山口大学教育学部研究論叢　第 75 巻　245 頁～ 254 頁　2026 年 1 月

筋電位でのジェスチャー操作におけるデータ量と計算コストの定量評価

伊藤　正剛*・北本　卓也**

Quantitative Evaluation of Data Volume and Computational Cost in Gesture Manipulation Using Myoelectric Potentials

ITO Masataka*, KITAMOTO Takuya**

（Received September 26, 2025）

筆者らによるこれまでの研究[1]-[3]で、握り・反り・動作なし3動作の表面筋電位（以下、筋電位）によるテキストカーソル（以下、カーソル）のジェスチャー操作を実現した。短時間の事前学習でカーソル操作を高い精度で行うことに成功した。しかし、まれに起こる誤識別が問題であった。そこで、学習に用いるデータを増やすことを考える。これによりジェスチャーによるカーソル操作が安定し、精度が増すことが期待される。また、作成したゲームのスコアの推移により、データ追加による動作識別の精度向上を客観的に示す。さらに、理論的な計算量と学習時間にフィットする関数の種類の対応関係を明らかにする。

1．はじめに

データを解析し規則性を見つける試みは古くから行われてきた[4]。近年、機械学習手法、特に深層学習の目覚ましい進歩により、画像認識、音声認識に加え、大規模言語モデルによる自然言語への応答は社会に大きな影響を与えている。筋電位の解析も例に漏れず機械学習、中でも深層学習が使われている。ここで、「表面筋電位」とは運動により表在筋を中心とした骨格筋に発生する活動電位を皮膚表面から抽出・記録した信号である。

筋電位を用いた動作識別方法には、離散型と連続型がある。離散型では基準値を超すかどうかを検出する閾値型[5]や2次元空間あるいはそれ以上の空間に何らかの特徴量を用いて分離する方法[6]、機械学習を用いて分類する方法などがある[7]-[11]。機械学習を用いる離散型では教師ラベルを与えられた動作について手の動作識別を行う。例えば、西川らは、2つの電極2チャンネルの筋電位から10動作識別を試みている[12]。また、辻らは、4対の表面電極を使い6動作識別を報告している[13]。一方、連続型は現在の動作をなめらかに予測する。例えば、カスタマイズされた Inception-time model を用い関節加速度を回帰により推定している研究もある[14]。最近では、離散型、連続型ともに複数のセンサを用いて分類や回帰を行うことが多い。

さらに、機械学習手法の発展だけでなく、ビッグデータがより簡単に得られるようになっている。例えば、インターネットの普及による画像、自然言語、あるいは様々なセンサからの大規模なデータなどである。筋電位の分野では、高密度に配置されたセンサ（高密度表面筋電図；HD-sEMG）をつけて多くの筋電位を測定し動作識別を行う研究もある[15]。

また、ビッグデータに関連した理論的な背景も研究されている。Sun らは JFT-300M データセット（画像分類モデルの学習に使う Google のデータセット）を用いて学習データの増加により対数的に精度が上昇することを示した[16]。また、Kaplan らは、ニューラル言語モデルにおけるスケーリング則（べき乗則）を見出した。つまり、Transformer において、データ量、パラメータ量、計算資源を増やすと損失がべき乗に沿って減少することを示した[17]。これらの研究は筋電位データや解析においても重要な示唆を与えている。

以前の筆者らの研究では、深層学習手法を使ったオフラインでの動作識別の精度解析[1]や機械学習の自動化ライブラリである PyCaret を用いて動作識別[2]を行った。最新の論文[3]では、深層学習から出力される信号

＊ 放送大学教養学部　　＊＊ 山口大学教育学部　〒753-8513　山口市吉田1677-1　kitamoto@yamaguchi-u.ac.jp

をさらに学習分類に用いる2段階の推論が精度良く可能であることを報告した。これまでの研究は、データ数を変化させることなく、短時間の事前学習や機械学習手法そのものに焦点を当ててきた。しかし、深層学習はデータ追加による分類性能の改善が様々な分野において見込まれるため[11], [12], [14]-[18]、筋電位を用いた動作識別においても性能が向上する可能性が高い。実際に、筋電位のビッグデータを集め深層学習を用いて高い分類精度を出している研究も存在する[13]。

