 型 RNN

Elman 型 RNN[17]は隠れ素子がリカレントな結合をもっている。このタイプは TDNN のように入力が決まれば、一意に出力が決まるわけではなく、その再帰的な結合により時系列の学習が可能となる。Figure 2-5 にこの RNN の図を示す。

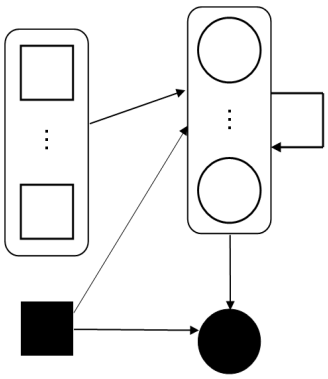

Figure 2-5. Elman 型 RNN

Elman は、TDNN と同様に入力層、隠れ層、出力層からなるネットワークであり、入力層素子から隠れ層素子、隠れ層素子から出力層素子へのフィードフォワードな結合がある。また、バイアス素子も持つ。TDNN と異なる点は隠れ層素子から隠れ層素子への再帰的結合があることである。なお、出力層素子は TDNN 同様 1 つとした。各層素子の出力を以下の式により定義される。

$$y_j(t) = f\left(s_j(t)\right)$$

$$s_j(t) = \sum_{i \in I \cup B} w_{ji} x_i(t) + \sum_{j' \in H} r_{j'j} y_{j'}(t-1)$$

$$z(t) = f\left(u(t)\right)$$

$$u(t)=\sum_{j\in H\cup B} v_j y_j(t)$$

時間tでの入力層素子、隠れ層素子、出力素子はTDNNと同様にそれぞれ$x_i(t)$，$y_j(t)$，$z(t)$で表わされる。なお、バイアス素子は入力層に、隠れ層に1つずつ入っている。$s_j(t)$は隠れ層素子のネット値、$u(t)$は出力素子のネット値である。Iは入力素子を、Hは隠れ素子集合を表す。Bはバイアス素子を表す。w_{ji}は素子iからjへの結合重みを示す。v_jは隠れ素子jから出力素子への結合重みを示す。$f(\bullet)$は隠れ層素子と出力層素子で使われる出力関数で、シグモイド関数を用いた。

Elman型RNNを学習させるためにネットワークを時間方向に展開して誤差逆伝搬法を適用する学習法である通時的逆伝播法[18]を用いた。時間0からTまでの教師信号を学習した。教師信号とElmanの出力との誤差Eは以下のように定義される。

$$E=\frac{1}{2}\sum_{t=0}^{T}\left(d(t)-z(t)\right)^2$$

$d(t)$と$z(t)$は教師信号と出力素子の出力をそれぞれ示す。このアルゴリズムでは誤差を小さくするために結合重みを更新する。更新量Δw_{ji}、$\Delta r_{j'i}$およびΔv_jの式は以下の通りである。

$$\Delta v_j=-\sum_{t=0}^{T}\left(d(t)-z(t)\right)f'\left(u(t)\right)y_j(t)$$

$$\Delta w_{ji}=\sum_{t=0}^{T}\delta_j(t)x_i(t)$$

$$\Delta r_{j'j}=\sum_{t=0}^{T}\delta_{j'}(t)y_j(t-1)$$

ここで、ηは学習率と呼ばれ、正の定数である。$\delta_j(t)$は以下の式を時間に対し逆順で再帰的に計算することで得られる。

$$\delta_j(t)=-f'\left(s_j(t)\right)\left(d(t)-z(t)\right)v_j \qquad (t=T)$$

$$\delta_j(t)=f'\left(s_j(t)\right)\left(-\left(d(t)-z(t)\right)v_j+\sum_{j'=0}^{J}r_{j'j}\delta_{j'}(t+1)\right) \qquad (t\neq T)$$

その誤差が基準値に達するか、指定された回数計算するまで、各結合重みを更新した。

第3項　完全結合型 RNN (Fully connected recurrent neural network; FRNN)

FRNN[18]は隠れ素子と出力素子が完全に相互結合しており、それらの素子へ入力素子か

らの入力があるネットワークである。このネットワークも再帰的な結合を持っているので時系列の学習が可能である。このネットワークを Figure 2-6 に示す。

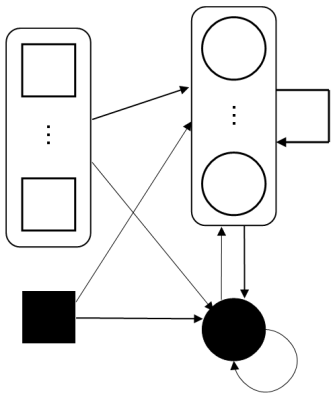

Figure 2-6. 完全相互結合型 RNN

FRNN は、入力素子はすべての隠れ素子と出力素子に結合している。隠れ素子もまたすべての出力素子、他の隠れ素子、及び自分自身に結合している。出力素子もまたすべての隠れ素子と出力素子と自分自身に結合を持っている。また隠れ素子と出力素子が様々な閾値を持てるように、ネットワークにバイアス素子を追加した。各素子の出力を以下の式により定義される。

$$y_j(t) = f\left(s_j(t)\right) \qquad j \in H \cup O$$

$$s_j(t) = \sum_{i \in I \cup B} w_{ji} x_i(t-1) + \sum_{j' \in H \cup O} w_{jj'} y_{j'}(t-1)$$

添え字 i は素子番号を示し、t は時間を示す。I は入力素子を、H は隠れ素子集合を、O は出力素子、B はバイアス素子を示す。$x_i(t)$ は入力素子とバイアス素子の時間 t での出力であり、$y_j(t)$ は隠れ素子、または出力素子の時間 t での出力である。$s_j(t)$ は隠れ素子と出力素子のネット値である。w_{ji} は素子 i から j への結合重みを示す。$f(\bullet)$ は隠れ素子と出力素子で使われる出力関数で、シグモイド関数を用いた。

FRNN の学習には通時的逆伝播法[18]を用いた。そして、時間 0 から T までの教師信号を学習した。教師信号と FRNN の出力との誤差 E は以下のように定義される。

$$E = \frac{1}{2}\sum_{t=0}^{T}\left(d(t) - y_o(t)\right)^2$$

$d(t)$ と $y_o(t)$ は教師信号と出力素子の出力をそれぞれ示す。このアルゴリズムでは誤差を小さくするためにシナプス結合重みを更新する。各素子の出力を以下の式のように $z_i(t)$ としてまとめる。

$$z_i(t)=\begin{cases}x_i(t) & i\in I\cup B\\ y_i(t) & i\in H\cup O\end{cases}$$

すると更新量Δw_{ji}の式は以下のように表現できる。

$$\Delta w_{ji}=-\eta\sum_{t=0}^{T-1}\delta_j(t+1)z_i(t)$$

ここで、ηは学習率と呼ばれ、正の定数である。$\delta_j(t)$は以下の式を時間に対し逆順で再帰的に計算することで得ることができる。

$$\delta_j(t)=f'(s_j(t))(o(t)-d(t)) \qquad (t=T)$$

$$\delta_j(t)=f'(s_j(t))\left((o(t)-d(t))+\sum_{j'\in H\cup O}w_{jj}\delta_i(t+1)\right) \qquad (t\neq T)$$

その誤差が基準値に達するか、指定された回数を計算するまで、結合重みを更新した。

第4項　　RNN のパラメータ

各 RNN のパラメータをここにまとめる。3 種類の RNN に対して、時間tの入力には膜電位$V(t)$、膜電位とコンダクタンスの履歴 5 ステップ分$V(t-i)$，$g(t-i)$(i=1,2,3,4,5)を用いた。つまり入力素子数は 11 であった。RNN には 11 個のデータを入力するが、そのうち V(t)が神経活動の計算に主に使われる入力である。他の入力が RNN の学習のために必要であったので追加した。また出力はコンダクタンスの値$g(t)$とした。出力素子は 1 つとした。3 つのタイプの RNN での違いを明らかにするために、出来るだけ条件を揃えるようにした。例えば、学習するパラメータである結合重み数を同程度にするため、隠れ素子数については TDNN では 20、Elman では 11、FRNN では 10 とした。その結果、結合重み数は TDNN、Elman、FRNN それぞれ 261、265、253 と同程度になった。また、RNN の計算にあたっては、入力と出力は 0.01 から 0.99 に正規化した。すなわち、実際にコンダクタンスを計算するときには膜電位及びコンダクタンスはその範囲に正規化されて入出力した。そして、入出力値は、その範囲で出力されるので、もとのスケールに戻して神経活動の計算では使用した。

第4節　学習用電位固定パターン

神経細胞の膜電位を様々な膜電位に固定した時、測定されるチャネルコンダクタンスの時間変化を RNN の学習パターンとした。本論文では 4 つの RNN 学習用膜電位の固定パターンを準備した。

膜電位固定パターン 1 は、一般的に実験で良く行われる電位固定パターンを用いた時の時間パターンである。すなわち静止膜電位（ここでは-65 mV）から様々な電位にステップ的に固定するパターンである。ステップ電位は-100 から+50 mV まで 10 mV 刻みとした。Figure 2-7 にこのパターンを示す。

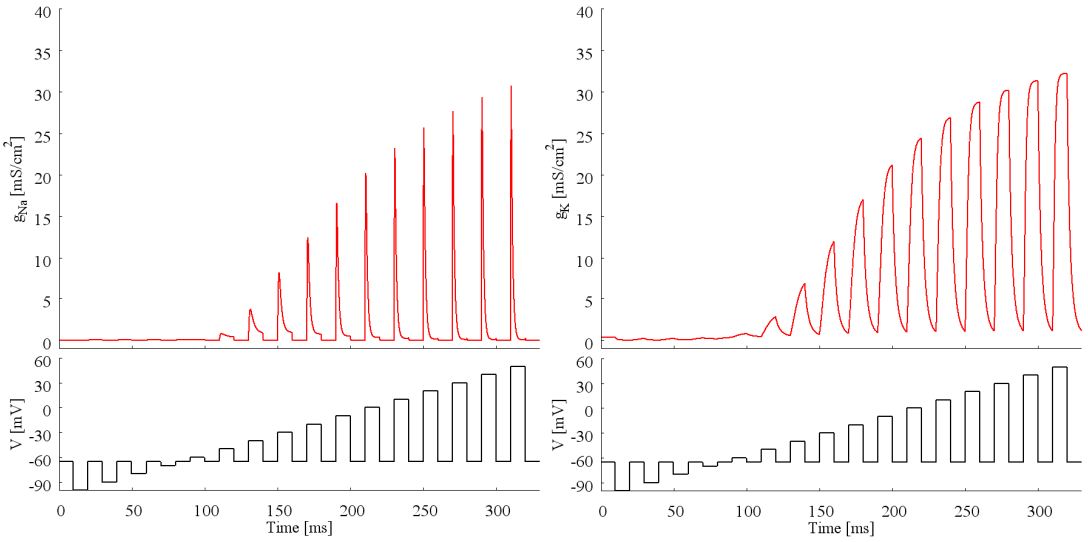

Figure 2-7. RNN の学習に用いた電位固定パターン 1 左図が Na^+、右図が K^+の電位固定パターン及びコンダクタンスの時間変化である。共に上部にコンダクタンスの時間変化、下部に固定電位パターンを示した。

2 つ目の電位固定パターンは、パターン 1 のような静止膜電位からステップ的に電位を変化させたパターンに加え、膜電位を-40 mV から固定したパターンを追加したものである。これは Na^+については不活性化ゲートの影響をより学習するためである。Figure 2-8 にこのパターンを示す。

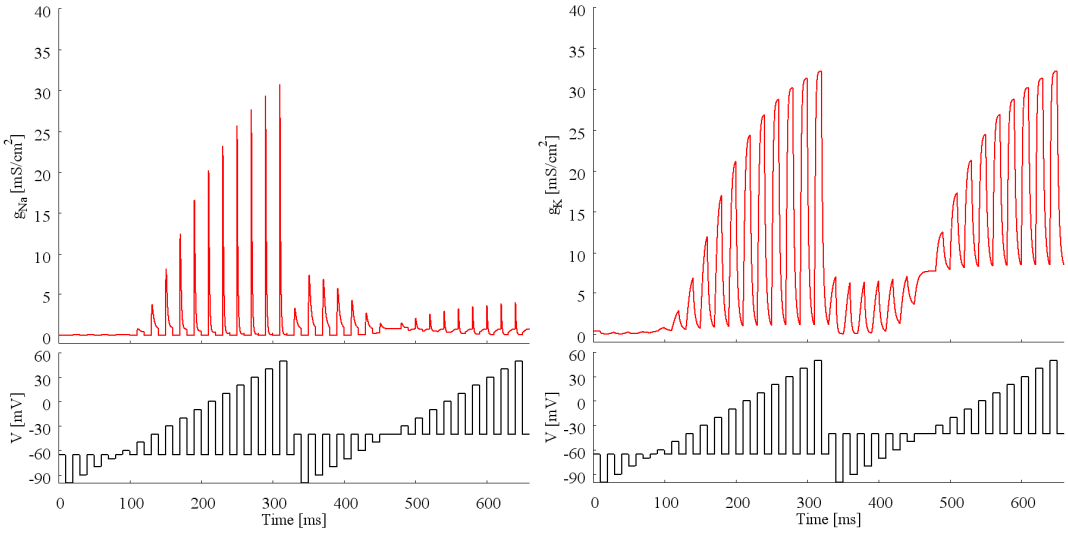

Figure 2-8. RNN の学習に用いた電位固定パターン 2 図の意味は Figure 2-7 と同様である。

図のように、Na^+コンダクタンスの時系列が静止膜電位（-60mV）と-40mV で固定した時で、時間変化が変わっている。

実際に神経活動を計算するとき、膜電位はステップ的ではなく、滑らかに変化する。そこでパターン 1，2 のようにステップ的に膜電位を変化させるのではなく、それらに比べより滑らかに様々な振幅と周波数で電位を固定するパターン 3（Figure 2-9）も用いた。

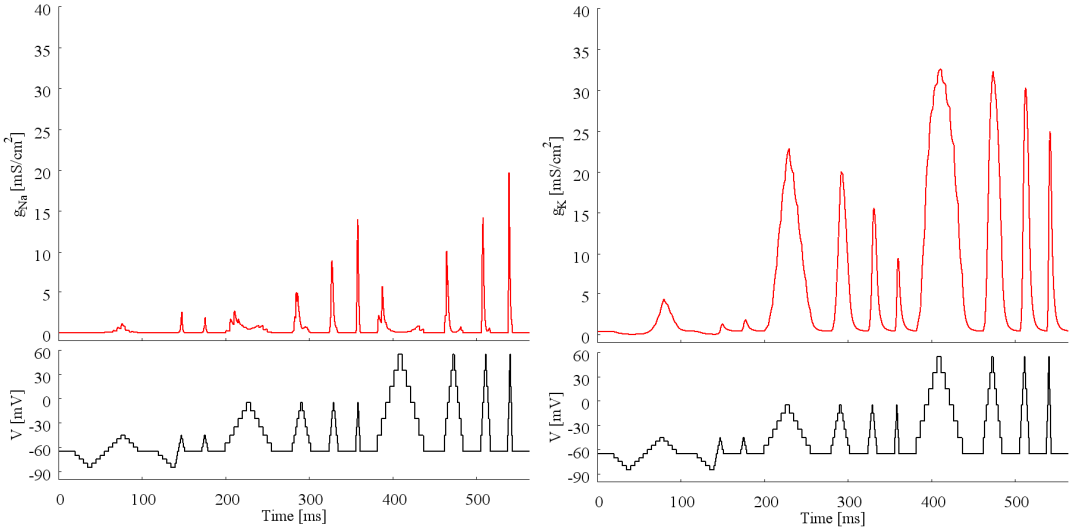

Figure 2-9. RNN の学習に用いた電位固定パターン 3 図の意味は Figure 2-7 と同様である。

電位固定パターン 4 として神経活動時の膜電位波形そのものを入力とした場合のコンダクタンス時間変化を用いた。その波形は閾下反応、活動電位、不応期、リバウンド活動、

周　期　的　発　火　の　信　号　と　し　た　。

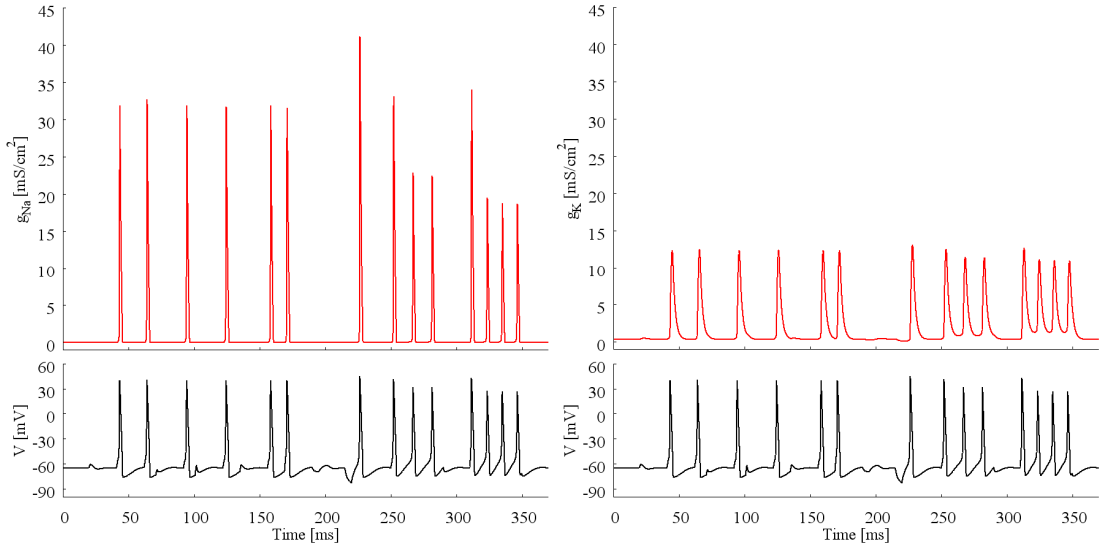

Figure 2-10 にこのデータを示す。

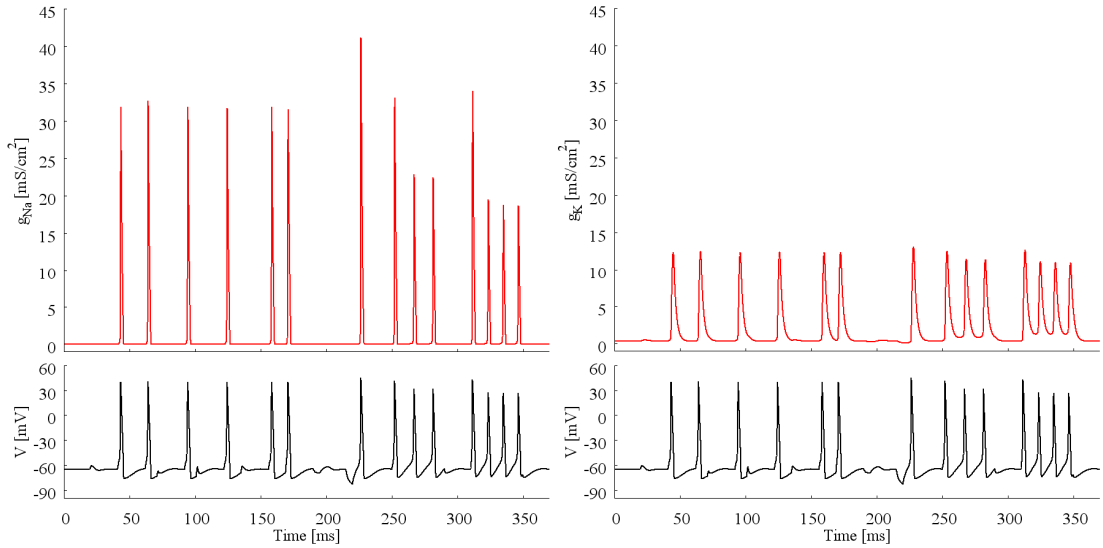

Figure 2-10. 電位固定パターン 4 図の意味は Figure 2-7 と同様である。

第5節　テスト用電位固定パターン

RNN が学習した後、学習していないデータに関してもコンダクタンスの時間変化を再現出来るかをテストした。そのような RNN の汎化能力を調べるため、それぞれの電位固定パターンの分布が等しい、すなわち平均値と分散が等しいようなテスト用電位固定パターンを合計 8 つ用意しテストに用いた。Figure 2-11 には電位固定パターン 1～4 までの固定電圧値の頻度分布を示した。

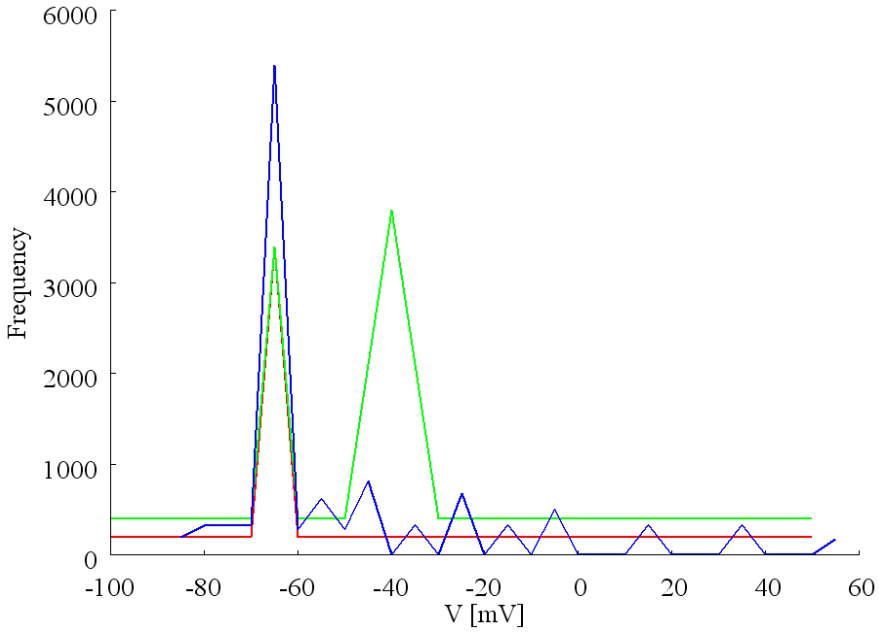

Figure 2-11. 電位固定パターンの電圧値の頻度分布　赤線が電位固定パターン 1、緑線が電位固定パターン 2、青線が電位固定パターン 3 である。

この分布電圧値の頻度分布が同じになるように各電位固定パターンからテストパターンを 2 個ずつ作成し計 8 つのテストパターンを使用した。それらのテストパターンを、例えばパターン 3 から作成されたものであれば、3_1、3_2 と表した(Figure 2-12)。

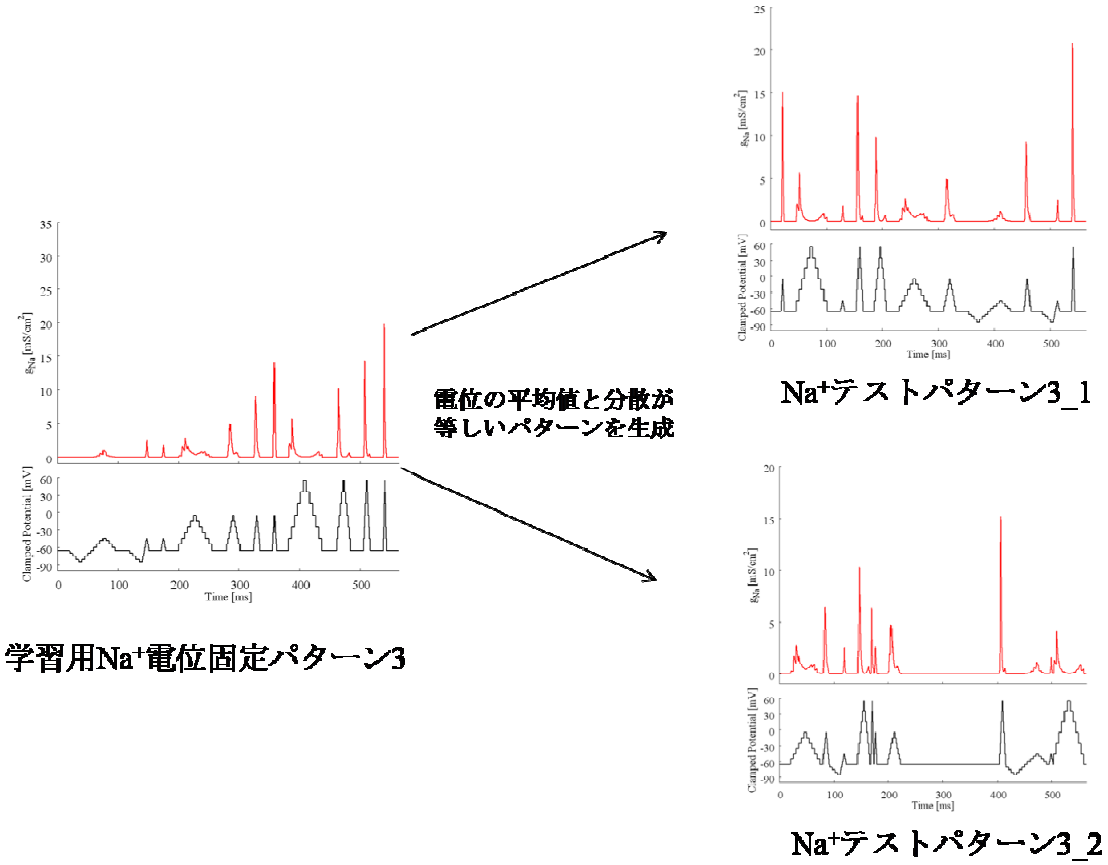

Figure 2-12 テストパターンの生成例（Na^+のパターン 3 の場合)

実験に使用したテストパターンを Figure 2-13 から Figure 2-20 に示す。テスト時、RNN には膜電位固定するパターンを入力し、出力されるコンダクタンスの時間変化が正しいかどうかで RNN の汎化能力を調べた。

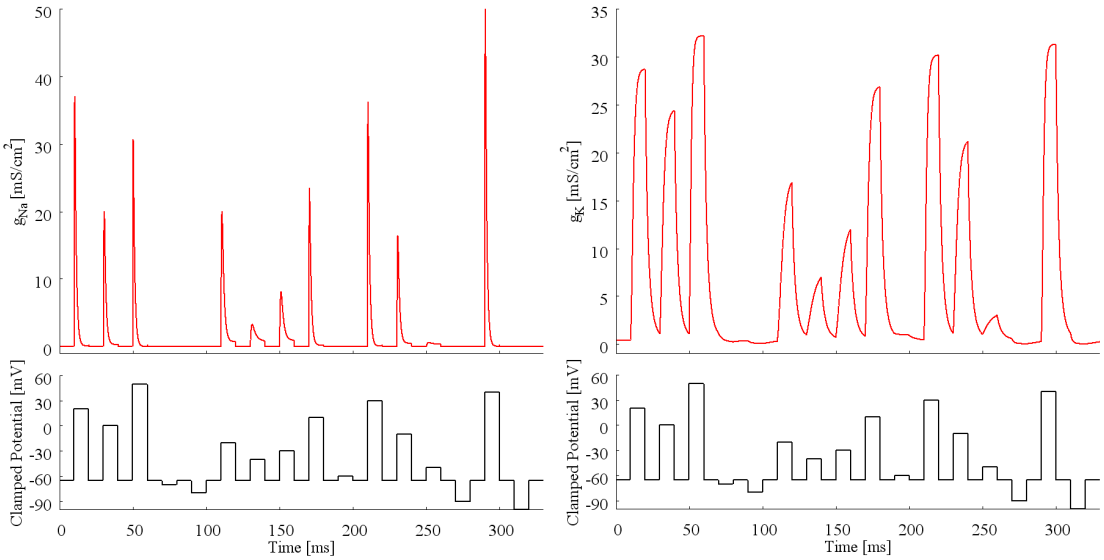

Figure 2-13. テストパターン 1_1 左図に Na^+、右図に K^+のコンダクタンス時系列が示されている。

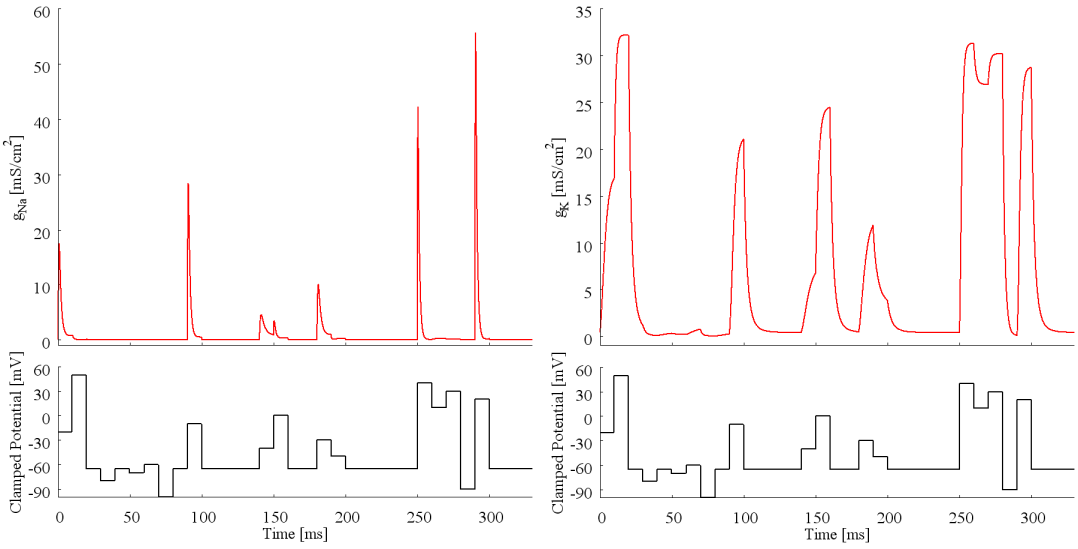

Figure 2-14. テストパターン 1_2

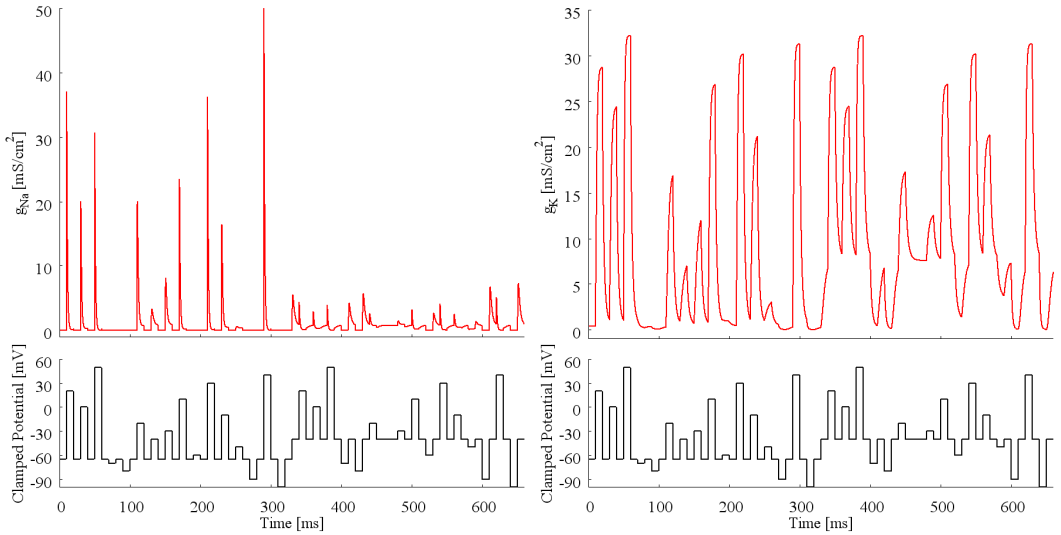

Figure 2-15. テストパターン 2_1

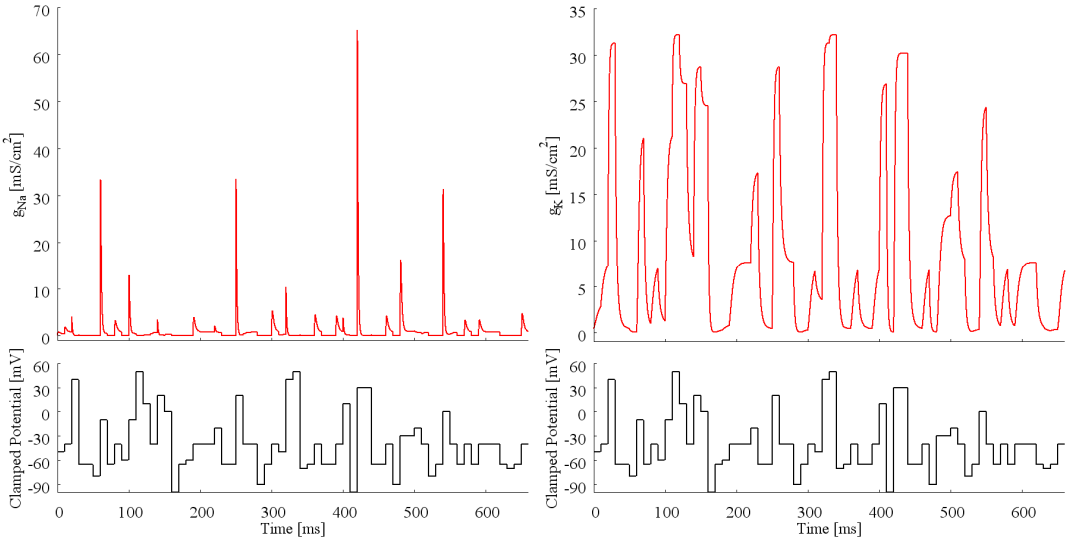

Figure 2-16. テストパターン 2_2

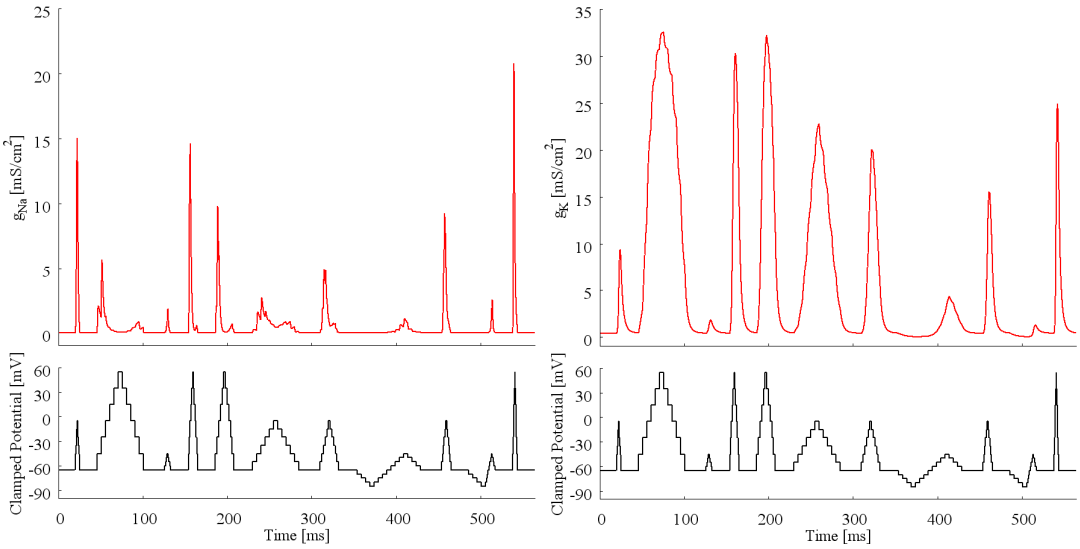

Figure 2-17. テストパターン 3_1

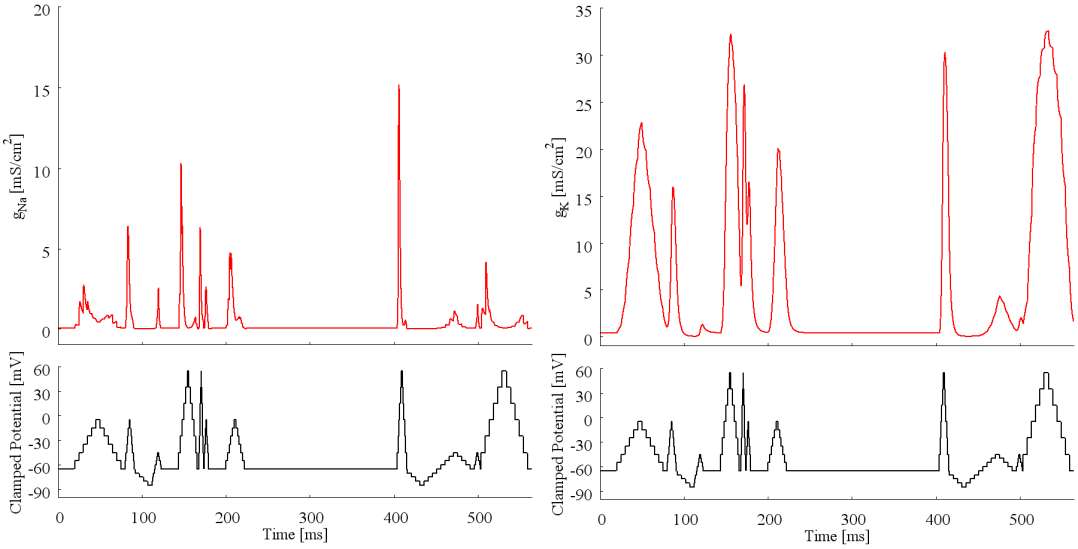

Figure 2-18. テストパターン 3_2

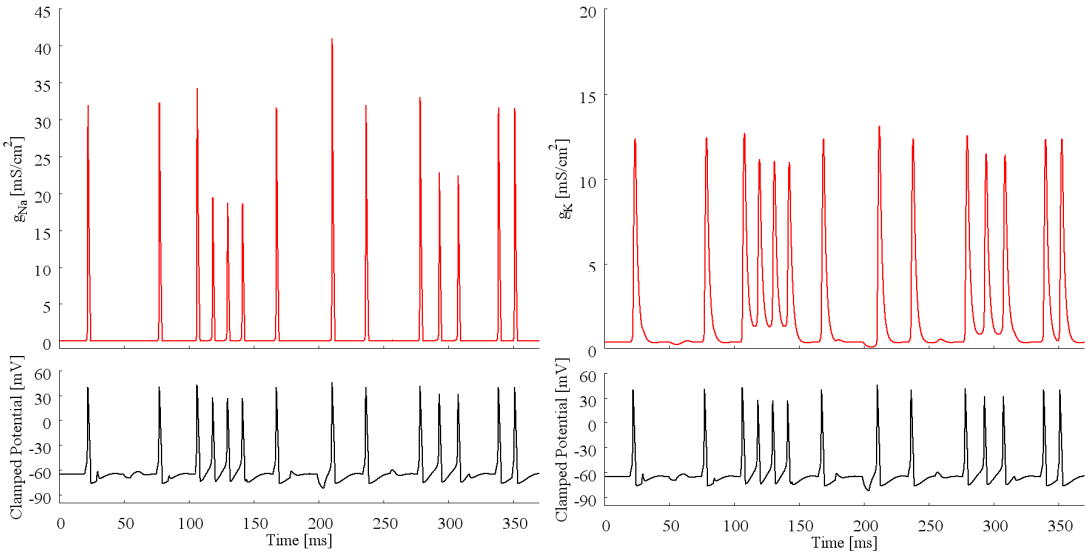

Figure 2-19. テストパターン 4_1

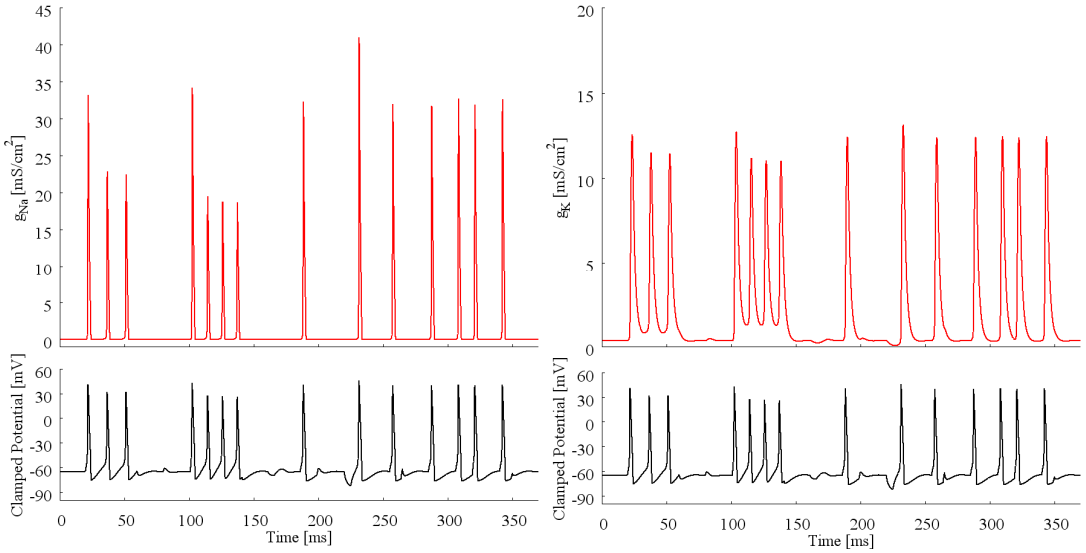

Figure 2-20. テストパターン 4_2

第6節　生成した RNN 神経

前述したように、RNN（TDNN, Elman 型、FRNN）の学習には BP 法[19]、若しくは BPTT 法[18]を用いた。RNN の学習を行う場合、ランダムな結合重みからスタートし学習させる。従って学習させる電位固定パターンが同じであっても、同じ回数学習したとしても初期の結合重みが異なるので同じ結果になるとは限らない。RNN の種類による RNN 神経の活動に対する影響や異なる電位固定パターンによる RNN 神経活動への影響を比較するためには、

初期の結合重みを変えて学習させた RNN を複数生成し、それらを組み合わせた RNN 神経活動の結果から判断する必要がある。チャネルコンダクタンスダイナミックスを学習している RNN を ion-RNN(ion は Na^+もしくは K^+, RNN は TDNN, Elman もしくは FRNN)と呼ぶ。

どのタイプの RNN が SG 神経活動を再現可能なのかを調べるため、電位固定パターン3(Figure 2-9)を用いて、TDNN、Elman、FRNN の 3 種の RNN で Na^+-RNN と K^+-RNN を各々10 個ずつ生成した。このときの RNN 生成のイメージを Figure 2-21 に示す。

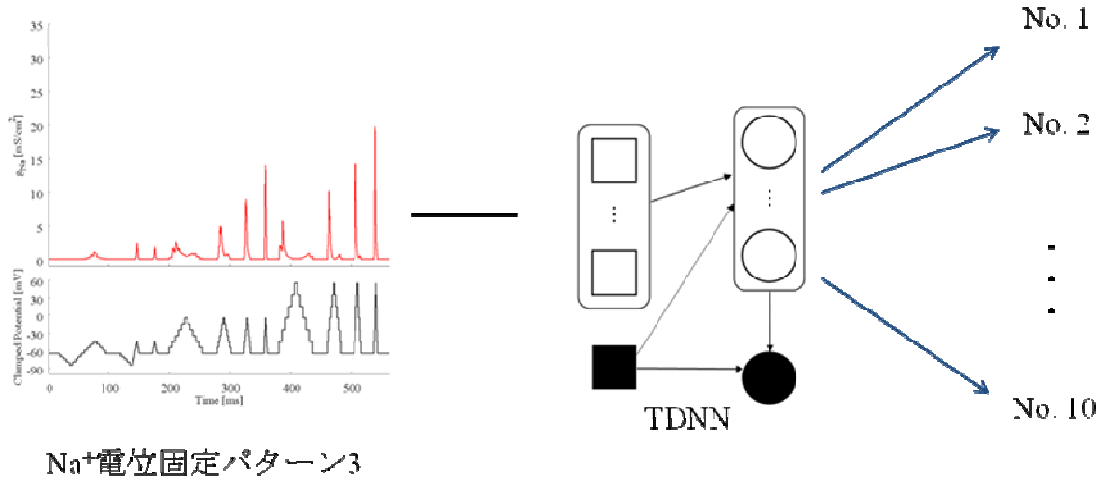

Figure 2-21 ion-RNN の生成イメージ図（Na^+-TDNN の場合） Na^+の電位固定パターン 3 を用いて 10 個の Na^+-TDNN を生成した。

そして、それらを組み合わせて 100 個の RNN 神経を生成し神経活動の性質を調べた。このときの RNN 神経生成のイメージを Figure 2-22 に示す。

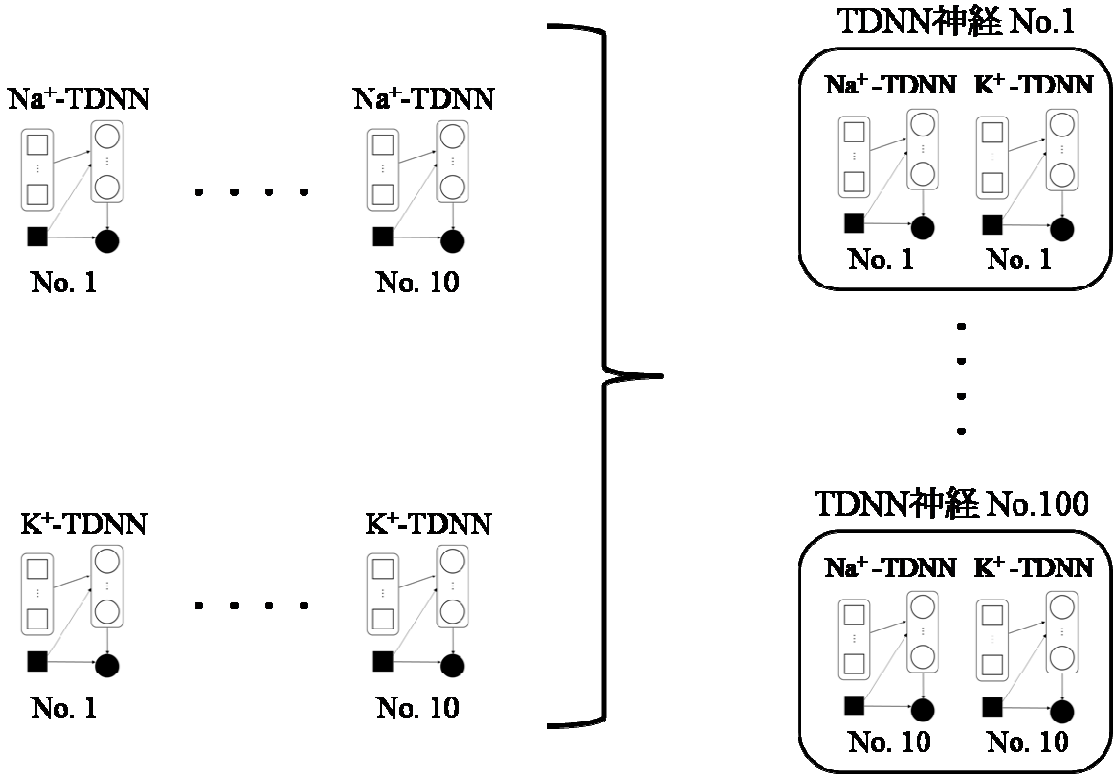

Figure 2-22 RNN 神経の生成イメージ図（TDNN 神経の場合） Na^+-TDNN と K^+-TDNN を組み合わせて 100 個の TDNN 神経を生成した。