また、2025年8月時点でタッチタイピングの左右手首筋電位データセット emg2qwerty[19][20]、ヒトやロボットの筋電義手のコントロールを目的としたマルチモーダルデータセット Ninapro[21]、手のジェスチャーデータセット UC2018 DualMyo[22]などが利用可能である。

このようなデータセットはあるものの本研究に特化したデータセットは存在しない。データを追加したときの動作識別精度への影響も分かっていない。そこで今回、独自に筋電位データを収集する。次に、収集したデータ追加による動作識別精度の向上を、作成したゲームのスコア推移により客観的に明らかにする。（補足として、動作を分類し信号を出力することを動作識別、その信号に基づいてカーソル操作、ゲーム操作（後述）を行うことから、動作識別≒カーソル操作（方向）≒ゲーム操作（方向）である）また、詳しくは2.7節で述べるが、理論的な計算量と実際の学習時間にフィットする関数の種類の対応関係を調査する。ここで、関数の種類とは、一次関数、二次関数、指数関数などである。この調査により、学習にかかる時間の指標を作り予測に用いることができる。

2．方法

2.1　実験参加者と使用するデータ

本研究の実験では、健常な2名、筆者（著者A）と別の著者の1人（著者B）が参加した。なお、本研究は放送大学研究倫理委員会の倫理審査の承認を得たものである（通知番号2023-73）。実験参加者は、自由意思のもと同意を得た上で実験を行った。本研究では、追加でデータ収集を行っているが、全ての収集したデータは著者Aと著者Bのデータである。追加に使用したデータは著者Aのもののみである。また、データを追加してゲーム実験を行ったのは主に著者Bである。

2.2　ARV 計測機器、使用コンピュータ、使用言語・ライブラリなど

ARV（整流平滑化）計測機器として、スポーツセンシング社の DSP ワイヤレス筋電センサ湿式タイプ（SS-EMGW-HM）を使用した。ここで、ARV とは筋電位の絶対値を求め平滑化（このセンサでは移動平均）したものである。シンプルで頑健な構造を目指すためセンサの数は1つ、1ch としている。また、電極は最初の研究から引き続き左前腕橈側手根屈筋の部位に貼り付けた。
実験コンピュータスペックを表１に示す。

表１　実験コンピュータスペック

OS	Windows 11 Pro 64bit
CPU	Intel Core i7-8565U (1.80GHz-4.60GHz 4コア/8スレッド)
RAM	16GB LPDDR3 2133MHz

コンピュータは2019年製であるが現在（2025年4月）の動作識別プログラムにおいて処理速度は特に問題はない。データ追加なしの学習時間は3動作15行20列の ARV データ（データについての詳細は後述する）で、1分30秒程度である。使用言語は Python3.11である。また、深層学習用ライブラリを表２に示した。

表２　深層学習用ライブラリ

TensorFlow	2.16.1
Keras	3.3.3
PyTorch	2.5.1

2.3　教師ラベル設定とゲーム操作

ジェスチャー操作を行うため学習に用いる動作は3つである（3動作識別）。参考文献[3]から引用し動作と教師ラベルを図１に示した。教師ラベルは、握り0、反り1、動作なし2に設定した。握ってすぐ力を抜くなど動作は瞬間的である。カーソル操作・ゲーム操作は、動作識別の目に見えるフィードバックシステムとしても働く。