またどの電位固定パターンを学習した RNN が SG 神経活動を再現可能なのかを調べるために、RNN として FRNN を用いて、パターン 1 から 4 までの 4 つの電位固定パターンに対して各 10 個の Na^+-FRNN と K^+-FRNN を生成し、それらを組み合わせて 100 個の FRNN 神経を生成した。

第7節　RNN 神経による神経活動の評価方法

Na^+-RNN と K^+-RNN と組み合わせて作成した RNN 神経の神経活動を SG 神経と比較するために、一つの評価方法として、それら 2 つの神経から得られる神経活動の膜電位変化の差から計算できる最小自乗誤差を用いることが考えられる。しかしながら、RNN 神経の神経活動は SG 神経のものと比較すると、発火の位相がずれた結果となる場合がある。このとき、最小自乗誤差では誤差がとても大きくなってしまい正確な違いの評価にならない。すなわち、人間の目で判断する場合と異なってしまう。この場合の例を Figure 2-23 に示す。

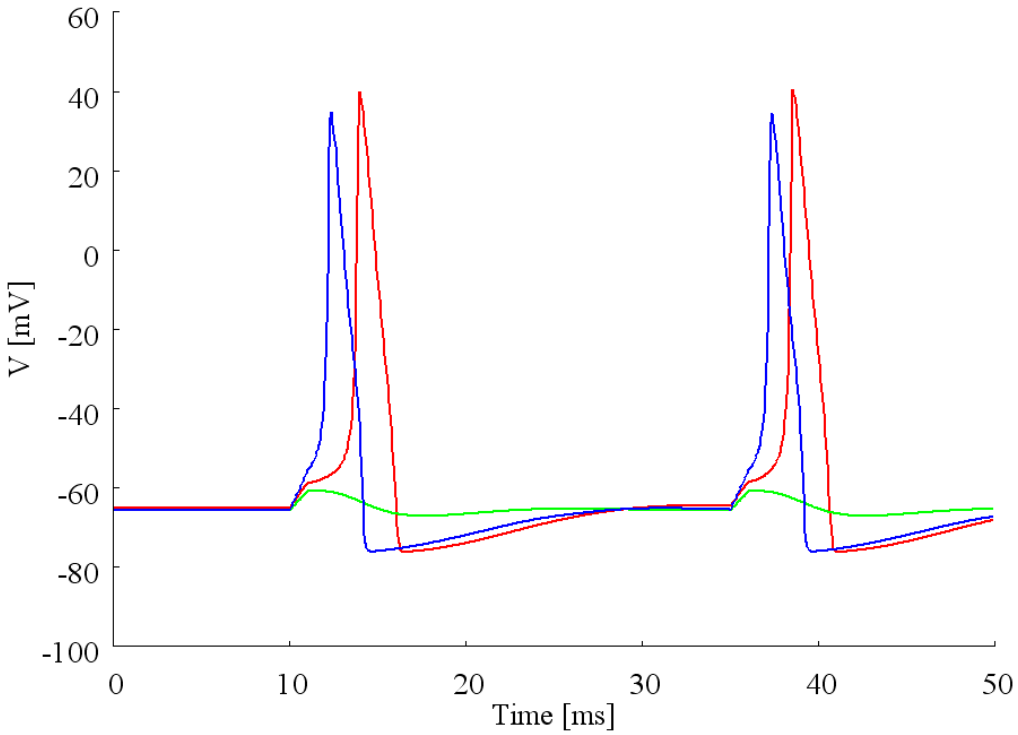

Figure 2-23　位相がずれた場合の例　赤線が SG 神経、青線と緑線が RNN 神経の活動を示している。この場合 MSE で比較すると緑線のほうが青線よりも赤線に近いと判断されてしまう。

そこで本研究における神経活動の比較には、2 つの時間変化（時系列）間の類似度を計ることが出来る相互相関関数を用いる事にした。特に相互相関関数の最大値を、2 つの信号間の類似を計る評価値として採用した。この値が 1 に近いほど 2 つの時系列は似ている事になる。以上の方法を用いれば、Figure 2-23 のような場合でも人間の判断とよく合う評価ができた。すなわち緑線よりも青線のほうが赤線に近いという評価ができた。

第8節　SG 神経の計算方法

本論文では、HH モデルを計算したデータを実験から測定したデータとみなした。すなわち、RNN の学習に用いる 4 つの電位固定パターンや SG 神経の神経活動は HH モデルを 4 次のルンゲクッタ法で計算することで得た。その際、時間刻み幅は 0.05 ms とした。

第9節　計算環境

RNN の学習や、神経活動の計算に用いた計算機の性能について Table 2-1 にまとめる。

Table 2-1 計算機の性能

OS	Microsoft Windows XP Professional Version 2002 Service Pack 3
CPU	Core2 Duo E6750 2.66GHz
メモリ	2GB
ディスク	256GB

第3章 結果

第1節　コンダクタンスの時間変化

第1項　　3 つの RNN の比較

TDNN, Elman 型, FRNN の 3 種の RNN 間のうち、神経活動を再現するための RNN はどれか評価するために、電位固定パターン 3 を 3 つの RNN に学習させ、その再現性を比較した。

1. コンダクタンス再現例

Na^+-TDNN と K^+-TDNN が電位固定パターン 3 を学習した時の最終結果を示す(Figure 3-1)。SG 神経の Na^+チャネル、K^+チャネルのコンダクタンスを再現する事が出来た。

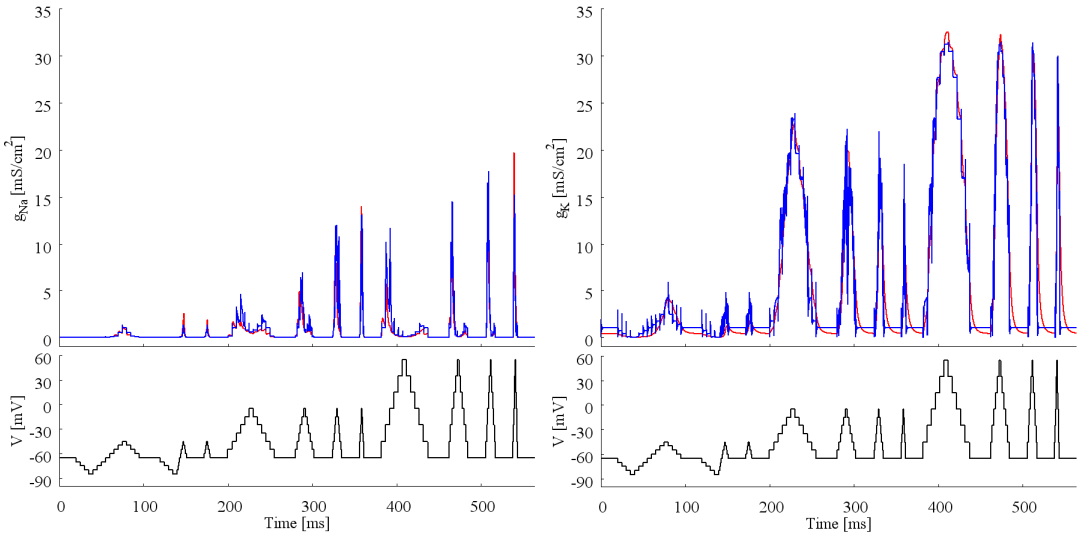

Figure 3-1. TDNN を用いたときのコンダクタンス再現結果　赤線が ion-RNN の学習用に用いたコンダクタンス、青線が ion-RNN からの出力を表す。左が Na^+-TDNN、右が K^+-TDNN である。

Na^+-ElmanRNN と K^+-ElmanRNN が電位固定パターン 3 を学習した時の最終結果を示す(Figure 3-2)。SG 神経の Na^+チャネル、K^+チャネルのコンダクタンスを再現する事が出来た。

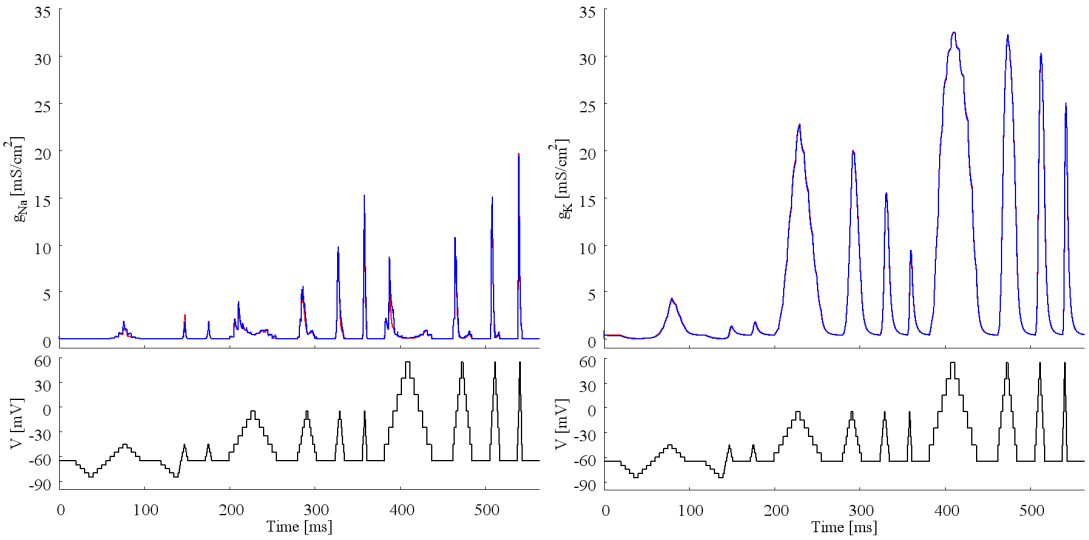

Figure 3-2. Elman を用いた場合のコンダクタンス再現結果　赤線が ion-RNN の学習用に用いたコンダクタンス、青線が ion-RNN からの出力を表す。左が Na^+-Elman、右が K^+-Elman である。

Na^+-FRNN と K^+-FRNN が電位固定パターン 3 を学習した時の最終結果を示す(Figure 3-3)。SG 神経の Na^+チャネル、K^+チャネルのコンダクタンスを再現する事が出来た。すなわち、RNN が学習後、膜電位パターンを RNN に入力した時、RNN は対応するチャネルコンダクタンスの時間変化を生成した。但し、100 ms から 200 ms の間、Na^+コンダクタンスでは違いがあった。

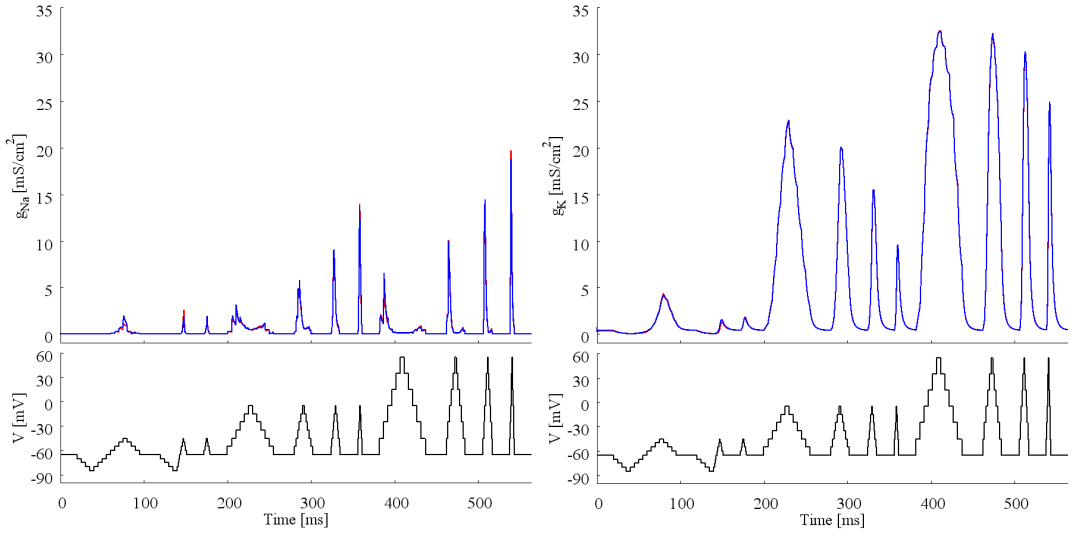

Figure 3-3. FRNN を用いた場合のコンダクタンス再現結果

2. 学習中の誤差値比較

3種のRNNを用いて、Na^+コンダクタンス学習中の誤差の経時変化をFigure 3-4に示した。10個の Na^+-RNN について全て図示した。

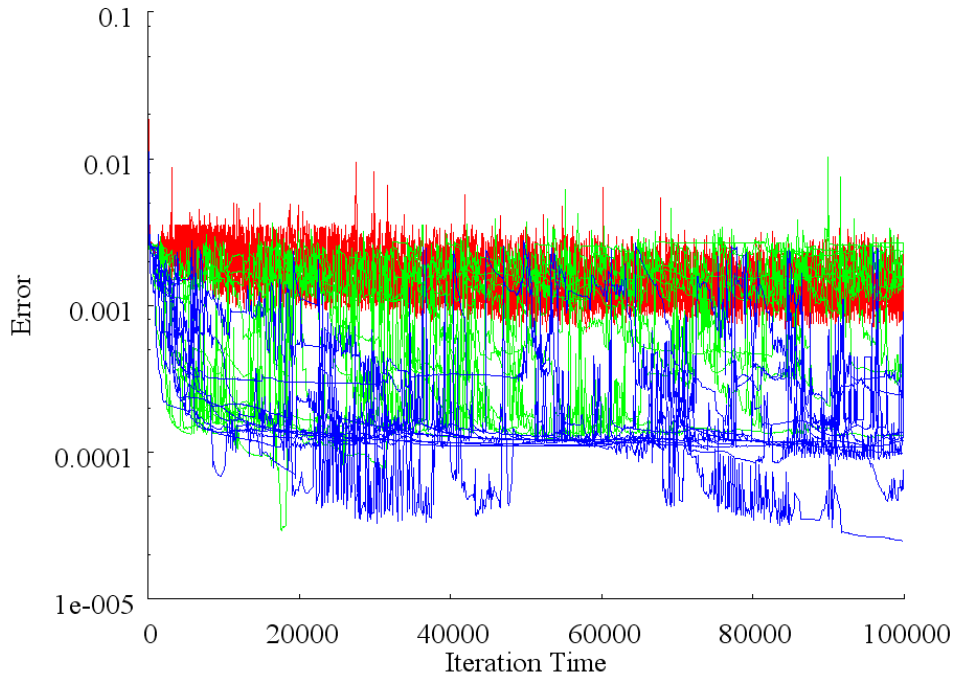

Figure 3-4. 各 RNN で Na^+コンダクタンスを学習中の誤差変化　赤線が TDNN、緑線が Elman、青線が FRNN の誤差変化を示している。横軸がイテレーションタイムで縦軸が最小自乗誤差である。また、グラフは片対数グラフである。

神経活動の計算に用いた RNN は誤差が最小値の時のものを用いた。そのときの誤差の平均値±標準誤差を Table 3-1 に示した。10 例の平均である。また、棒グラフで表した結果を Figure 3-5 に示した。Na^+-TDNN の結果がほかの 2 つに比べて有意に誤差が高く（****p<0.001, Steel-Dwass 法）、また FRNN、Elman の誤差は TDNN とくらべて小さいという結果だった。

Table 3-1 各 RNN による最小誤差の平均値（Na^+コンダクタンス）

RNN 名	最小誤差の平均値
TDNN	7.48E-04±1.90E-05 ****
Elman	1.43E-04±2.66E-05
FRNN	8.39E-05±1.35E-05

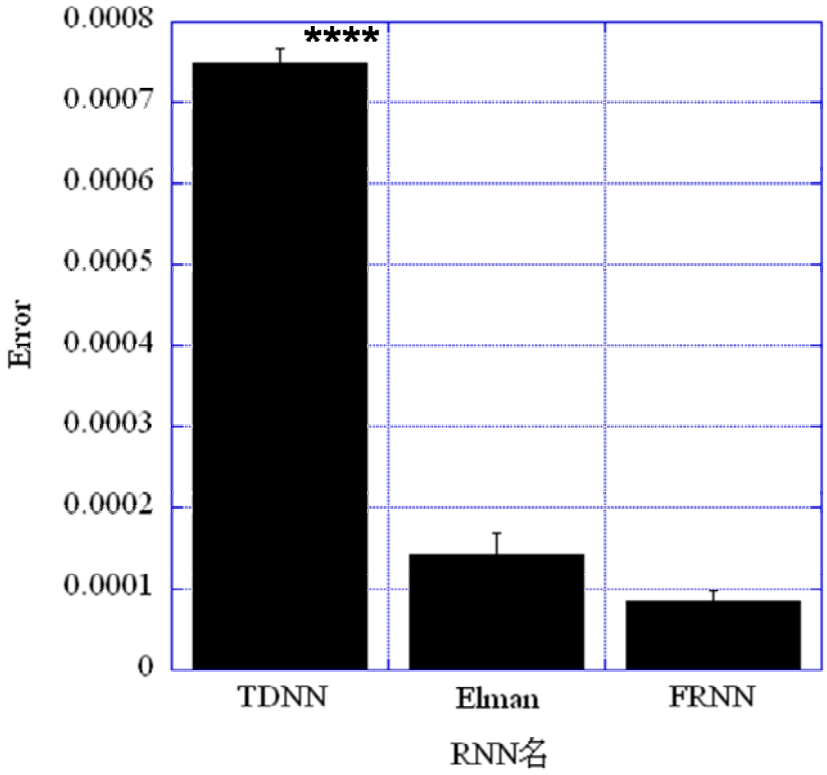

Figure 3-5 各 RNN による最小誤差の平均値の棒グラフ（Na^+コンダクタンス）

次に K^+コンダクタンス学習中の誤差の変化を Figure 3-4 に示した。これも同様に 10 例分すべて図示した。

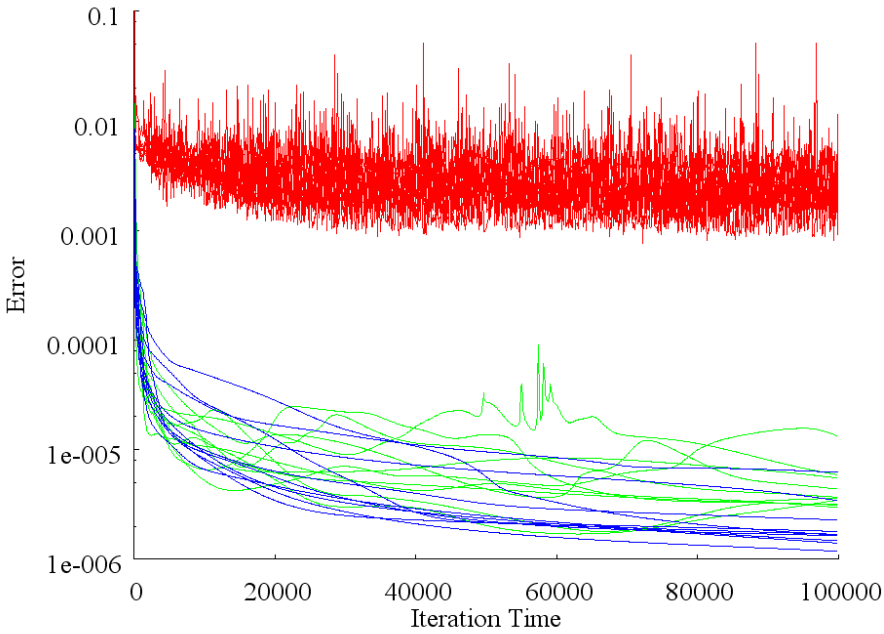

Figure 3-6. 各 RNN を用いて K^+コンダクタンスを学習中の誤差変化 グラフの色の説明は Figure

3-4 のものと同様である。

そのときの誤差の平均値±標準誤差を Table 3-2 に示し、また棒グラフで Figure 3-7 に示した。Na^+のときと同様に TDNN の誤差が有意に大きくて（****p<0.001, Steel-Dwass 法）、FRNN の誤差が他の 2 つの RNN に比べ有意に小さかった（*p<0.05, Steel-Dwass 法）。

Table 3-2 各 RNN による最小誤差の平均値（K^+コンダクタンス）

RNN 名	最小誤差の平均値
TDNN	1.41E-03±1.81E-04 ****
Elman	3.59E-06±4.10E-07
FRNN	2.27E-06±4.54E-07 *

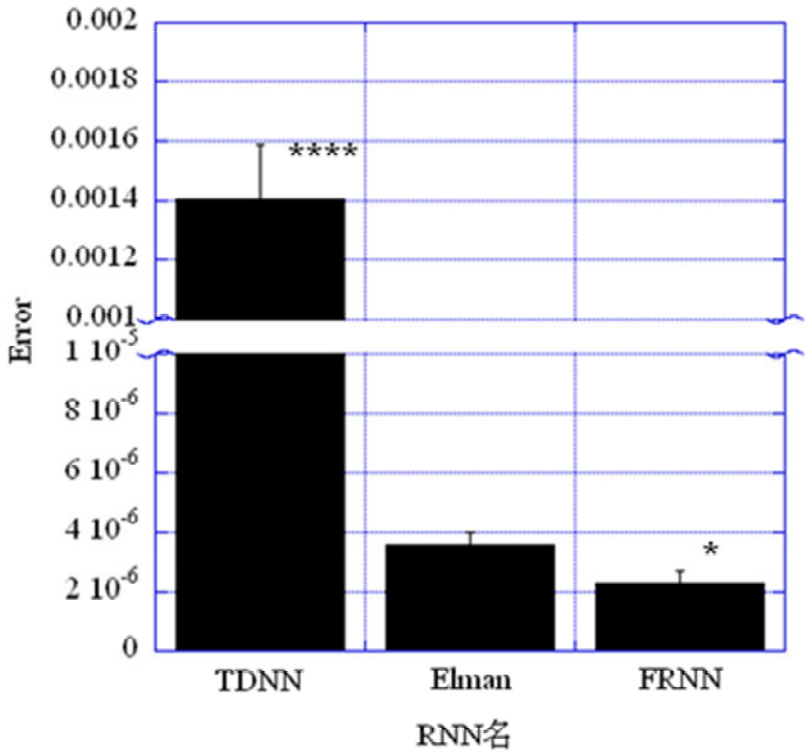

Figure 3-7 各 RNN による最小誤差の平均値の棒グラフ（K^+コンダクタンス）

3. 学習時に用いていないテストパターンを RNN に与えた時の誤差値の比較

RNN の学習に用いた電位固定パターンと平均値、分散が等しいデータを 8 つ用意しテストに用いた。つまり、Na^+チャネル、K^+チャネルのダイナミックスには従うが、RNN が学

習していない時間変化（時系列）に関して、RNN が再現可能かどうかを調べた。その結果を Figure 3-8 に示した。

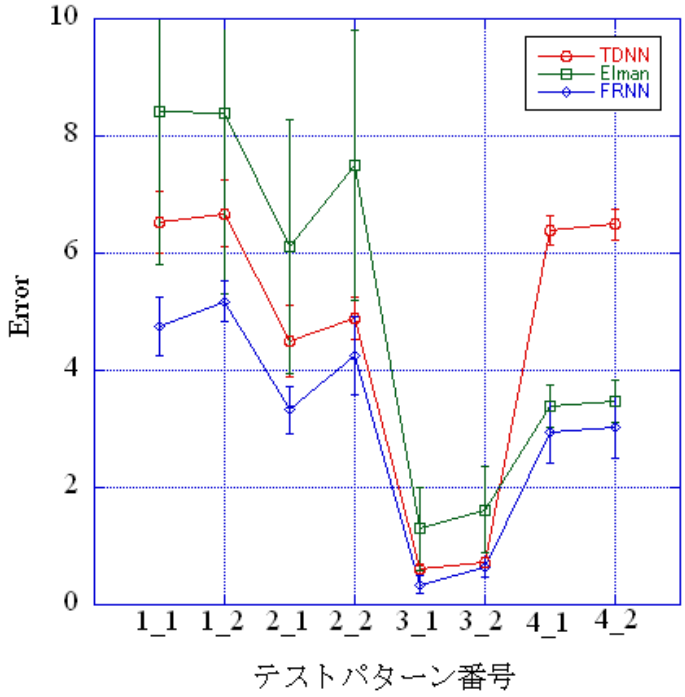

Figure 3-8 Na^+コンダクタンスの RNN 別テスト結果　縦軸は、最小自乗誤差であり、横軸は、テストデータ番号（Figure 2-13~Figure 2-20）である。赤は TDNN、緑は Elman、青は FRNN を用いた結果である。

有意差は出なかったものの、FRNN が 8 つのテストパターンすべての場合で誤差が小さいという結果となった。

次に、K^+コンダクタンスについてのテスト結果を示した。

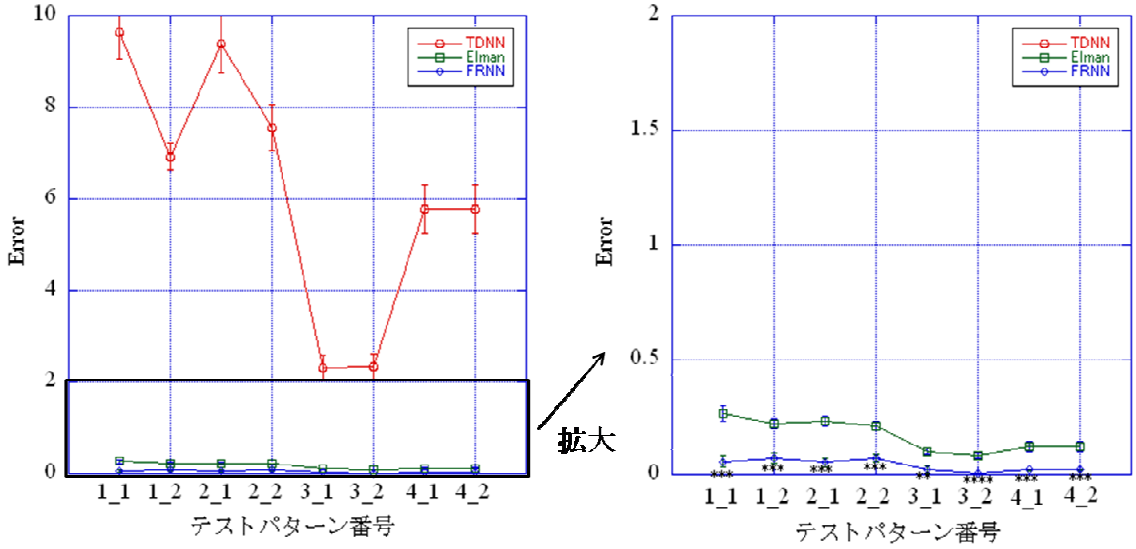

Figure 3-9 K^+コンダクタンスの RNN 別テスト結果 右図は左図の拡大図である。

拡大図で確認できるように、全てのテストパターンにおいて有意に FRNN の誤差が小さかった。 (**p<0.01, ***p<0.005, ****p<0.001; Steel Dwass 法)。

以上の結果より、FRNN は学習させた時間変化そのものだけでなく、Na^+チャネル、K^+チャネルの各チャネルダイナミックスを学習していると考えられる。3 種類の RNN の中では、各チャネルダイナミックスを学習する FRNN を神経活動の計算に用いれば良いと考えられる。

第2項　　RNN に学習させる電位固定パターンはどれが良いのか？

4 種の電位固定パターンのうち、RNN の学習にはどのパターンが良いのか比較するため、用いる RNN を FRNN に固定して、4 つの電位固定パターンを用いて FRNN を学習させた。

1. チャネルコンダクタンスの再現

電位固定パターン 1 を用いて学習させた Na^+-FRNN と K^+-FRNN の出力結果の 1 例を示す (Figure 3-10)。

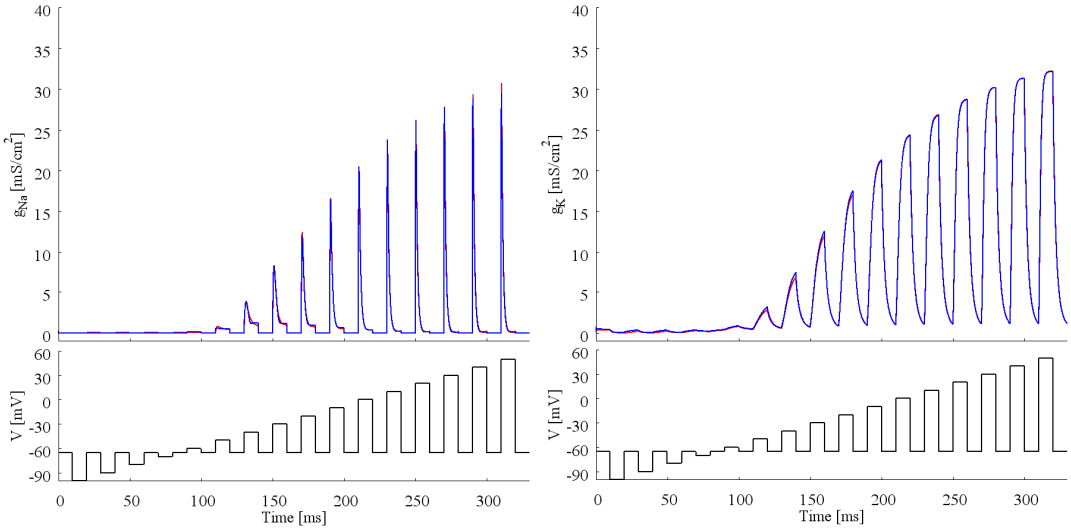

Figure 3-10. 電位固定パターン 1 を用いた時のコンダクタンス再現結果　左が Na^+チャネル、右が K^+チャネルの結果である。上段がコンダクタンス、下段が電位固定パターンである。このグラフから Figure3-11 までは同じフォーマットのグラフである。

電位固定パターン 2 を用いて学習させた Na^+-FRNN と K^+-FRNN の出力結果の 1 例示す (Figure 3-11)。

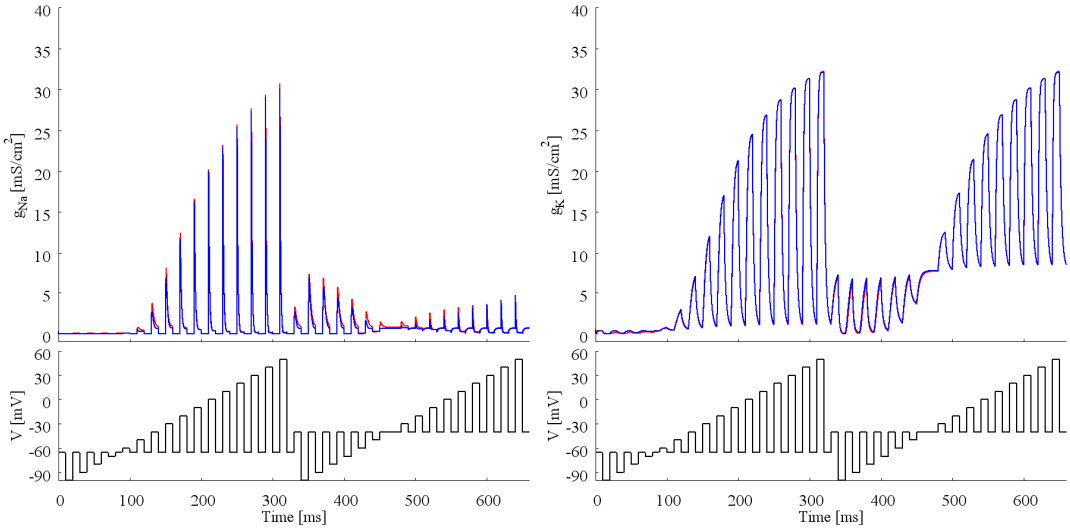

Figure 3-11. 電位固定パターン 2 を用いた時のコンダクタンス再現結果

電位固定パターン 3 を用いて学習させた Na^+-FRNN と K^+-FRNN の出力結果の 1 例示す (Figure 3-11)。

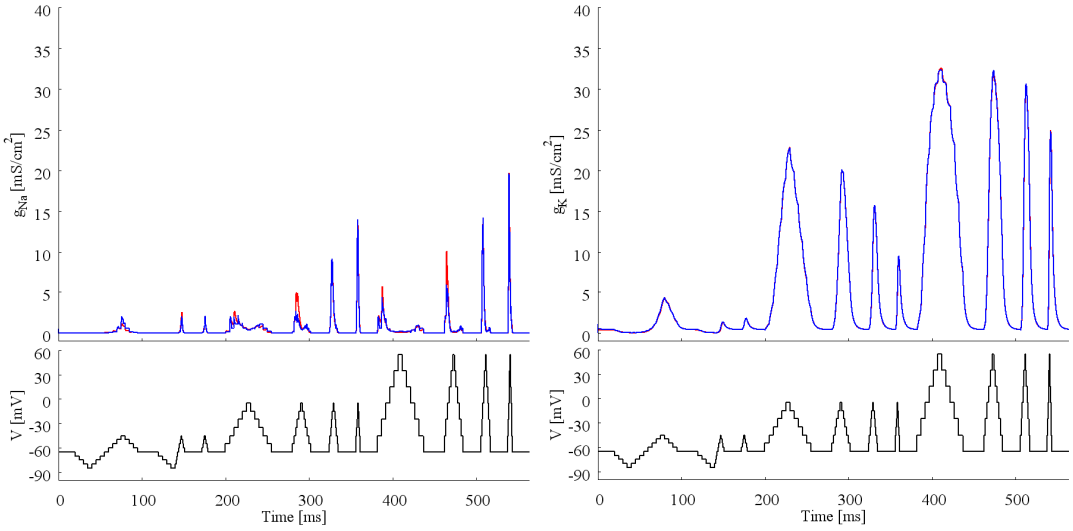

Figure 3-12. 電位固定パターン 3 を用いた時のコンダクタンス再現結果

電位固定パターン 4 を用いて学習させた Na^+-FRNN と K^+-FRNN の出力結果の 1 例示す (Figure 3-13)。

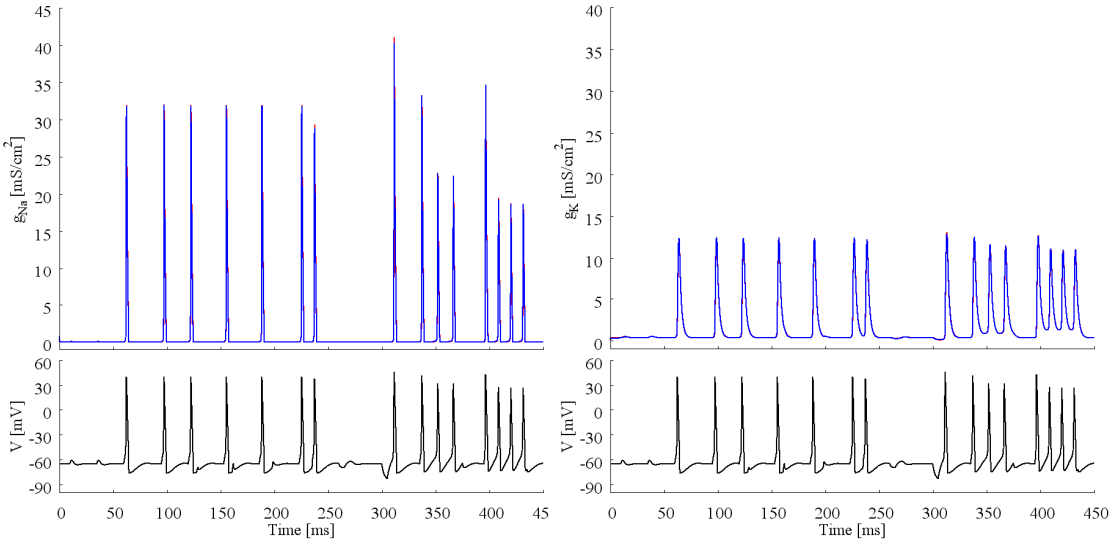

Figure 3-13. 電位固定パターン 4 を用いたときのコンダクタンス再現結果

4 つ電位固定パターンの全てにおいて定性的にはコンダクタンスの再現が出来た。

2. 4 種の電位固定パターンを用いた学習中の誤差値比較

各電位固定パターンを用いて、Na^+コンダクタンス学習中の RNN の出力信号と教師信号

との誤差を 4 種の電位固定パターン各 10 例分全て Figure 3-14 に示した。

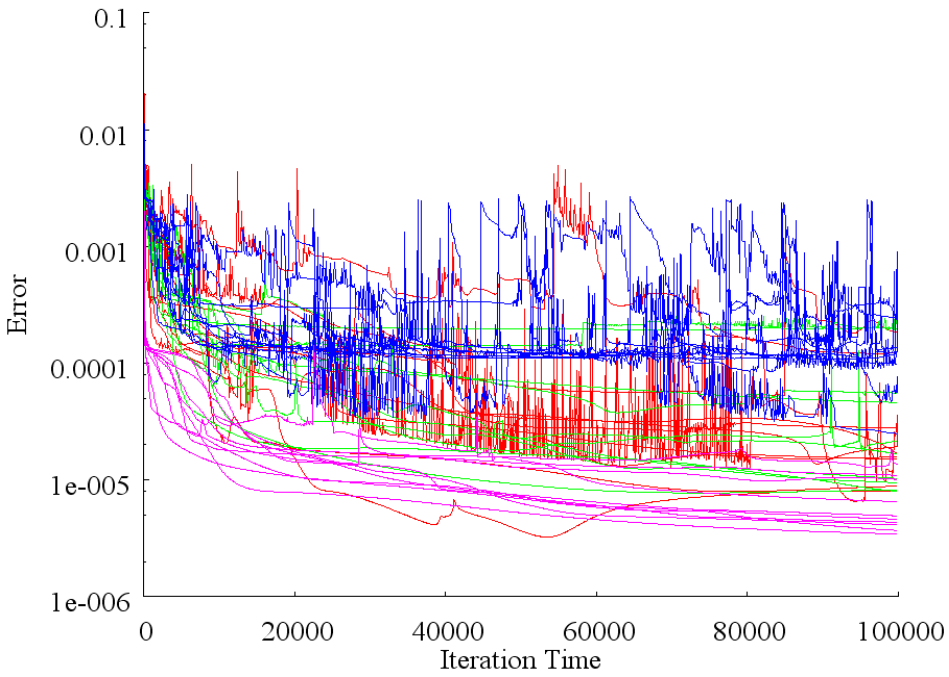

Figure 3-14. 電位固定パターン１～４を用いて Na^+コンダクタンス学習中の RNN の出力誤差の経時変化　赤線がパターン 1、緑線がパターン 2、青線がパターン 3、桃線がパターン 4 の誤差変化を示している。片対数グラフを用いた。

また、各電位固定パターンの最小誤差の平均値および標準誤差を Table 3-3 に示した。棒グラフで Figure 3-15 に示した。

Table 3-3　電位固定パターン１～４による最小誤差の平均値（Na^+コンダクタンス）

パターン名	最小誤差の平均値
パターン 1	2.68E-05±1.06E-05
パターン 2	4.85E-05±1.93E-05
パターン 3	8.39E-05±1.35E-05
パターン 4	6.53E-06±1.03E-06

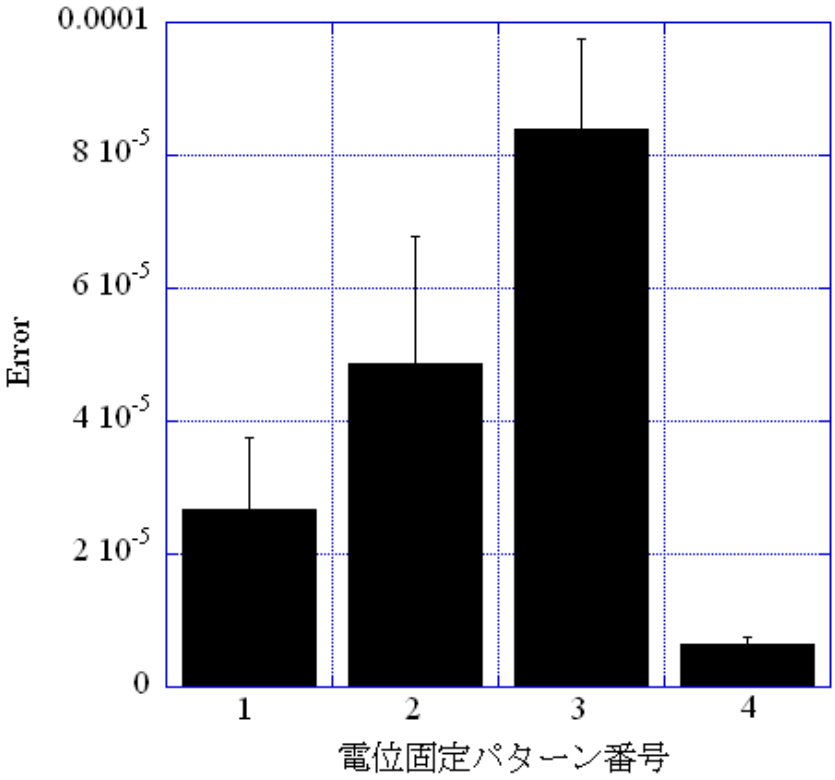

Figure 3-15　電位固定パターン 1～4 による最小誤差の平均値の棒グラフ（Na^+コンダクタンス）

電位固定パターン 4 が、他よりも誤差が小さい傾向があったが、有意差は観察出来なかった。

次に、各電位固定パターンを用いて、K^+コンダクタンス学習中の RNN の出力信号と教師信号との誤差を 4 種の電位固定パターン各 10 例分全て Figure 3-16 に示した。

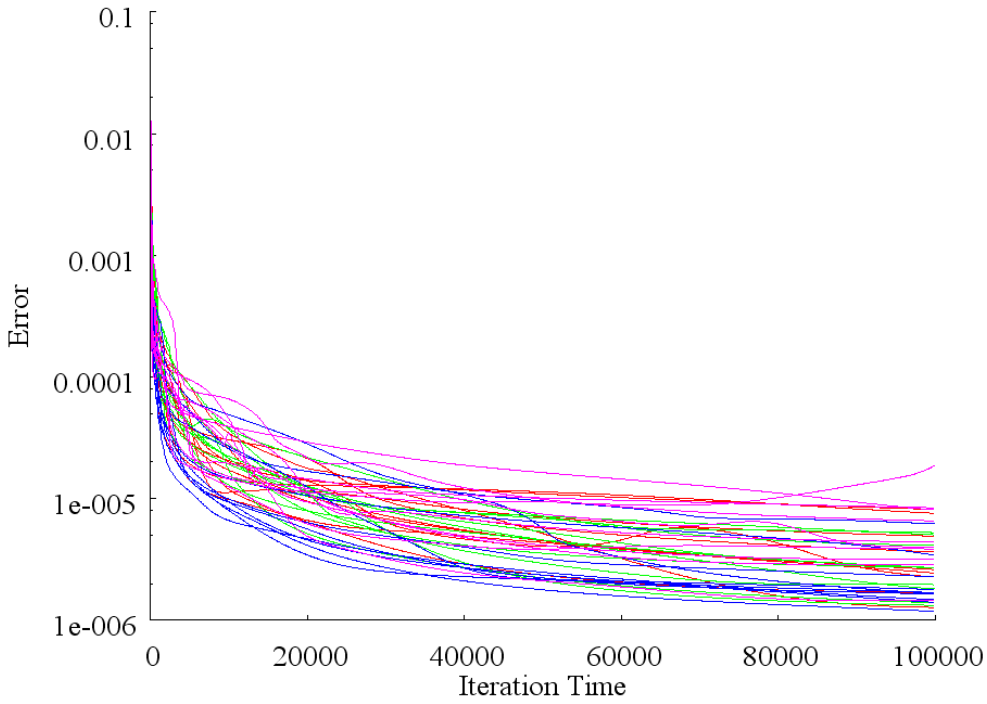

Figure 3-16. 電位固定パターン 1〜4 を用いて K^+コンダクタンス学習中の誤差変化　グラフの意味は **Figure 3-14** と同様である。

また、最小誤差の平均値および標準誤差を Table 3-4 に示し、また棒グラフで Figure 3-17 に示した。　4 種の電位固定パターンで有意な差は観察出来なかった。

Table 3-4　電位固定パターン 1〜4 による最小誤差の平均値（K^+コンダクタンス）

パターン名	最小誤差の平均値
パターン 1	3.72E-06±7.21E-07
パターン 2	2.90E-06±4.30E-07
パターン 3	2.27E-06±4.54E-07
パターン 4	5.15E-06±7.81E-07