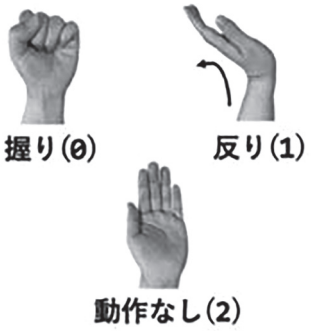

図１　動作（ジェスチャー）と教師ラベル

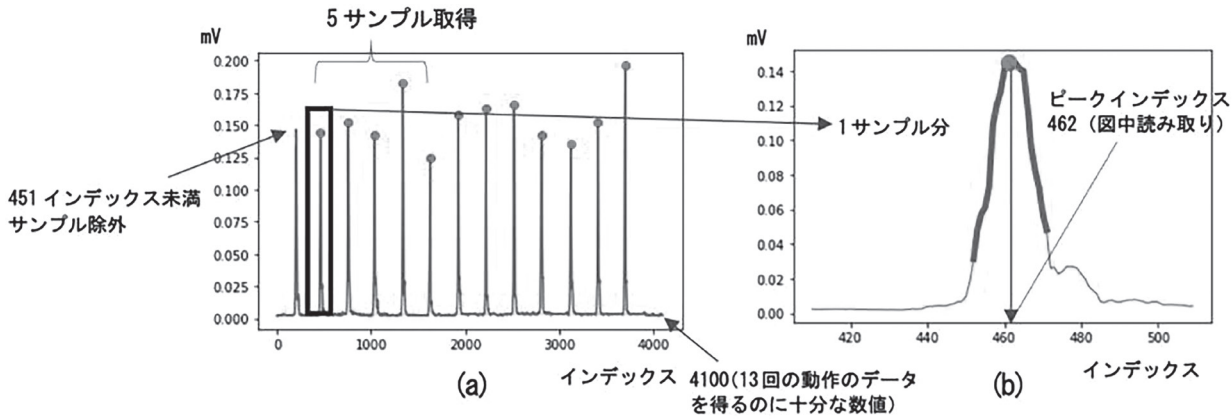

図２　事前学習の握り ARV データ取得。(a) 測定された4100個のデータ、13回動作を行うため13個のピークがある。初期設定で1動作5サンプル取得。(b) 握り1サンプル分拡大。太線はデータ20個取得部分。丸はピークを表し、ピークインデックスを基準にしてデータを取得する。

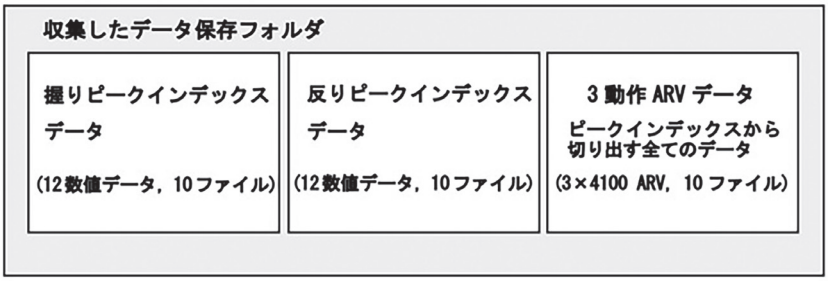

図３　収集したデータ（標準データ）のフォルダ中のファイル構成。握りと反りはピークを基準にデータを得るため、ピークインデックスデータ（ピーク位置データ）ファイルが必要。

2.4　事前学習データの取得

まず、使用する用語について定義する。図２(a)の2番目の波形（四角部分）を取り出した図２(b)、1つの波形からなる1行20列のデータを「1サンプル」と呼ぶ。1サンプルは、ピーク前10個、ピーク1個、ピーク後ろ9個の合計20個のデータを取得する。また、「1サンプル内」でデータを増やす方法もあるが、本論文では扱わない。

データ取得は、(1)13回握り、13回反り、動作なしを行い、握りと反りのピークインデックスデータ（ピーク位置データ）と3動作 ARV データを取得する工程、(2)そのデータから握りと反りはピーク位置をもとに（動作なしは異なる方法）15行20列の ARV からなる「事前学習データ」を切り出す工程に分けられる。(1)の握りと反りでは、図２(a)に示したように全てのデータ点4100個を得る。(1)の動作なしは、握りと反りと処理を揃えるため800個のデータ点を多めに複製し、先頭から4100個にする。(2)の握りと反りは、(1)で集めた4100個の ARV データから、451インデックス未満のサンプルを除く。そして、握りと反りのピークインデックスデータをもとに、指定した数（今回は5）だけ先頭からサンプルを取得する。なお、451インデックス未満を除くのは、測定開始時にノイズの発生や手の動きが不安定だからである。動作なしは握りや反りと比べてフラットな波形なので、どのインデックスからデータをとっても実用上問題ない。

動作なしデータはインデックス200から始まり200から220、201から221、202…のように5サンプルの ARV データを得る。

したがって、握り、反り、動作なしを各5サンプル、まとめると15行20列の ARV 行列データ（以下、事前学習データ）が切り出される。

2.5　収集データとデータ追加について

2.5.1　収集データ構成とデータ追加方法

図３に収集したデータのフォルダ中のファイル構成を示す。事前学習のデータ取得工程(1)において得られるデータ（握りと反りのピークインデックスデータと3動作 ARV データ）を10回分（10ファイル）収集した。