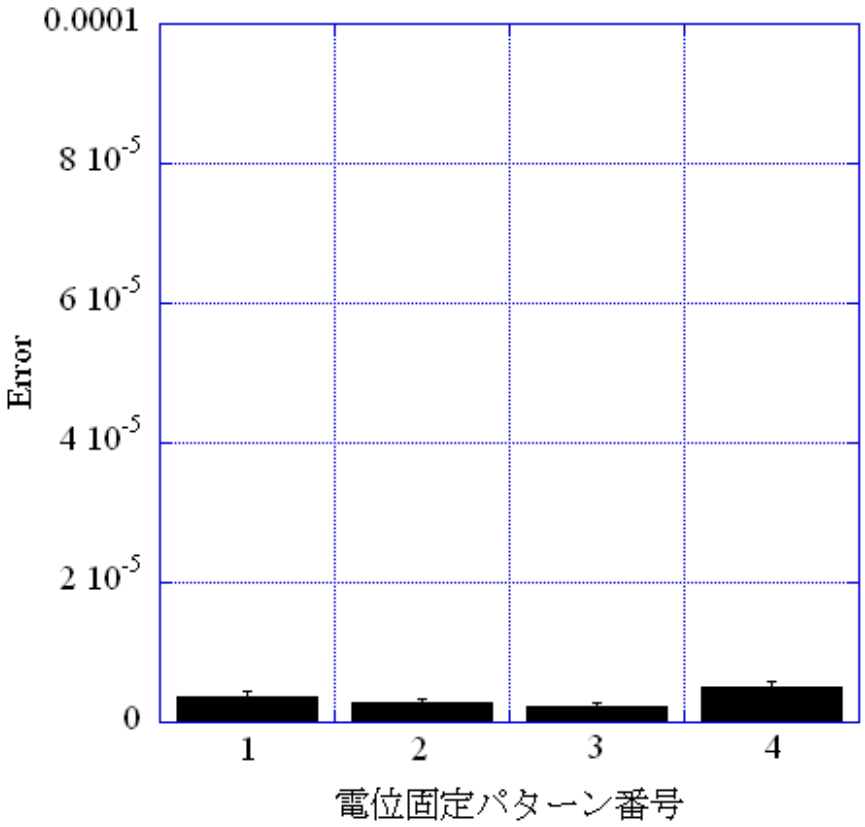

Figure 3-17　電位固定パターン 1～4 による最小誤差の平均値の棒グラフ（K^+コンダクタンス）
縦軸のスケールは Na^+の場合と同じにした。

3. 学習時に用いていないテストパターンを RNN に与えた時の誤差値の比較

3 種の RNN 間で比較した時と同様に、学習に用いていない電位テストパターン入力した時に正しいコンダクタンスの時間変化を出力出来るかどうかテストを行った。Na^+-FRNN の結果を Figure 3-18 に示す。テストデータとしては Figure 2-13~Figure 2-20 を用いた。

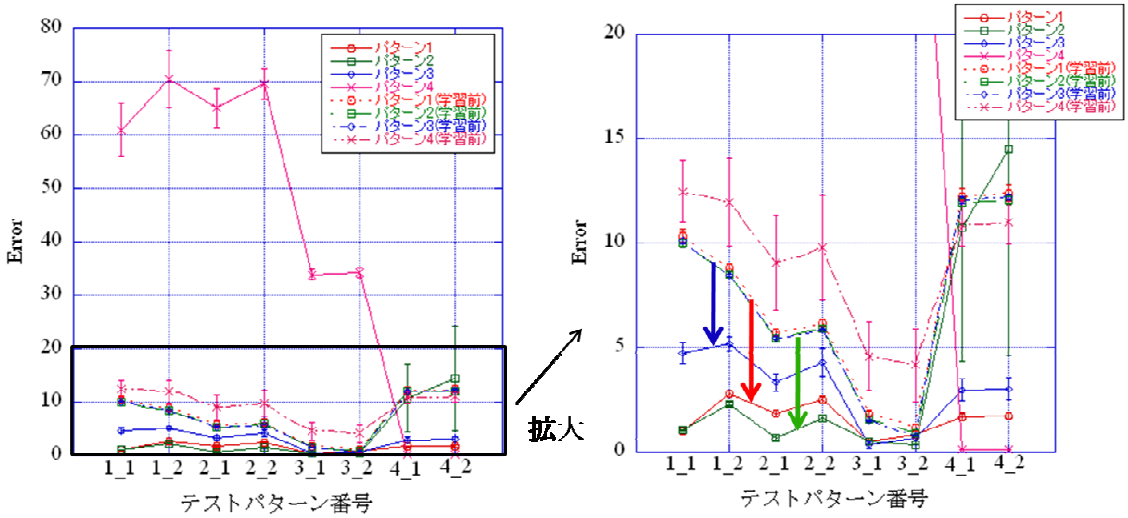

Figure 3-18　電位固定パターン 1～4 で学習させた Na^+-FRNN のテスト結果　右図は左図の拡大図である。赤がパターン 1、緑がパターン 2、青がパターン 3、黒がパターン 4 の結果である。学習前と学習後の誤差を、色を対応させて表示した。点線が学習前で実戦が学習後のデータである。矢印は学習前と後との誤差変化を示している。

電位固定パターン 2 で学習した FRNN がテストパターン 3, 4 において有意に誤差が小さかった(*p <0.05, ***p<0.005; Steel Dwass 法)。またパターン 4 で学習したものも 2 つのテストパターン 7, 8 において有意に誤差が小さかった(****p<0.005; Steel Dwass 法)。しかしパターン 4 で学習したものは誤差が小さかったもの以外は他の 3 つに比べて誤差が非常に大きいという結果となった。学習前と後で比べるとパターン 1,2,3 については全体的に誤差が落ちており、ダイナミクスを再現できていると考えられる。

次に K^+コンダクタンスでの結果を示す。

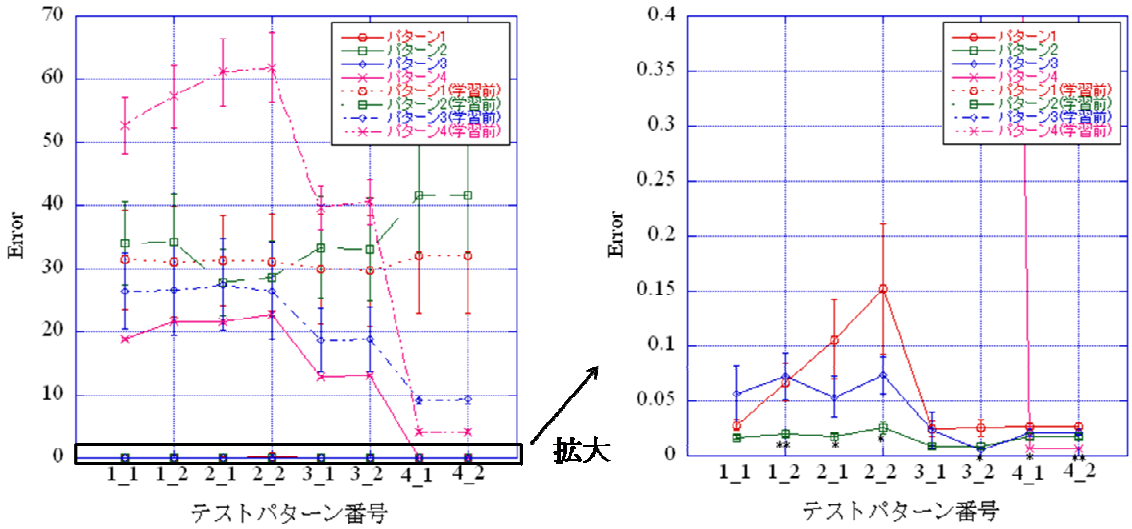

Figure 3-19　電位固定パターン 1～4 で学習させた K^+-FRNN のテスト結果　右図は左図の拡大図である。図の意味は **Figure 3-18** とものと同じである。

電位固定パターン 2 で学習した場合がテストパターン 2, 3, 4 で有意に誤差が小さかった(*p<0.05, **p<0.01; Steel Dwass 法)。また、パターン 4 で学習した場合が 2 つのテストパターン 7, 8 で有意に誤差が小さかった(*p<0.05, **p<0.01; Steel Dwass 法)。しかし、Na^+のときと同じように、この 2 つ以外は誤差がとても大きい結果となった。また、4 つのパターンすべての場合で学習前よりも学習後のほうがすべてのテストパターンで誤差値が下がっており、ダイナミクスを再現できていると考えられる。

第2節　RNN 神経による神経活動

第1項　　RNN 神経の神経活動の例

RNN 神経の一つの例として、固定電位パターン 3 を用いて学習させた FRNN を用いて計算した FRRNN 神経の神経活動を示す。最初に、弱い刺激と強い刺激を与えた。弱い刺激（Figure 3-20 左）を与えても SG 神経も RNN 神経も活動電位は発生せず、強い刺激（Figure 3-20 右）を与えると活動電位が発生した。Na^+, K^+コンダクタンスの値も示したが、学習に用いられていない時間変化（時系列）を生成した（Figure 3-20）。

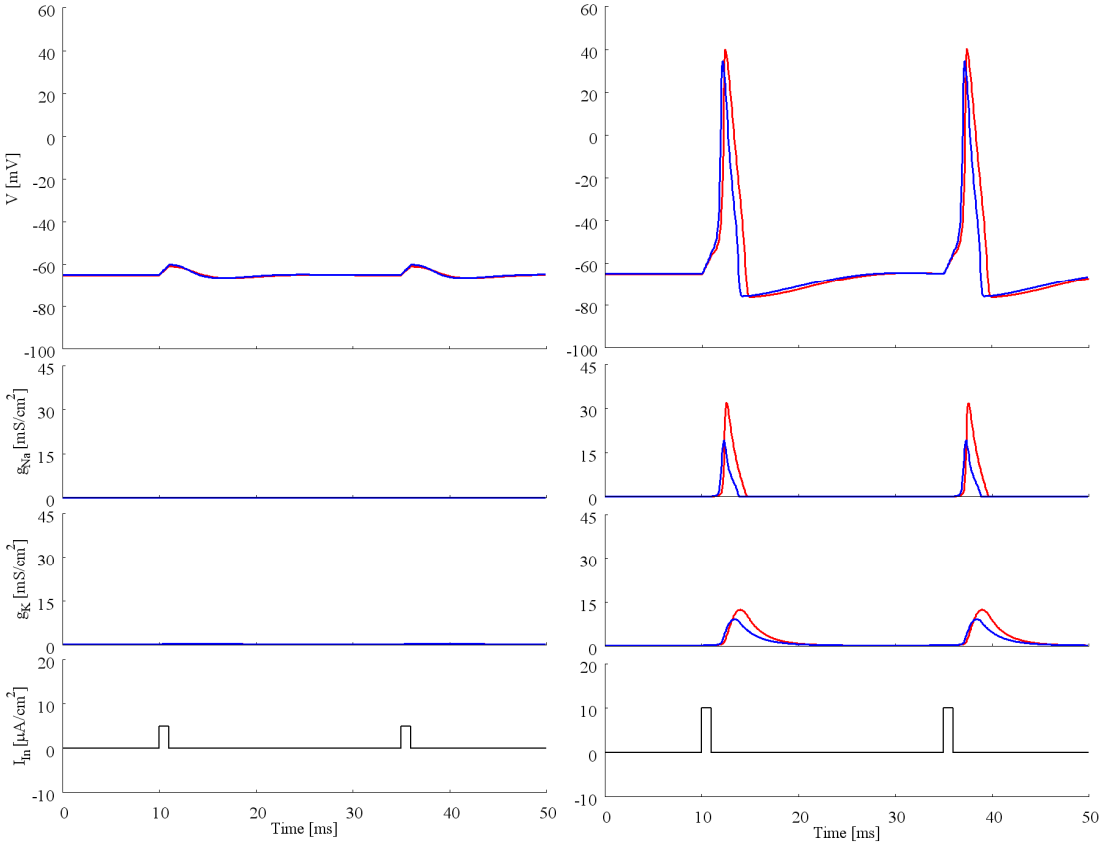

Figure 3-20. RNN 神経による閾値の確認と活動電位の生成　赤線が SG 神経、青線が RNN 神経の膜電位を表す。上から膜電位、Na^+コンダクタンス、K^+コンダクタンス、刺激電流強度が示されている。

また、パルス刺激の強度と活動電位の振幅についての関係を Figure 3-21 に示した。SG 神経、RNN 神経共に同様な傾向であった。

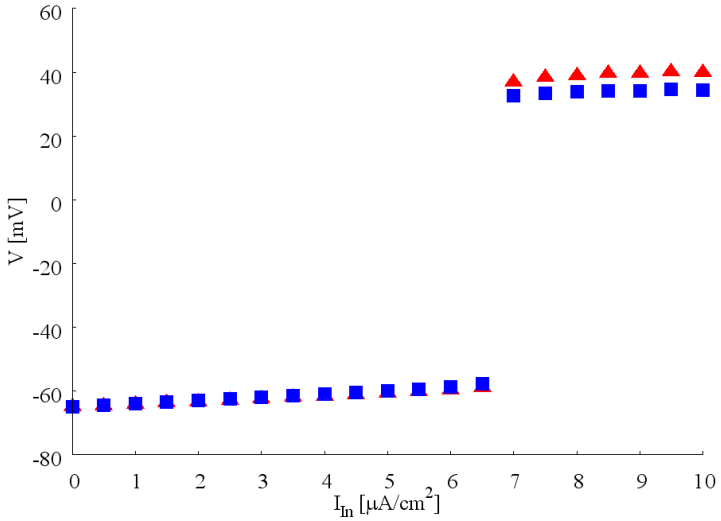

Figure 3-21. パルス刺激と膜電位の振幅との関係　赤三角が SG 神経、青四角が RNN 神経の結果

である。

次に、絶対不応期について調べるため短いインターバルの連続刺激を与えた。2 回目の刺激は 1 回目と同じ刺激強度及び 2 倍の刺激強度で与えた。RNN 神経は SG 神経と同様に、どちらでも 2 回目の活動電位を起こらなかった。この結果より、RNN 神経でも絶対不応期が存在する事が示唆された（Figure 3-22）。

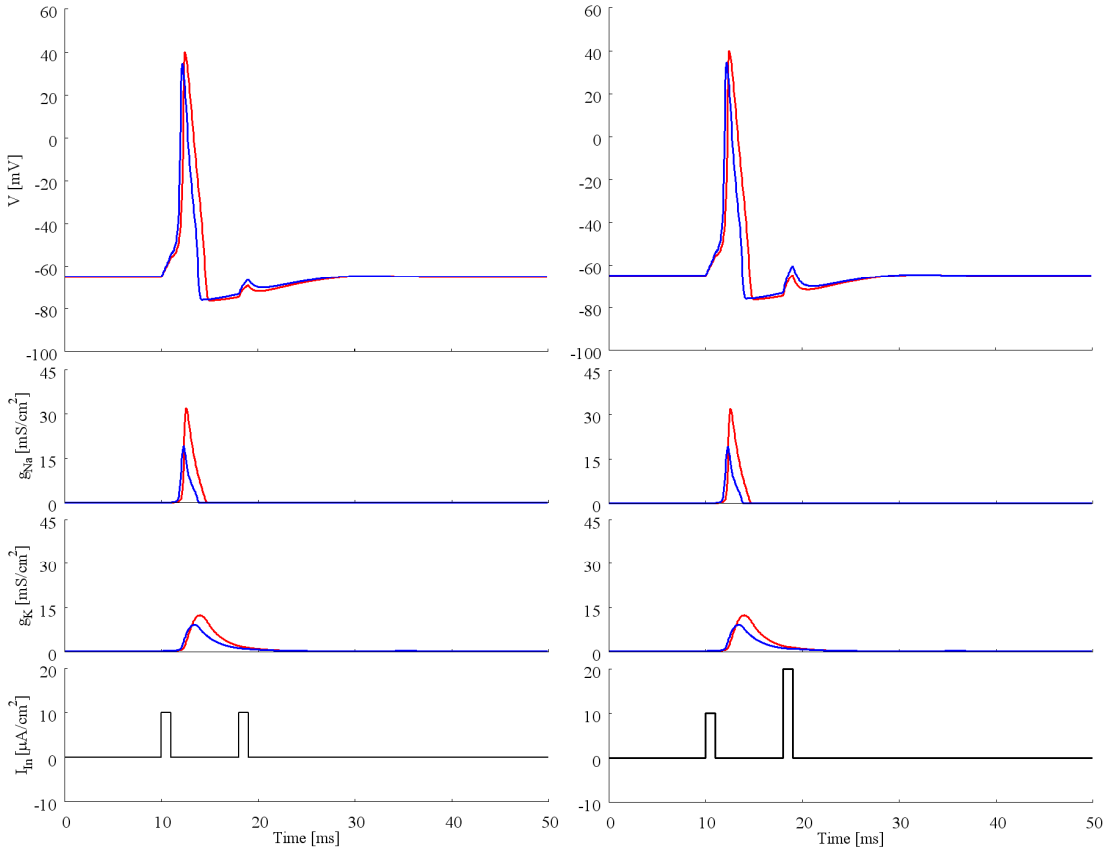

Figure 3-22. RNN 神経における絶対不応期　2 つの連続刺激を短いインターバル 8　ms 間隔で与えた。赤線が SG 神経、青線が RNN 神経の膜電位を表す。

また、Figure 3-22 の刺激のインターバルを長くして相対不応期についても調べた。RNN 神経において、1 回目と同じ強さの 2 回目の刺激では活動電位を発生しなかったが、その強さを 2 倍にしたとき活動電位が発生した。従って、FRNN 神経において相対不応期の存在が示唆される（Figure 3-23）。

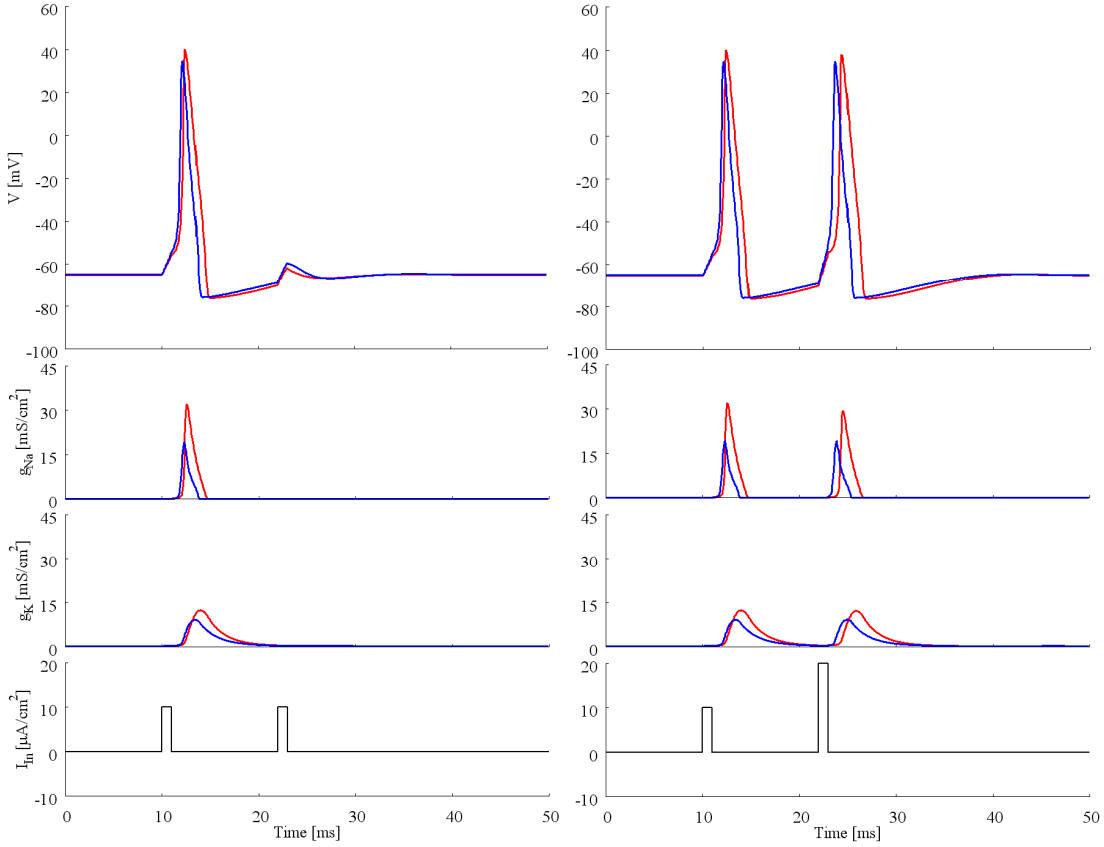

Figure 3-23. RNN 神経における相対不応期　2 つの連続刺激を Figure3-18 より長いインターバル 12 ms 間隔で与えた。赤線が SG 神経、青線が RNN 神経の膜電位を表す。活動電位のピークは SG 神経に比べて前にずれていた。

SG 神経は過分極刺激によりリバウンド発火による活動電位が誘導される[1]。それと同様に RNN 神経に過分極刺激を与えると、リバウンド発火による活動電位が誘導された(Figure 3-24)。

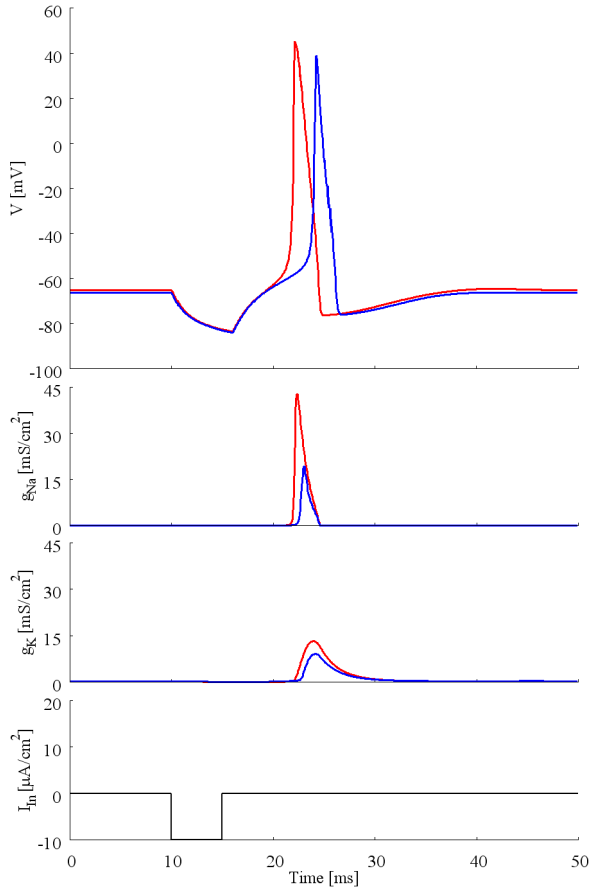

Figure 3-24. リバウンド発火の発生　RNN 神経に刺激強度-10μA/cm^2、持続時間 5 ms の過分極パルスを 10 ms の時点で与えた。赤線が SG 神経、青線が RNN 神経の膜電位を表す。但し、SG 神経に比べて、活動電位のピークは後ろにずれていた。

次に RNN 神経細胞に定常刺激を与えた結果を示す。刺激の強さが 10 **A/cm^2** の場合と 20 **A/cm^2** の場合の２つの結果を示す。RNN 神経も SG 神経と同様に周期的発火を生成した。強い刺激の方が高い周波数になるという結果になった（Figure 3-25）。但し、全ての周期発火の活動電位のピークは SG 神経に比べて時間的に前にずれていた。

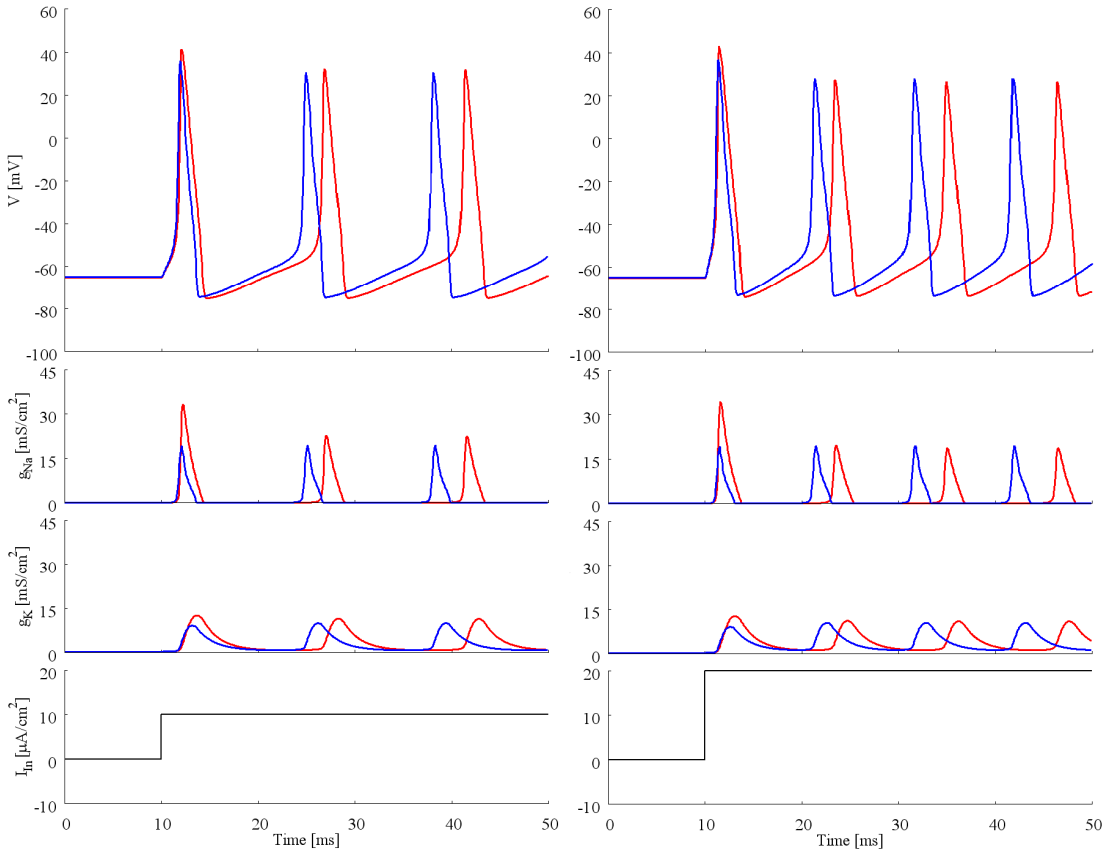

Figure 3-25. 周期的な活動電位の発生

また、定常刺激強度とその結果生成する周期的発火の周期との関係を Figure 3-26 に示した。RNN 神経は定性的に SG 神経と類似の周期を持っていた。

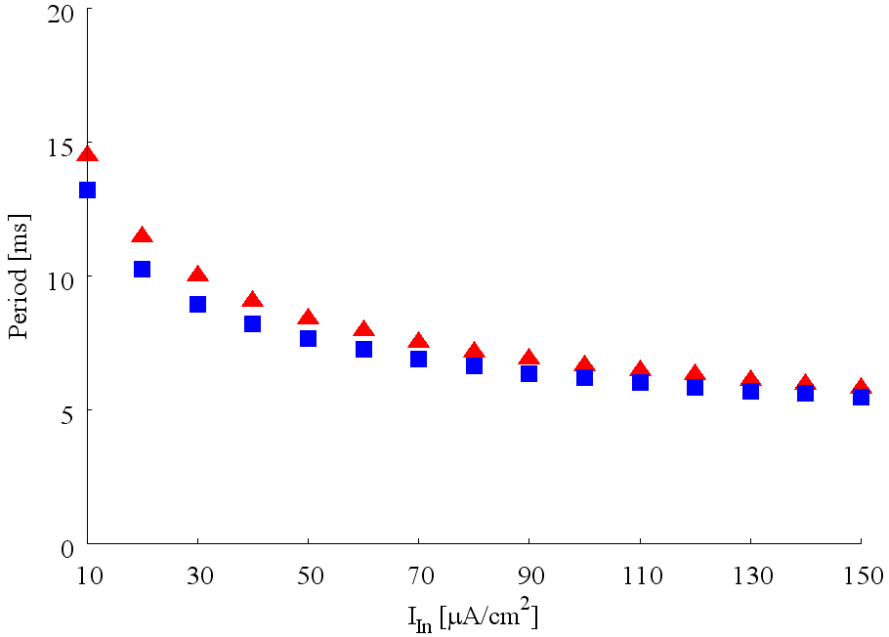

Figure 3-26. 定常刺激強度と周期的発火の関係 赤三角が SG 神経、青四角が RNN 神経を表す。

第2項　　3 つの RNN 間での神経活動の比較評価

TDRNN 神経、Elman 神経、FRRNN 神経の 3 種の RNN 神経について、様々な神経活動を再現しているかと言う視点で比較した。SG 神経と RNN 神経で神経活動の各現象について相互相関関数の最大値を計算し、RNN 神経 100 例の平均値を示した。神経活動の評価項目としては、強弱 2 種の刺激による実験、2 種の刺激強度を用いた絶対不応期の実験、2 種の刺激強度を用いた相対不応期の実験、リバウンド活動、2 種の刺激強度を用いた周期的発火の実験、の計 9 項目である。具体的には、Figure 3-20 で示したように強弱 2 種の刺激を与えた場合での閾値下応答と活動電位波形、Figure 3-22 で示したような 2 種の刺激パターンによる絶対不応期の測定実験波形、Figure 3-23 に示したような 2 種の刺激パターンによる相対不応期の測定実験波形、Figure 3-24 に見られる過分極パルスを与えた時のリバウンド神経発火の波形、Figure 3-25 に示したような周期的活動発火波形を用いて評価を行った。

Figure 3-27 に RNN の種類による違いについて調べた結果を示す。FRNN を用いた場合が Elman や TDNN を用いた場合に比べすべての神経活動について有意(****p<0.001; Steel Dwass 法)にその最大値が大きいという結果になった。つまり FRNN を用いた RNN 神経が一番 SG 神経に性質が類似している事が明らかになった。

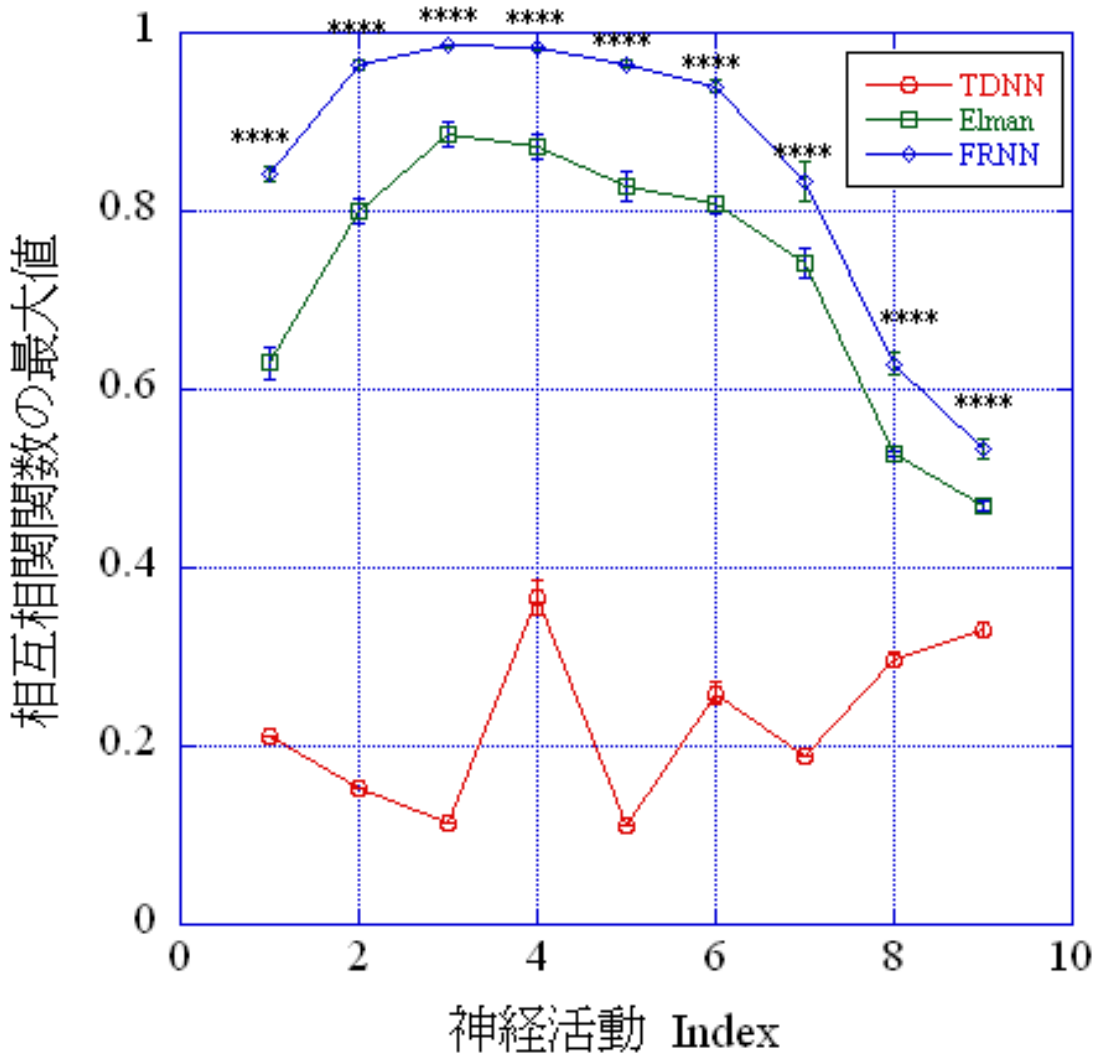

Figure 3-27 神経活動評価項目の RNN 別比較結果 神経活動 Index は、1 は閾値下応答、2 は活動電位、3,4 は 2 種の刺激強度による絶対不応期、5,6 は 2 種の刺激強度による相対不応期、7 はリバウンド活動、8,9 は 2 種の刺激強度による周期的発火波形による結果を示している。相互相関関数の最大値が 1 に近いほど、SG 神経の波形と類似している。

第3項　　4 種の電位固定パターンで学習した RNN 神経間での神経活動の比較評価

Figure 3-28 に、異なる電位固定パターンで学習した RNN 神経の神経活動の違いについて

評価した結果を示した。Figure 3-27 と同様に比較には相互相関関数の最大値を用いた。パターン 3 を用いた場合が、閾値下応答、リバウンド活動、弱い刺激での周期的活動の 3 項目について有意(*p<0.05, **p<0.01, ****p<0.001; Steel Dwass 法)に相互相関関数の最大値が大きかった。

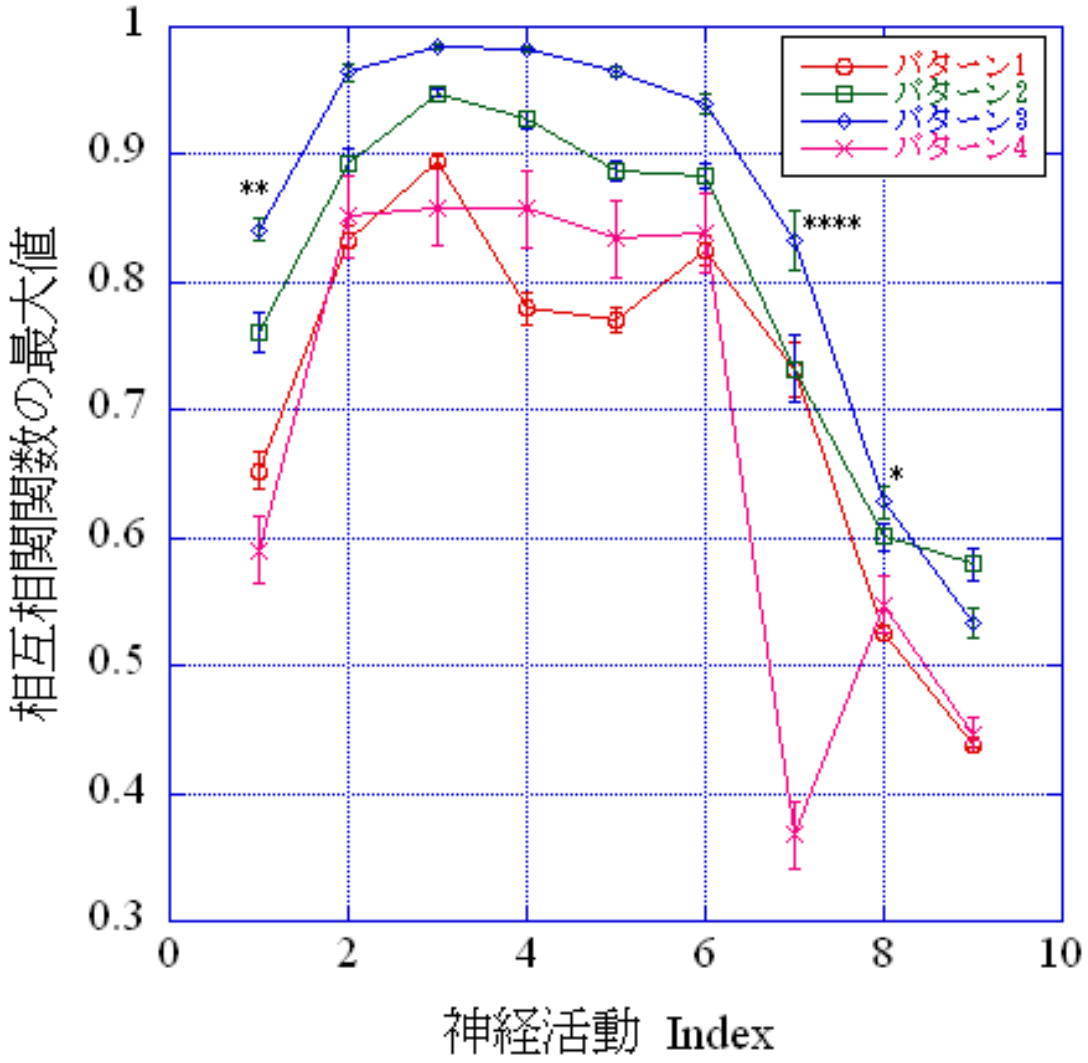

Figure 3-28 異なる電位固定パターンで学習した RNN 神経の神経活動比較結果 神経活動 Index は Figure 3-27 と同様である。

第3節　HH モデル以外のチャネルコンダクタンス時系列の再現

私の提案方法をヤリイカ巨大軸索のチャネル以外のチャネルにも適用出来るかどうか、ヤリイカの Na^+チャネル、K^+チャネル以外の別のチャネルに対して適用した。電位依存性 Ca^{2+}チャネルは海馬錐体細胞でのバースト発火を起こす役割を持ち、神経内部に Ca^{2+}を流入させ様々な Ca^{2+}依存的な分子ネットワークを活動させる。脳には様々な種類の Ca^{2+}チャネルがある。Kay と Wong により発見された海馬錐体細胞の Ca^{2+}チャネル[20]は電位依存性であった。今回の提案手法により、FRNN はその Ca^{2+}チャネルコンダクタンスの時間変化を学習し、膜電位に依存する時系列を再現することが出来た。Figure 3-29 に FRNN 学習時の誤差変化と膜電位固定パターンに対するコンダクタンス結果を示した。

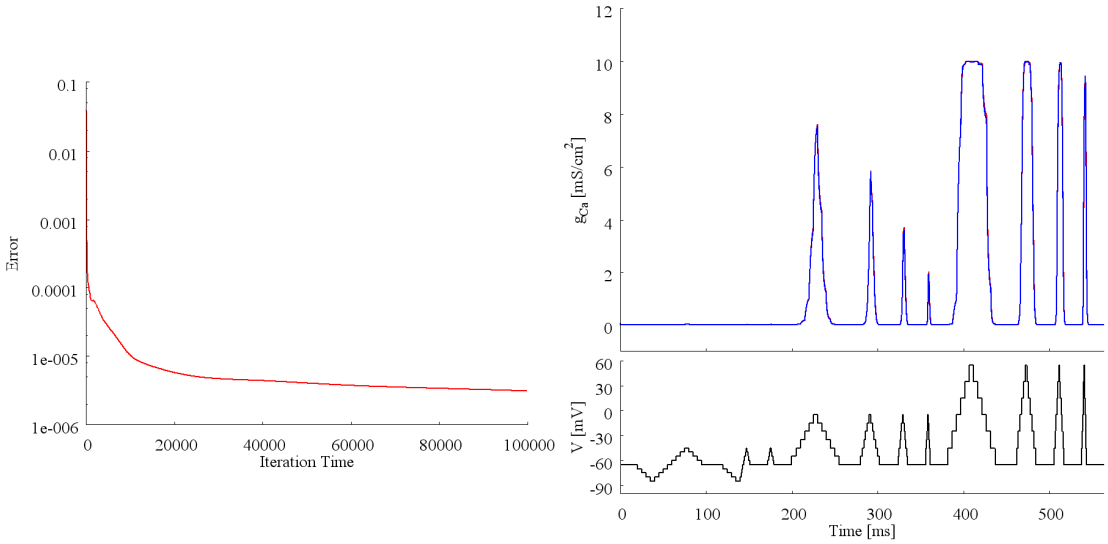

Figure 3-29. 海馬錐体細胞 Ca^{2+}チャネルのコンダクタンスダイナミックスの再現　左図に学習中の誤差変化、右図にコンダクタンスと固定電位パターンを示した。

第4章 考察と今後の課題

第1節　RNN によるコンダクタンス、及び神経活動の再現

ヤリイカ巨大軸索の Na^+、及び K^+コンダクタンスの時間変化を固定膜電位と共に RNN に学習させ、膜電位を与えた時のコンダクタンス時間変化を再現させた。そして再現した RNN を用いて閾値下応答、活動電位、絶対不応期、相対不応期、リバウンド神経発火、周期的活動など主な神経活動を再現することが出来た。私の提案手法である、実験で得られたチャネルコンダクタンスデータから RNN によりダイナミックスを推定し神経活動のシミュレーションに用いる事は可能であると考えられる。

それらの神経活動中で Na^+-RNN と K^+-RNN は、学習したチャネルコンダクタンス時系列とは異なる時系列を生成することが可能だった。このことからチャネルコンダクタンスを RNN に学習させる事により、RNN は学習したチャネルコンダクタンスの時系列だけでなく、チャネルコンダクタンスのダイナミックスも学習して神経活動の再現が可能であったと示唆される。

実際、RNN に学習していない電位固定パターン時系列を入力してもある程度正確なコンダクタンス時系列を再現出来る（Figure 3-8, Figure 3-9, Figure 3-18, Figure 3-19）ので、上記示唆は正しいと考えられる。

第2節　実験結果のまとめ

ここで、ion-RNN のテスト結果と RNN 神経と SG 神経の神経活動の類似度比較についての結果をまとめる。統計的に有意に評価が良かったものを Table の中に示した。Table 4-1 に RNN 別の結果を、Table 4-2 に固定膜電位パターン別の結果をまとめた。

RNN 別では、FRNN が多くの項目で有意に評価が良かった。

また、パターン別では、コンダクタンスダイナミックスのテスト結果では、パターン 2, 4 が良かったが、実際の神経活動の再現においては、パターン 3 を用いた場合が有意に評価が良かった。

Table 4-1 RNN 別結果のまとめ

	TDNN 型	Elman 型	FRNN 型
Na^+テストパターン 1			
Na^+テストパターン 2			
Na^+テストパターン 3			
Na^+テストパターン 4			
Na^+テストパターン 5			
Na^+テストパターン 6			
Na^+テストパターン 7			
Na^+テストパターン 8			
K^+テストパターン 1			***
K^+テストパターン 2			***
K^+テストパターン 3			***
K^+テストパターン 4			***
K^+テストパターン 5			**
K^+テストパターン 6			****
K^+テストパターン 7			***
K^+テストパターン 8			***
閾値下振動			****
活動電位			****
絶対不応期（弱）			****
絶対不応期（強）			****
相対不応期（弱）			****
相対不応期（強）			****
リバウンド活動			****
周期的活動（弱）			****
周期的活動（強）			****

Table 4-2 パターン別結果まとめ

	パターン 1	パターン 2	パターン 3	パターン 4
Na^+テストパターン 1				
Na^+テストパターン 2				
Na^+テストパターン 3		***		
Na^+テストパターン 4		*		
Na^+テストパターン 5				
Na^+テストパターン 6				
Na^+テストパターン 7				****
Na^+テストパターン 8				****
K^+テストパターン 1				
K^+テストパターン 2		**		
K^+テストパターン 3		*		
K^+テストパターン 4		*		
K^+テストパターン 5				
K^+テストパターン 6			*	
K^+テストパターン 7				*
K^+テストパターン 8				**
閾値下振動			**	
活動電位				
絶対不応期（弱）				
絶対不応期（強）				
相対不応期（弱）				
相対不応期（強）				
リバウンド活動			****	
周期的活動（弱）			*	
周期的活動（強）				

第3節　神経活動の再現に有効な RNN、及び RNN を学習させる電位固定パターン

RNN 神経の再現された神経活動と SG 神経の神経活動を比較すると、FRNN を用いた場合が Elman や TDNN を用いた場合に比べ有意(****p<0.001)に相互相関関数の最大値が大きい、すなわち SG 神経の神経活動と近かった（Figure 3-27）。そのため、神経活動の再現のためには RNN のうちで FRNN を用いれば良い事が示唆された。

膜電位固定パターンのうち、パターン 3 を用いた場合が、神経活動の 9 つの評価項目のうち 3 つ(閾値下応答、リバウンド活動、弱い刺激による周期的活動)について有意に相互相関関数の最大値が大きかった。有意差はなかったものの 5 つ（活動電位、絶対不応期（弱刺激、強刺激）、相対不応期（弱刺激、強刺激）でパターン 3 のときがその最大値が高かった。従って、パターン 3 のような、様々な振幅や周波数をもつ滑らかな電位固定パターンが RNN の学習のためには良く、神経活動をうまく再現できるということが示唆された。神経活動中の Na^+や K^+コンダクタンスはステップ的ではなく滑らかに変化する。よって滑らかなコンダクタンス時間変化を含むパターン 3 が有効だったのではないかと考えられる。

従って私の提案手法により、膜電位固定の実験で実際の神経細胞からチャネルコンダクタンスを取得する際、今回用いたパターン 3 のように電位を固定して得られたコンダクタンスデータを使って学習させた RNN を用いて神経細胞のシミュレーションを行えば、神経活動の再現には有効であると考えられる。よって、私のチャネルダイナミックス自動推定手法を含む、膜電位固定実験から神経細胞計算機シミュレーションまでの流れは Figure 4-1 のようになる。