3動作 ARV データは、1行目が握り、2行目が反り、3行目が動作なしとなっている。今回除外されたのはどのファイルでも1サンプルであったため、3動作 ARV データを1ファイル切り出すと、握り、反り、動作なしそれぞれ12サンプルを得る。10ファイルあるため120サンプルが得られた。これを「標準データ」とする。

これらの収集したデータの追加方法としては、最初に、実験参加者に事前学習データを取ってもらい、次に、標

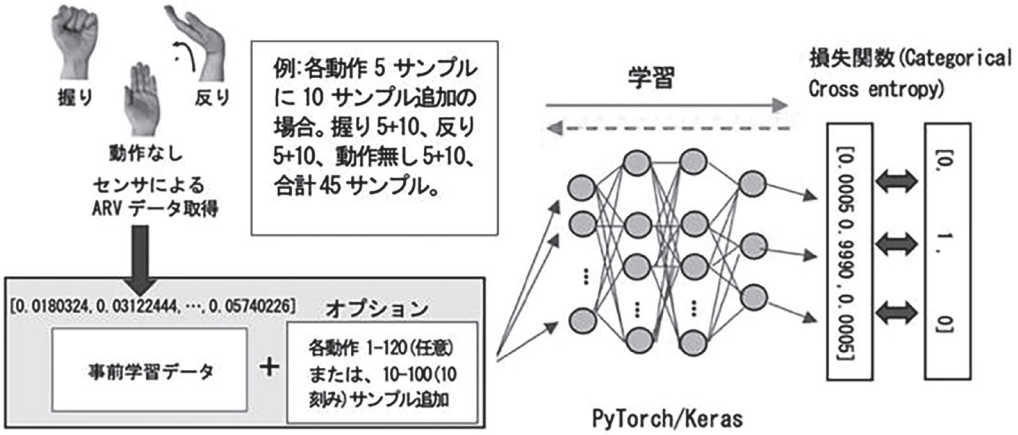

図4　データ取得から学習の流れ

準データからデータを取り出し追加する。

そのため、得られたデータはその人の特徴とその日の特徴を捉えている。さらに、標準データの追加によりデータ量が増し、精度がよくなることが考えられる。次節でデータ追加プログラムの説明を行う。

2.5.2　データ追加プログラムとその数理

作成したデータ追加プログラムは2パターンある。(1)データを120サンプルまで任意の数追加できるもの、(2)3動作 ARV データから1ファイルあたり10サンプルまで得られ、最大100サンプルまで追加できるものである。多くのデータを追加する場合、(1)の方式ではファイル構成の都合から何度も追加サンプルを指定する必要がある。そのため、細やかな調整が必要なとき以外は使用しておらず、通常は(2)の方式でデータを追加している。

次に、簡単のためにデータ追加の例を数式で表す。(1)と(2)の方式で核となる仕組みは同じである。今回、(2)の法式で10サンプル追加を考える。

事前学習データを、

$$\boldsymbol{X}_{train} = \begin{pmatrix} \boldsymbol{X}_{train1} \\ \boldsymbol{X}_{train2} \\ \boldsymbol{X}_{train3} \end{pmatrix}, \quad \boldsymbol{X}_{train} \in \boldsymbol{P}^{N\times D} \tag{1}$$

$\boldsymbol{P}=[0, 8.3]$(閉区間)、全サンプル数 $N=15$、1サンプルのデータ数 $D=20$である。$\boldsymbol{X}_{train1}\in\boldsymbol{P}^{k\times D}$) は握りデータ、$\boldsymbol{X}_{train2}\in\boldsymbol{P}^{k\times D}$ は反りデータ、$\boldsymbol{X}_{train3}\in\boldsymbol{P}^{k\times D}$ は動作なしデータである。各動作のサンプル数 $k=5$、全サンプル数は $N=3k$ である。

標準データから取り出した、追加するデータを、

$$\boldsymbol{X}_{add} = \begin{pmatrix} \boldsymbol{X}_{train1\_add} \\ \boldsymbol{X}_{train2\_add} \\ \boldsymbol{X}_{train3\_add} \end{pmatrix}, \quad \boldsymbol{X}_{add} \in \boldsymbol{P}^{3l\times D} \tag{2}$$