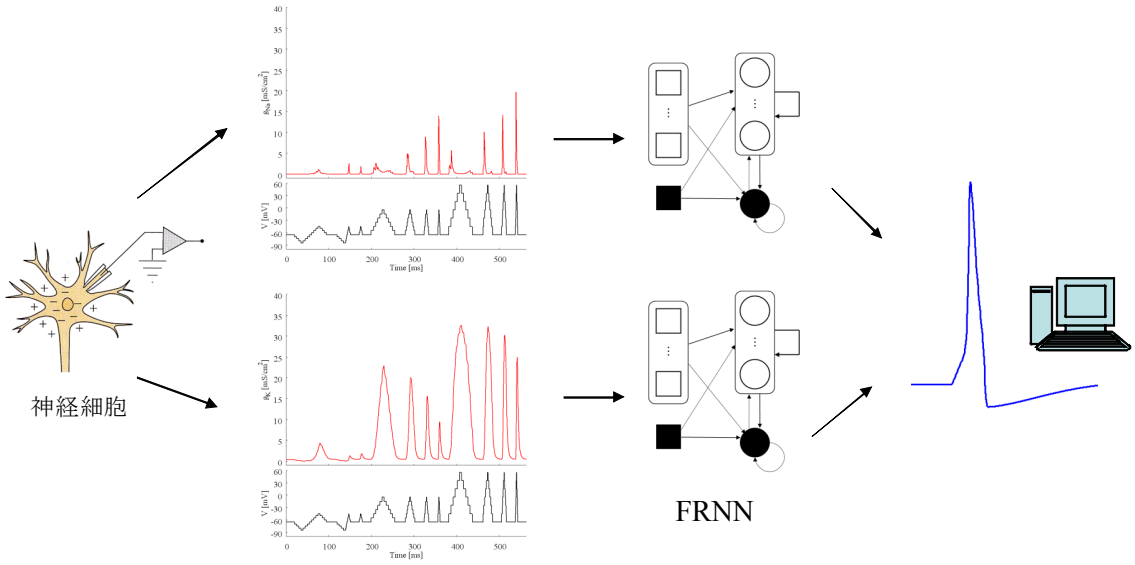

Figure 4-1　有効と示唆された実験からシミュレーションまでの流れ

第4節　Na^+チャネルの不活性化ゲートの再現

RNN 神経により得られた結果はいくらか SG 神経により得られたものと異なった（Figure 3-20～Figure 3-26）。それには 2 つの原因があると考えている。1 つ目は、神経活動を十分にシミュレートできるほど RNN がチャネルコンダクタンスの時系列を外挿することが出来ていない可能性がある。コンダクタンス時系列の再現能力は電位固定パターンに依存する。従って RNN がチャネルコンダクタンスをより正確に学習できる電位固定パターンを探す必要があると考えている。2 つ目は、Na^+-RNN は K^+-RNN ほど完璧に電位固定パターンを再現できていないことが考えられる。Na^+チャネルコンダクタンスは HH モデルによれば、活性化ゲートおよび不活性化ゲートと言う 2 つのゲート変数により計算される。HH モデルによれば、膜電位もコンダクタンスも定常であっても、2 つのゲート変数が変化している場合がある。m^3h の値が一定であっても m，h の値は様々存在する事に対応する。このことによる影響は Figure 2-9 の 100 ms から 200 ms　での間で見られる(Figure 4-2)。その間に 2 回膜電位を脱分極させる箇所がある。その時、対応して Na^+-コンダクタンス値の上昇があるが、1 回目の上昇は 2 回目に比べて大きい。この結果は RNN では学習出来なかった（Figure 3-3）。これは Na^+チャネルの不活性化ゲートの時間変化を RNN が学習出来なかったのではないか、と考えられる。今後は、そのような結果を学習できるように RNN の構造を修正したいと考えている。一つの可能性としては、入出力の履歴数を調節すれば改善されるかもしれない。

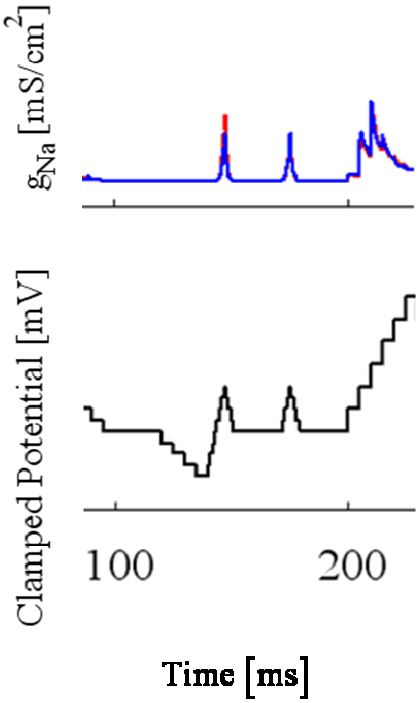

Figure 4-2 再現しきれていない Na^+コンダクタンスの変化 赤線が学習データで青線が RNN の出力である。不活性化ゲートの働きを十分に学習できていないと考えられる。

第5節　本提案手法の他チャネルへの適応の可能性

本提案手法は、ヤリイカ巨大軸索の Na^+チャネル、K^+チャネルだけでなく、海馬錐体細胞の Ca^{2+}チャネルコンダクタンスも再現する事が出来た。従って電気生理実験により測定されたチャネルコンダクタンスのダイナミクスの自動推定のための一般的方法として本提案手法が使用出来る事が示唆される。また、HH モデルでは再現できない活動電位が最近報告されている[21], [22]。それは早い開始スロープと様々な開始時間をもつ活動電位である。HH モデルとは異なり、複数の Na^+チャネル間で相互作用を考えている。このようなノン HH モデルに対しても、本提案手法は、適切な学習データを用いて RNN を学習させる事が出来れば、HH モデルが適応出来ない活動電位も再現できる可能性がある。

第6節　今後の課題、及び展望

本研究では、隠れ素子の数は恣意的に任意に決めた。しかしながら、課題に適した数が考えられる。今後は隠れ素子数も自動的に決められるようにする必要がある。また、入力

と出力の履歴、すなわち、膜電位とコンダクタンスの履歴をコンダクタンスの計算に用いた。その履歴数もまた任意に決めたので、この数も自動的に決められる仕組みが必要だと考えている。

本提案手法では、時間tの状態を入力しRNNはtのコンダクタンスの値を出力する仕様となっている。このままでは、タイムステップ幅を自由に変更することが出来ない。このためには値ではなく傾きを出力するように仕様を変更すればステップ幅を学習後にも自由に設定できる可能性がある。

チャネルダイナミクスには膜電位に依存するだけでなく細胞内 Ca^{2+}に依存するものもある。今後は本提案手法を、そのようなチャネルダイナミクスにも対応できるようにする必要がある。また、チャネルコンダクタンスダイナミクス再現の仕組みがRNNを用いているためブラックボックスになっている。すなわち、HHモデルで見られるような、ゲート変数に対応する情報を得ることは難しい。今後はこれらの情報を取得できるようにRNNの構造を改良する必要があると考えている。

この研究によりチャネルコンダクタンスダイナミクスが実験データを人の手で定式化することなく容易にRNNを用いて再現できるようになり、定式化に不慣れな研究者でもそのチャネルを用いてシミュレーションできるようになる。そして、シミュレーション結果を自分の研究、例えば電気生理実験など、にフィードバックできると考えられる。また、そのRNNを誰もがアクセスできるインターネット上のデータベースに保存すれば、他の研究者もそのRNNチャネルを扱うことができるようになる。そこで、神経細胞シミュレータのHIPPOSTATIONシステムにRNNを用いてコンダクタンスを計算する機能を加えたいと考えている。そうすると、コンダクタンスを計算する際、データベースに保存されているモデル化されたチャネルダイナミクスを用いることに加えて、RNNをもちいてコンダクタンスを再現、計算することができるようになる。このときのシステムの概要図をFigure 4-3に示す。このシステムを用いると、自分で測定したチャネルコンダクタンスだけでなく他の研究者によるチャネルコンダクタンスも容易に扱えるようになる。従って、誰もが数多くの種類のチャネルコンダクタンスを定式化なしに簡単に扱うことができるようになり、より一層の脳・神経の研究の発展につながると考えている。

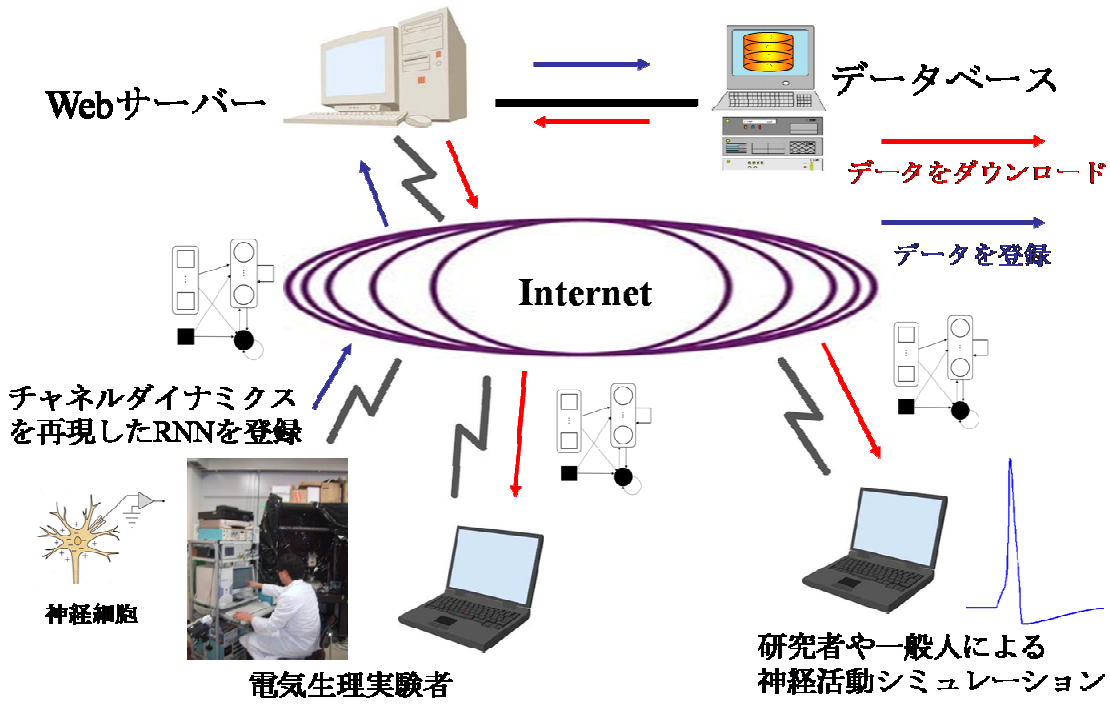

Figure 4-3　本提案手法を用いる場合の HIPPOSTATION システムの概要図　実験データを再現した RNN をデータベースに登録し、クライアントがそれをダウンロードして用いることができる。

数多くのチャネルがコンピュータ上で実現できるようになると、それらを組み合わせることで膨大な種類の神経細胞のシミュレートが可能になると考えられる。そしてシナプスを介してそれらを結合することで神経細胞ネットワークが構築できる。すなわち、海馬や小脳などの脳の部位の再現につながる。それらを適切につないでいくことで最終的には全脳のシミュレーションにつながっていくと考えている。そうすると、その仮想的な脳から脳波を記録したり、ロボットの制御に用いたりすることもできるようになるだろう。さらに脳だけでなく体のほかの部分、すなわち筋肉や各種臓器をシミュレーションして脳と協調させることができれば、1 つの生命体が実現につながる。このように全脳、そして 1 つの生命をコンピュータの中で再現することにこの研究はつながっていくと考えている。

本研究で用いた RNN は連続な外部入力をもちいて連続値を出力するシステムであり、チャネルダイナミクスの再現だけでなく様々な工学的な応用が可能だと考えている。まず、脳波による装置の操作にこの RNN を応用し、誰でも訓練の必要なくすぐに使える装置を開発できる可能性がある。例えば脳波によりマウスカーソルを右に動かすことを考えると、そのときの脳波は人によって異なる場合があり、誰もが訓練なしに使える装置を開発するのは難しいと思われる。この問題を解決するためにはそのような脳波の違いを吸収して同じ出力を行える仕組みが必要であり、そのために RNN を用いることが出来ないかと考えている。すなわち、RNN によって脳波パターンの時系列と出力を関連付けられるのではない

かと考えている。この例だと、右に動かすことに対応する脳波の数多くのパターンを学習させて、それらのパターンはもちろん、それ以外の右に動かすときのパターンも正しく識別できるようになるのではないかと考えている。そうすることで、誰にでも訓練なしに使えるインターフェイスの実現ができるのではないだろうか。また、インターネット上のトラフィックの負荷分散にも用いることができるとも考えている。RNN でネットワークのトラフィック量の時系列を学習することで、ビジー状態になるネットワークを予測して、そのネットワークを避けてルーティングが行えるような仕組みが作れるのではないかと考えている。そうできればトラフィック量の少ないネットワークへパケットが流れ、インターネット全体としてみて負荷が分散されていく。さらに、株価の予測にもこの RNN が用いられるのではないだろうか。株価の予測にはその株価自身の時系列と関連すると考えられる銘柄の時系列を入力とする予測システムが作れるのではないかと考えている。この場合はそれらの入力は連続値となっている。以上のようにこの RNN のシステムはチャネルコンダクタンスの計算のみでなく、他にも応用ができると考えられる。

第5章 結語

神経活動をシミュレートするために、膜電位固定実験のチャネルデータに基づき微分方程式を用いて神経のチャネルコンダクタンスのダイナミクスを定式化することが必要であるが、その定式化を必要としない方法として、RNN を用いてチャネルコンダクタンスのダイナミクスを推定する自動推定法を提案、開発した。ヤリイカ巨大軸索の Na^+と K^+チャネルコンダクタンスデータ時系列を用いて 2 つの RNN を学習させた。その結果、学習に用いた電位固定パターン以外のテストデータに対してもコンダクタンス時系列を再現出来た。この事は学習した時系列そのものと言うより、チャネルダイナミックスを RNN が学習したと考えられる。そして、それらの RNN を用いて、そのヤリイカの巨大軸索の神経活動を再現できた。RNN 神経においても、閾値性、活動電位、不応期、リバウンド活動、周期的活動が SG 神経同様観察出来た。従って、今回の推定手法は実験データより新しく見つかったチャネルのコンダクタンスの時系列を自動的に推定することができ、またその結果を用いて容易に、新しいチャネルを用いた神経活動をシミュレートすることが出来ると予想される。また、相互相関関数の最大値を用いて、様々な神経活動について比較を行ったところ、FRNN でかつ滑らかな電位固定パターンで学習させれば、神経活動を良く再現出来たので、それらの組み合わせにより私の提案手法が様々なチャネルダイナミックスの自動推定に用いられる事が示唆される。

第6章 謝辞

本研究を行うにあたり，終始多大なる御指導ならびに御鞭撻を賜りました，夏目季代久教授に心より感謝申し上げます.

学位論文の審査を快くお引き受けくださり，貴重なご助言をいただきました九州工業大学　石川　真澄　教授，立野　勝己　准教授に厚く御礼申し上げます.

なお，本研究は，九州工業大学 21 世紀 COE プログラム「生物とロボットが織りなす脳情報工学の世界」の補助を受けて行われました．ここに感謝いたします.

最後になりましたが，本研究を行うにあたって，常に御援助頂きました夏目研究室の全ての皆様に感謝いたします.

第7章 参考文献

[1] A. Hodgkin and A. Huxley. 1952. A quantitative description of membrane current and its application to conduction and excitation in nerve. Journal of Physiology 117:500–544.

[2] E. M. Izhikevich. 2007. Dynamical Systems in Neuroscience.-Geometry of Excitability and Bursting. The MIT Press.

[3] K. Doya, A. I. Slverston, and P. F. Rowat. 1994. A Hodgkin-Huxley type neuron model that learns slow non-spike oscillation. Advances in Neural Information Processing Systems 6: 566–573.

[4] M. Jordan. 1986. A parallel distributed processing approach. ICS Report 8604.

[5] Jun Namikawa and Jun Tani, 2009, Building Recurrent Neural Networks to Implement Multiple Attractor Dynamics Using the Gradient Descent Method, Volume 2009, Article ID 846040, 11 pages.

[6] S. Usui 2003. Visiome: neuroinformatics research in vision project. Neural Networks. 16: 1293-1300.

[7] M. L. Hines and N. T. Carnevale.1997. The NEURON simulation environment. Neural Computation 9: 1179–1209.

[8] J. M. Bower and D. Beeman. 1998. The Book of GENESIS, second edition, Springer.

[9] 吉松 周平. 2004. ブレインシミュレータ HIPPO-STATION の開発. 修士論文.

[10] M. Takahashi and K. Natsume. 2005. The Development of Brain Simulator HIPPOSTATION. IEICE Technical Report NLP2005-80: 53-57.

[11] 高橋 正明. 2006. 汎用的ブレインシミュレータ HIPPOSTATION システムの開発. 修士論文.

[12] W.S. McCulloch and W. Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 7: 115-133.

[13] K. Funahashi. 1989. On the approximate realization of continuous mappings by neural networks. Neural Networks. 2: 183-192.

[14] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and D. Jackel. 1990. Handwritten digit recognition with a back-propagation network. Advances in Neural Information Processing systems. 2. MIT Press.

[15] T. J. Sejnowski and C. R. Rosenberg. 1989. Parallel Networks that Learn to Pronounce English Text. Complex Systems 1: 183-192.

[16] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. 1989. Phoneme Recognition Using Time-Delay Neural Networks. IEEE Transactions on Acoustics, Speech, and Signal Processing 37. 3: 328–339.

[17] J. L. Elman. Finding structure in time. 1990. Cognitive Science 14: 179–211.

[18] R. J. Williams and D. Zipser. 1989. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. Neural Computation 1: 270-280.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. Parallel Distributed Processing. 1. 8. MIT Press.

[20] A. R. Kay and R. K. Wong. 1987. Calcium current activation kinetics in isolated pyramidal neurons of the Ca1 region of the mature guinea-pig hippocampus. Journal of Physiology 392: 603–616.

[21] B. Naundorf, F. Wolf, and M. Volgushev. 2006. Unique features of action potential initiation in cortical neurons. Nature 440: 1060–1063.

[22] G. Baranauskas and M. Martina. 2006. Sodium Currents Activate without a Hodgkin and Huxley-Type Delay in Central Mammalian Neurons. The Journal of Neuroscience 26: 671-684.

第8章 付録

第1節　Hodgkin-Huxley モデル

Hodgkin-Huxley モデルの膜電位とそれぞれのゲート変数の式を以下に示す。

$$C\frac{dV}{dt} = -I_{\mathrm{Na}} - I_{\mathrm{K}} - I_{\mathrm{L}} + I_{\mathrm{in}}$$

$$I_{\mathrm{Na}} = g_{\mathrm{Na}}\left(V - V_{\mathrm{Na}}\right)$$

$$I_{\mathrm{K}} = g_{\mathrm{K}}\left(V - V_{\mathrm{K}}\right)$$

$$I_{\mathrm{L}} = \bar{g}_{\mathrm{L}}\left(V - V_{\mathrm{L}}\right)$$

$$g_{\mathrm{Na}} = \bar{g}_{\mathrm{Na}} m^3 h$$

$$g_{\mathrm{K}} = \bar{g}_{\mathrm{K}} n^4$$

$$\frac{dm}{dt} = \alpha_m\left(1-m\right) - \beta_m m$$

$$\frac{dh}{dt} = \alpha_h\left(1-h\right) - \beta_h h$$

$$\frac{dn}{dt} = \alpha_n\left(1-n\right) - \beta_n n$$

$$\alpha_m = \frac{0.1\left(V+40\right)}{1-\exp\left(-\left(V+40\right)/10\right)}$$

$$\beta_m = 4\exp\left(-\frac{V+65}{18}\right)$$

$$\alpha_h = 0.07\exp\left(-\frac{V+65}{20}\right)$$

$$\beta_h = \frac{1}{1+\exp\left(-\left(V+35\right)/10\right)}$$

$$\alpha_n = \frac{0.01\left(V+55\right)}{1-\exp\left(-\left(V+55\right)/10\right)}$$

$$\beta_n = 0.125\exp\left(-\frac{V+65}{80}\right)$$

V は膜電位を表す。$C = 1.0\mu\text{F/cm}^2$ は膜容量で、I_{Na}、I_{K}、I_{L} はそれぞれ Na^+、K^+、リーク電流を表す。I_{In} は刺激電流を表す。$V_{\text{Na}} = 50\text{mV}$ は Na^+平衡電位で、$V_{\text{K}} = -77\text{mV}$ は K^+平衡電位、$V_{\text{L}} = -54.4\text{mV}$ はリーク平衡電位である。g_{Na} は Na^+コンダクタンスであり、活性化ゲート変数 m 及び不活性化ゲート変数 h によりその値が決まる。$\overline{g}_{\text{Na}} = 120\text{mS/cm}^2$ はその最大コンダクタンス値である。g_{K} は K^+コンダクタンスであり、活性化ゲート変数 n によりその値が決まる。$\overline{g}_{\text{K}} = 36\text{mS/cm}^2$ はその最大コンダクタンス値である。また、$\overline{g}_{\text{L}} = 0.3\ \text{mS/cm}^2$ は最大リークコンダクタンスである。ゲート変数膜電位に影響を受け、時間とともに変化する。活性化ゲート変数は膜電位が上昇するにつれて値が高くなる性質があり、逆に不活性化ゲート変数は膜電位が上昇するにつれて値が低くなる性質を持つ。α_m と β_m は m ゲートに使われる速度定数で、α_h と β_h は h ゲートに、α_n と β_n は n ゲートに使われる。

第2節　Izhikevich のチャネルコンダクタンスダイナミクスの定式化方法

ゲート変数のダイナミクスを固定膜電位に対するゲート変数の収束値と収束する時の時定数で表現する方法である。m の場合を例にあげると、適切に電位固定実験を行うことで、収束値 $m_\infty(V)$ と時定数 $\tau_m(V)$ を実験的に求め、$m_\infty(V)$ をボルツマン関数で、$\tau_m(V)$ をガウシアン関数で近似する。それら 2 つの式を以下に示す。

$$m_\infty(V) = \frac{1}{1+\exp\left((V_{1/2}-V)/k\right)}$$

$$\tau_m = C_{\text{base}} + C_{\text{amp}}\exp\left(\frac{-(V_{\max}-V)^2}{\sigma^2}\right)$$

$V_{1/2}$、k、C_{base}、C_{amp}、$V_{\max}$、σ はパラメータで、実験結果を再現できるようにフィッティングする。そしてゲート変数のダイナミクスを以上の 2 つの関数を用いて以下のように表現する。

$$\frac{dm}{dt} = \frac{m_\infty(V)-m}{\tau_m(V)}$$

他のゲートについても同じように適切に電位固定法を行い求める。

第3節　Java プログラム

本論文のプログラムには Java 言語を用いた。作成した各クラス、およびそのメソッドを説明する。

- RateConstant クラス

  α や β などの速度定数の計算式を表現する抽象クラスである。従って、実際の式はサブクラスで実装する必要がある。

  RateConstant クラスのメソッド

  1. abstract public double alpha(double v)

     膜電位を用いて速度定数 α を計算する。

  2. abstract public double beta(double v)

     膜電位を用いて速度定数 β を計算する。

- RC_HH_Na_m クラス

  HH モデルの m ゲートの α や β を計算するクラスである。RateConstant クラスのサブクラスであり、alpha 関数と beta 関数を実装する。