とする。$\boldsymbol{X}_{train1\_add}\in\boldsymbol{P}^{l\times D}$ は握り追加データ、$\boldsymbol{X}_{train2\_add}\in\boldsymbol{P}^{l\times D}$ は反り追加データ、$\boldsymbol{X}_{train3\_add}\in\boldsymbol{P}^{l\times D}$ は動作なし追加データである。10刻みの追加を考え、各動作追加サンプル数を例えば、$l=10$とする。

$$\boldsymbol{X}_{train\_added} = \begin{pmatrix} \boldsymbol{X}_{train1} \\ \boldsymbol{X}_{train1\_add} \\ \boldsymbol{X}_{train2} \\ \boldsymbol{X}_{train2\_add} \\ \boldsymbol{X}_{train3} \\ \boldsymbol{X}_{train3\_add} \end{pmatrix}, \quad \boldsymbol{X}_{train\_added} \in \boldsymbol{P}^{M\times D} \tag{3}$$

のように、同じ動作のデータの末尾に追加していく。1回各動作に対してデータを10サンプル追加したとき、$M=N+3l=45$となる。追加されて作成された新たなデータは、例えば握りでは(3)の四角の網掛け部分である。これを追加する回数だけ繰り返す。

2.6　深層学習

データ取得から学習の流れを図4に示す。図4はデータを追加しない「通常の学習」あるいはオプションで「データを追加して学習」がある。それ以外の工程は同じである。

次に、今回使用した全結合のネットワークのハイパーパラメータは以下に表3、表4で示す。ライブラリとして、Keras と PyTorch を使用した。

表3　Keras ハイパーパラメータ

隠れ層の数	隠れ層ニューロン数	活性化関数	最適化手法
2	1200	隠れ層全てReLU	SGD

epoch	バッチサイズ	損失関数	Dropout
2000	8	Categorical Cross Entropy	隠れ層全て0.5

表4　PyTorch ハイパーパラメータ

隠れ層の数	隠れ層ニューロン数	活性化関数	最適化手法
2	1200	隠れ層全てReLU	SGD

epoch	バッチサイズ	損失関数	Dropout
10000	8	Categorical Cross Entropy	隠れ層全て0.5

（a）モード選択画面　　（b）HORIZONTAL Game プレイ中

図５　ゲーム画面

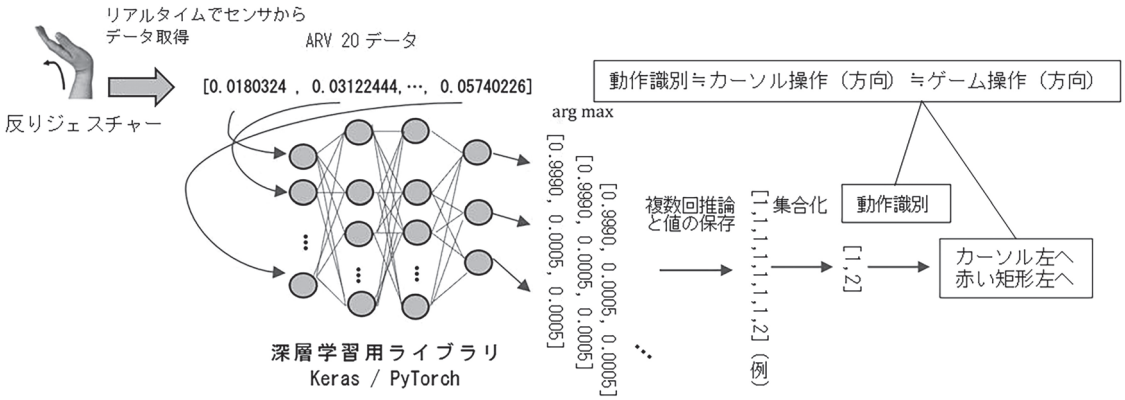

図６　反りのジェスチャーでの動作識別とカーソル・ゲーム操作

Keras の epoch 数は研究開始時の2000epoch から変わっていない。PyTorch の epoch 数は、事前学習データでは Keras と同じ2000epoch では収束しない。そのため10000epoch となっている。