  RC_HH_Na_m クラスのメソッド

  1. public double alpha(double v)

     m ゲートの速度定数 α を計算する。

  2. public double beta(double v)

     m ゲートの速度定数 β を計算する。

- RC_HH_Na_h クラス

  HH モデルの h ゲートの α や β を計算するクラスである。RateConstant クラスのサブクラスであり、alpha 関数と beta 関数を実装する。

  RC_HH_Na_h クラスのメソッド

  1. public double alpha(double v)

     h ゲートの速度定数 α を計算する。

2. public double beta(double v)

h ゲートの速度定数 β を計算する。

- RC_HH_K_n クラス

HH モデルの n ゲートの α や β を計算するクラスである。RateConstant クラスのサブクラスであり、alpha 関数と beta 関数を実装する。

RateConstant クラスのメソッド

1. public double alpha(double v)

n ゲートの速度定数 α を計算する。

2. public double beta(double v)

n ゲートの速度定数 β を計算する。

- DifferentialCalculation クラス

ダイナミクスを計算する抽象クラスである。Euler 法か Runge-Kutta 法を用いる。

- Diff_type1 クラス

DifferentialCalculation クラスを継承したクラスで、RateConstant クラスを用いてゲートのダイナミクスを計算する。

Diff_type1 クラスのコンストラクタ

public Diff_type1(RateConstant rc, double timeStep)

速度定数と時間刻み幅を引数にしてインスタンス化される。

- Gate クラス

ゲートを表現した抽象クラスで膜電位を用いてゲートを計算する。

- GateNormal クラス

速度定数を用いてダイナミクス計算をおこなうゲートを表現したクラスである。Gate クラスのサブクラスとして定義されている。

GateNormal クラスのコンストラクタ

public GateNormal(String gateName, DifferentialCalculation dc, double exponent, double initial_gate)

ゲートの名前、ダイナミクス計算クラス、乗数、初期値を用いてインスタンス化さ

れる。

- Channel クラス

  チャネルを表現した抽象クラスでチャネル電流を計算する。

  Channel クラスのメソッド

  1. abstract public void calcNextCurrent(double t, double v)

     次の時間のコンダクタンス値及び、チャネル電流を計算する。

  2. public double getNowConductance()

     現在のコンダクタンス値を出力する。

- ChannelNormal クラス

  ゲートを用いるチャネルを表現したクラスで Gate クラスを用いてチャネル電流を計算する。

  ChannelNormal クラスのコンストラクタ

  public ChannelNormal(String channelName, List<Gate> gateList, double gmax,
  double ep)

  チャネルの名前、用いるゲートをまとめたリスト、最大コンダクタンス、平衡電位を引数にしてインスタンス化する。

  ChannelNormal クラスのメソッド

  public void calcNextCurrent(double t, double v)

  Gate クラスを用いてゲート変数を計算し、次の時間のコンダクタンス値及び、チャネル電流を計算する。

- ChannelRNN クラス

  RNN を用いたチャネルを表現したクラスで、RNN クラスを用いてチャネル電流を計算する。

  ChannelRNN クラスのコンストラクタ

  public ChannelRNN(String channelName, RNN channelRNN, double ep, double
  initial_V, double initial_g)

  チャネルの名前、コンダクタンスを学習した RNN、平衡電位、膜電位の初期値、コンダクタンスの初期値を引数としてインスタンス化する。

ChannelRNN クラスのメソッド

public void calcNextCurrent(double t, double v)

RNN を用いて次の時間のコンダクタンス値及び、チャネル電流を計算する。

- Compartment クラス

コンパートメントを表現するクラスで Channel クラスから取得されるチャネル電流値を用いて膜電位を計算する。

Compartment クラスのコンストラクタ

public Compartment(String compartmentName, List<Channel> channelList,
double Cm, double initial_V)

コンパートメントの名前、用いるチャネルのリスト、膜容量、膜電位の初期値を引数としてインスタンス化する。

Compartment クラスのメソッド

1. public void compartment.calcNextMP(double time, double stimulus)

刺激を用いて次の時間の膜電位を計算する。

2. public void compartment.updataMP()

現在の膜電位を次の時間の膜電位に更新する。

3. public double compartment.getNowMP()

現在の膜電位の値を取得する。

4. public double getConducatance(String channelName)

引数で指定されたコンダクタンスの値を返す。

- Stimulation クラス

刺激を表現する抽象クラスである。

Stimulation クラスのメソッド

abstract public double I_stim(double time)

時間に従い、刺激値を出力する。

- BlockStimulation クラス

矩形波刺激を表現するクラスである。コンストラクタを用いて刺激の開始時間、持続時間、強度を設定する。

BlockStimulation クラスのコンストラクタ

BlockStimulation(double[] intensities, double[] startTimes, double[] durations)

刺激の開始時間、持続時間、強度をそれぞれ配列で与える。これにより、複数の矩形波を使うことが出来る。

BlockStimulation クラスのメソッド

public double I_stim(double time)

時間に従い、刺激値を出力する。

- RNNProperty クラス

RNN のパラメータをまとめたクラスである。

RNN クラスのフィールド

1. public int _inputNum

   入力素子数。

2. public int _outputNum

   出力素子数。

3. public int _inputHisotoryNum

   入力についての履歴数。

4. public int _outputHistoryNum

   出力についての履歴数。

5. public int _samplingNum

   学習データの点の数。

6. public double _learningRate

   学習率。

7. public double _normMin

   入出力を正規化するときの最小値。

8. public double _normMax
入出力を正規化するときの最大値。

- RNN クラス

RNN を表現した抽象クラスで、学習データを学習し再現することが出来る。種類についてはサブクラスで決める。

RNN クラスのメソッド

1. public void learningWithSave(double[][] learningData, double errorCriterion, double learningNum, double normMin, double normMax, int samplingPeriod, String path)
学習データを学習する関数である。また、学習中で最小の誤差になったときの RNN を rnn ファイルとして引数 path で指定されたファイルに記録する。また、同時に学習中の誤差の変化もリストに記録される。

2. public List<List<Double>> getErrorRateList()
学習中の誤差変化のリストを取り出す。

3. public double[][] testWithStateRecorded(double[][] testData, double timeStep)
入力テストデータを用いて RNN の出力をテストする。そのときの隠れ素子値も同時に記録する。

- TDNN クラス

TDNN を表現したクラスで、RNN クラスのサブクラスである。

TDNN クラスのコンストラクタ

1. public TDNN(RNNProperty rnnp)
RNNProperty にまとめられた RNN のパラメータを用いて TDNN クラスを生成する。

2. public TDNN(String path)
path で指定される rnn ファイルを読み込んで、その中身を用いて TDNN を生成する。

- Elman クラス

Elman 型 RNN を表現したクラスで、RNN クラスのサブクラスである。

Elman クラスのコンストラクタ

1. public Elman(RNNProperty rnnp)
   RNNProperty にまとめられた RNN のパラメータを用いて Elman クラスを生成する。

2. public Elman(String path)
   path で指定される rnn ファイルを読み込んで、その中身を用いて Elman を生成する。

- FRNN クラス

FRNN を表現したクラスで、RNN クラスのサブクラスである。

FRNN クラスのコンストラクタ

1. public FRNN(RNNProperty rnnp)
   RNNProperty にまとめられた RNN のパラメータを用いて FRNN クラスを生成する。

2. public FRNN(String path)
   path で指定される rnn ファイルを読み込んで、その中身を用いて FRNN を生成する。

- Time クラス

時間を管理するクラスで、時間刻み幅を設定し時間を進めていくことが出来る。

Time クラスのメソッド

1. Public static Time getInstance()
   インスタンス化された Time オブジェクトを取得する。

2. Public void setTimeStep(double timeStep)
   時間刻み幅を設定する。

3. Public void setTime(double time)
   時間を引数 time で初期化する。

4. Public void forwardTime()
   時間刻み幅分時間を進める。

- MyMath クラス

  様々な計算をする関数をまとめたクラスで、関数はすべて static となっている。従って、このクラスをインスタンス化することはない。

  MyMath クラスのメソッド

  1. Public static double getAverageMSE(double[] data1, double[] data2)

     2 つのデータ列から平均 MSE（MSE をデータ列の長さで割った値）を計算する。

  2. Public static double[] getXCorrNormalized(double[] data1, double[] data2)

     2 つのデータ列より計算された相互相関係数を返す。0 から 1 にノーマライズされたものを返す。

  3. public static double getMax(double[] data)

     配列中の最大値を返す。

  4. public static double[][] transpose(double[][] data)

     2 次元配列を転置する。

- MyIO クラス

  ファイルの書き込み、読み込みを行う関数をまとめたクラスで、MyMath クラスと同様に関数はすべて static 関数である。

  MyIO クラス のメソッド

  1. public static void writeData(String, double[][])

     2 次元配列データを記録する。

  2. public static double[][] readDataArray(String path)

     引数 path で指定された記録された 2 次元配列データを読み込む。

ここから以上のクラスを用いた各 main 関数を示す。それらは学習データを生成するためのもの、RNN に学習データを学習させるもの、RNN の出力をテストするもの、2 つのデータ列から MSE を求めるもの、2 つのデータ列より相互相関関数の最大値を求めるもの、SG 神経の膜電位を計算するもの、RRNN 神経の膜電位を計算するものである。

- 学習データの生成

```
public static void main(String[] args) {
```

```
String channelName = "Na";
String learningPath = "……"; //  学習データを格納するディレクトリを指定

double timeStep = 0.05;
double restingPotential = -65;

Channel channel = createChannel(channelName, restingPotential, timeStep);

List<Double> potentialList = new ArrayList<Double>();

Time time = Time.getInstance();
time.setTimeStep(timeStep);
time.setTime(0.);

int patternID = 1;
// patternIDで指定される固定電位パターンをpotentialListに追加する。
addPattern(potentialList, patternID);

List<List<Double>> conList = new ArrayList<List<Double>>();
List<Double> l = new ArrayList<Double>();

l.add(time.getTime());
l.add(potentialList.get(0));

l.add(channel.getNowConductance());
l.add(channel.getNowConductance());

conList.add(l);

int samplingPeriod = 1; //  点数を間引く必要があるときにこの数を操作する

for (int t = 1; t < potentialList.size() - 1; t++) {
   time.forwardTime();

   if (t % samplingPeriod == 0) {
```

```
            l = new ArrayList<Double>();

            l.add(time.getTime());
        }

        channel.calcNextCurrent(time.getTime(), potentialList.get(t));

        if (t % samplingPeriod == 0) {

            l.add(potentialList.get(t + 1));

            l.add(channel.getNowConductance());

            conList.add(l);
        }
    }

    String channelDynamicsFileName = "……"; //  学習データの名前を指定

    MyIO.writeData(learningPath + channelDynamicsFileName, conList);
}

// channelNameで指定されたチャネルを生成
private static Channel createChannel(String channelName, double restingPotential, double
                                        timeStep) {
    double initial_V = restingPotential;
    Channel channel = null;

    if (channelName.equals("Na")) {
        // Naチャネルの作成

        // mゲートの作成
        RateConstant rc_m = new RC_HH_Na_m();
        DifferentialCalculation dc_m = new Diff_type1(rc_m, timeStep);
        double exponent_m = 4;
        double initial_m = rc_m.alpha(initial_V) / (rc_m.alpha(initial_V) +
```

```
                rc_m.beta(initial_V));
        Gate mGate = new GateNormal("HH_Na_m", dc_m, exponent_m, initial_m);

        // hゲートを作成
        RateConstant rc_h = new RC_HH_Na_h();
        DifferentialCalculation dc_h = new Diff_type1(rc_h, timeStep);
        double exponent_h = 1;
        double initial_h = rc_h.alpha(initial_V) / (rc_h.alpha(initial_V) +
                rc_h.beta(initial_V));
        Gate hGate = new GateNormal("HH_Na_h", dc_h, exponent_h, initial_h);

        //  ゲートを１つのListにまとめる。
        List<Gate> gateList_Na = new ArrayList<Gate>();
        gateList_Na.add(mGate);
        gateList_Na.add(hGate);

        double gmax_Na = 120;
        double ep_Na = 50;

        channel = new ChannelNormal("HH_Na", gateList_Na, gmax_Na, ep_Na);
    }
    else if (channelName.equals("K")) {
        // Kチャネルの作成

        // nゲートの作成
        RateConstant rc_n = new RC_HH_K_n();
        DifferentialCalculation dc_n = new Diff_type1(rc_n, timeStep);
        double exponent_n = 4;
        double initial_n = rc_n.alpha(initial_V) / (rc_n.alpha(initial_V) +
                rc_n.beta(initial_V));
        Gate nGate = new GateNormal("HH_K_n", dc_n, exponent_n, initial_n);

        //  ゲートを１つのListにまとめる。
        List<Gate> gateList_K = new ArrayList<Gate>();
        gateList_K.add(nGate);
```

```
        double gmax_K = 36;
        double ep_K = -77;

        channel = new ChannelNormal("HH_K", gateList_K, gmax_K, ep_K);
    }

    return channel;
}
```

- RNN による学習データの学習

```
public static void main(String[] args) {

    String rnnName = "FRNN";

    int samplingPeriod = 100;
    int patternID = 0;
    int rnnID = 0;
    int iterationCount = 100000;
    double criterion = 1.0E-7;

    String learningPath = "……"; //  学習データのあるディレクトリを指定
    String resultPath = "……"; //  結果を格納するディレクトリを指定

    String learningDataFileName = "……"; //  学習データの名前を指定

    double[][] learningData = MyIO.readDataArray(learningPath + learningDataFileName);

    rnnp._samplingNum = learningData.length;

    RNN rnn = null;
    RNNProperty rnnp = makeRNNp(); // RNNのパラメータをまとめたクラスを作成

    if (rnnName.equals("TDNN")) {
        rnn = new TDNN(rnnp);
    }
    else if (rnnName.equals("Elman")) {
```

```
        rnn = new Elman(rnnp);
    }
    else if (rnnName.equals("FRNN")) {
        rnn = new FRNN(rnnp);
    }

    String RNNFileName = "……"; //  記録するRNNの名前を指定

    rnn. learningWithSave(learningData, criterion, iterationCount, rnnp._normMin, rnnp._normMax,
samplingPeriod, resultPath + RNNFileName);

    String ErrFileName = "……"; //  記録するエラーファイルの名前を指定

    MyIO.writeData(resultPath + ErrFileName, rnn.getErrorRateList());
}
```

- RNN の出力のテスト

```
public static void main(String[] args) {
    String rnnName = "FRNN";

    double timeStep = 0.05;

    String testDataPath = "……"; //  テストデータのあるディレクトリを指定

    String rnnDataPath = "……"; //  テストするRNNファイルがあるディレクトリを指定

    String testDataFileName = "……"; //  テストに用いるデータのファイル名を指定

    double[][] testData = MyIO.readDataArray(testDataPath + testDataFileName);

    String testRNNFileName = null; //  読み出すテストするRNNの名前

    RNN testRNN = null;

    if (rnnName.equals("TDNN")) {
```

```
        testRNNFileName = "……"; // 読み出すテストするRNNの名前を指定
        testRNN = new TDNN(testRNNFileName);
    }
    else if (rnnName.equals("Elman")) {
        testRNNFileName = "……"; // 読み出すテストするRNNの名前を指定
        testRNN = new Elman(testRNNFileName);
    }
    else if (rnnName.equals("FRNN")) {
        testRNNFileName = "……"; // 読み出すテストするRNNの名前を指定
        testRNN = new FRNN(testRNNFileName);
    }

    String testResultFileName = "……"; // テスト結果を格納するファイルの名前を指定

    MyIO.writeData(rnnDataPath    + testResultFileName, testRNN
            .testWithStateRecorded(testData, timeStep));
}
```

- MSE の取得

```
public static void main(String[] args) {

    String trainingPath = "……"; // 学習データのあるディレクトリを指定
    String resultPath = "……"; // 結果を格納するディレクトリを指定

    String fileName1 = "……"; // データ1のファイル名を指定
    String fileName2 = "……"; // データ2のファイル名を指定

    double[][] data1 = MyMath.transpose(MyIO.readDataArray(trainingPath + fileName1));
    double[][] data2 = MyMath.transpose(MyIO.readDataArray(resultPath + fileName2));

    int column1 = 2;
    int column2 = 2;

    double MSE = 0.0;

    System.out.println(MyMath.getAverageMSE(data1[column1], data2[column2]));
```

```
}
```

- 相互相関関数の最大値の取得

```
public static void main(String[] args) {

    List<Double> stimList = new ArrayList<Double>();

    stimList.add(10.0);
    stimList.add(10.0);

    String resultPath1 = "……"; //  膜電位データ1のあるディレクトリを指定
    String resultPath2 = "……"; //  膜電位データ2のあるディレクトリを指定

    String fileName1 = "……"; //  膜電位データ1のファイル名を指定
    String fileName2 = "……"; //  膜電位データ2のファイル名を指定

    double[][] data1 = MyMath.transpose(MyIO.readDataArray(resultPath1 + fileName1));
    double[][] data2 = MyMath.transpose(MyIO.readDataArray(resultPath2 + fileName2));

    int column1 = 1;
    int column2 = 1;

    // 2つのデータより相互相関係数列を生成する。
    double[] xcorr = MyMath.getXCorrNormalized(data1[column1], data2[column2]);

    //  相互相関関数の最大値を求める。
    double xcorrMax = MyMath.getMax(xcorr);

    System.out.println(xcorrMax);
}
```

- SG 神経、または RRNN 神経の膜電位計算

```
public static void main(String[] args) throws IOException {
    boolean rnnFlag = false; // RNNを使うかどうかを指定
    double timeStep = 0.05; //  単位はms
```

```
// HHコンパートメントを作成
Compartment compartment = null;

if (rnnFlag) {
    compartment = createHHCompartmentWithRNN(timeStep);
}
else {
    compartment = createHHCompartment(timeStep);
}

// 矩形波刺激を作成
Stimulation s = createBlockStimulation();

Time time = null;
time = Time.getInstance();
time.setTimeStep(timeStep); // 単位ms

double simuTime = 50.0;
double stepNumber = simuTime / timeStep;

List<List<Double>> vList = new ArrayList<List<Double>>();

for (int i = 0; i < stepNumber; i++) {
    List<Double> list = new ArrayList<Double>();

    double t = time.getTime();
    double v;

    list.add(t);

    compartment.calcNextMP(t, s.I_stim(t));
    compartment.updateMP();
    v = compartment.getNowMP();

    list.add(v);
    list.add(s.I_stim(t));
```

```
            list.add(compartment.getConductance("HH_Na"));
            list.add(compartment.getConductance("HH_K"));

            vList.add(list);

            time.forwardTime();
        }

        MyIO.writeData(resultPath + MPFileName.getMepFileNameForPhenomenon(modelName,
usedMethod, phenomenonName), vList);
        System.out.println("Finish");
    }

    // 矩形波刺激を作成する。刺激強度、開始時間、持続時間はこの中で指定される。
    private static Stimulation createBlockStimulation() {
        double intensity1, startTime1, duration1, intensity2, startTime2, duration2;

        intensity1 = 10;
        startTime1 = 10;
        duration1 = 1;

        intensity2 = 10;
        startTime2 = 35;
        duration2 = 1;

        double[] intensities = { intensity1, intensity2 };
        double[] startTimes = { startTime1, startTime2 };
        double[] durations = { duration1, duration2 };

        return new BlockStimulation(intensities, startTimes, durations);
    }

    // 通常の方法（各ゲートをRK法で計算）を用いるHHコンパートメントを作成
    private static Compartment createHHCompartment(double timeStep) {
        double initial_V = -65;
```

```
// Naチャネルの作成

// mゲートの作成
RateConstant rc_m = new RC_HH_Na_m();
DifferentialCalculation dc_m = new Diff_type1(rc_m, timeStep);
double exponent_m = 4;
double initial_m = rc_m.alpha(initial_V) / (rc_m.alpha(initial_V) + rc_m.beta(initial_V));
Gate mGate = new GateNormal("HH_Na_m", dc_m, exponent_m, initial_m);

// hゲートを作成
RateConstant rc_h = new RC_HH_Na_h();
DifferentialCalculation dc_h = new Diff_type1(rc_h, timeStep);
double exponent_h = 1;
double initial_h = rc_h.alpha(initial_V) / (rc_h.alpha(initial_V) + rc_h.beta(initial_V));
Gate hGate = new GateNormal("HH_Na_h", dc_h, exponent_h, initial_h);

//  ゲートを１つのListにまとめる。
List<Gate> gateList_Na = new ArrayList<Gate>();
gateList_Na.add(mGate);
gateList_Na.add(hGate);

double gmax_Na = 120;
double ep_Na = 50;

Channel NaChannel = new ChannelNormal("HH_Na", gateList_Na, gmax_Na, ep_Na);

// Kチャネルの作成

// nゲートの作成
RateConstant rc_n = new RC_HH_K_n();
DifferentialCalculation dc_n = new Diff_type1(rc_n, timeStep);
double exponent_n = 4;
double initial_n = rc_n.alpha(initial_V) / (rc_n.alpha(initial_V) + rc_n.beta(initial_V));
Gate nGate = new GateNormal("HH_K_n", dc_n, exponent_n, initial_n);

//  ゲートを１つのListにまとめる。
```

```
    List<Gate> gateList_K = new ArrayList<Gate>();
    gateList_K.add(nGate);

    double gmax_K = 36;
    double ep_K = -77;

    Channel KChannel = new ChannelNormal("HH_K", gateList_K, gmax_K, ep_K);

    // Leakチャネルの作成
    double gmax_Leak = 0.3;
    double ep_Leak = -54.4;
    Channel LeakChannel = new ChannelNormal("HH_Leak", gmax_Leak, ep_Leak);

    List<Channel> channelList = new ArrayList<Channel>();
    channelList.add(NaChannel);
    channelList.add(KChannel);
    channelList.add(LeakChannel);

    // HHコンパートメントの作成
    double C = 1.0;

    Compartment HHCompartment = new Compartment("HH", channelList, C, initial_V);

    return HHCompartment;
}

// RNNで各コンダクタンスを計算するHHコンパートメントを作成
private static Compartment createHHCompartmentWithRNN(double timeStep) {
    double initial_V = -65;

    // Naチャネルの作成
    RNN rnn_Na = new FRNN("..."); // Naチャネルのrnnファイルのパスを指定

    double ep_Na = 50;
    double initial_Na_g = 0.01;
```

```
    Channel NaChannel = new ChannelRNN("HH_Na_RNN", rnn_Na, ep_Na, initial_V,
initial_Na_g);

    // Kチャネルの作成
    RNN rnn_K = new FRNN("..."); // Kチャネルのrnnファイルのパスを指定

    double ep_K = -77;
    double initial_K_g = 0.37;

    Channel KChannel = new ChannelRNN("HH_K", rnn_K, ep_K, initial_V, initial_K_g);

    // Leakチャネルの作成
    double gmax_Leak = 0.3;
    double ep_Leak = -54.4;
    Channel LeakChannel = new ChannelNormal("HH_Leak", gmax_Leak, ep_Leak);

    List<Channel> channelList = new ArrayList<Channel>();
    channelList.add(NaChannel);
    channelList.add(KChannel);
    channelList.add(LeakChannel);

    // HHコンパートメントの作成
    double C = 1.0;

    Compartment HHCompartment = new Compartment("HH", channelList, C, initial_V);

    return HHCompartment;
}
```