2.7　理論的な計算量の導出

ここで、後の節で必要となるため、ビッグオー記法を用いて計算量を求める。計算コストが大きいデータ数とニューロン数について考え、他のハイパーパラメータは固定する。データ数 D を変化させニューロン数を固定した場合、計算量は O(D)である。一方、データ数を固定しニューロン数を変化させた場合、このネットワークでは隠れ層1・2層目の計算に最もコストがかかる。他の部分はそれと比べるとかなり小さい。したがって、隠れ層1・2層の計算量は、$O(H^{(1)}H^{(2)})$となる。ここで$H^{(1)}=1200$（隠れ層1層目のニューロン数）、$H^{(2)}=1200$（隠れ層2層目のニューロン数）である。$H^{(1)}=H^{(2)}$なので、$O(H^2)$とする。今回、データ数とニューロン数の一方を固定し変化させたときの学習時間を3.3節で述べる。

2.8　客観的指標のためのゲーム作成

これまで本研究では実際に行った動作とカーソルの動きの正しさから精度を測ってきた。今回、より客観的に動作識別の精度を測るためゲームを作成した。ゲームの動きとカーソルの動きは1対1に対応している。記録されたスコアを用いて動作識別精度の変化を調査する。以下で、「HORIZONTAL Game」、「DOUBLE-CLICK Game」、「HORIZONTAL&VERTICAL Game」のゲームモード別に、スコアの推移について述べる。ゲームについて詳しくは、投稿予定の別の論文[23]を参照してほしい。

このゲームを使い、追加サンプル数を変えたときのスコアを記録していった。図５に実際のゲームのモード選択画面とゲーム中の画面を示す。(b)は注釈（白色部分）を追加した。図５(a)がモード選択画面で、右下に深層学習用ライブラリの選択記録ボタンや追加比をファイルに記録するためのボタンがある。左下にはゲームを行った人を区別するためにマーカーを入れる機能がある。図６(b)は、ゲームプレイ中の画面で、中央が赤い矩形で右側が青い矩形となっている。握ると赤い矩形は右に動き、反ると左に動く。反りのジェスチャーでの動作識別とカーソル操作・ゲーム操作の概略を図６に示す。

表５　ゲームスコアの推移 Unknown は記録なし

実験日	実験参加者	HORIZONTAL	DOUBLE-CLICK	HORIZONTAL&VERTICAL	制限時間(秒)	深層学習用ライブラリ	ブレンド比
2024-08-22	著者B	1400	800	200	60	Keras	1:0
2024-08-22	著者B	2200	1200	300	60	Keras	1:0
2024-10-16	著者B	2000			60	Keras	1:0
2024-10-16	著者B	2200	1400	600	60	Keras	1:0
2024-11-13	著者B	1600			60	Keras	1:0
2024-11-13	著者B	1900			60	Keras	1:0
2024-12-10	著者B	1300			60	PyTorch	1:0
2024-12-10	著者B	1700	1700	500	60	PyTorch	1:0
2024-12-25	著者B	1200	2100	100	60	Unknown	Unknown
2024-12-25	著者B	1500		400	60	Unknown	Unkonwn
2025-01-8	著者B	2500	2700	1200	60	PyTorch	1:0
2025-01-8	著者B	2400	2200	1200	60	Keras	1:0
2025-01-8	著者B	2800	3600	1800	60	PyTorch	1:2
2025-01-22	著者B	2400	3900	1700	60	PyTorch	1:5
2025-02-19	著者B	3100	4300	1300	60	PyTorch	1:4

３．結果と考察

3.1　ゲームスコアの推移と動作識別精度

表５に著者Ｂのゲームスコアの推移を示す。なお、参考文献[23]にも、このデータを掲載している。そちらは深層学習用ライブラリ間の性能比較を行ったもので、本論文ではデータ追加に着目してほしい。表の「ブレンド比」とは、「事前学習データ」（15サンプル）にどれくらいサンプルを追加したかを比で表している。つまり、ブレンド比の前項は今回常に15サンプルで、後項は標準データから追加したデータ数となる。その比を簡単にして表に示している。例えば1：4だと15サンプルに60サンプル追加しており、全75サンプルとなる。前項は全て著者Ｂのデータで、後項は全て著者Ａが収集したデータである。

まず、結果を述べる。スコアは必要に応じて四捨五入した。HORIZONTAL Game は、データ追加前の平均スコアは1920点で、データ追加後は、2767点である。DOUBLE-CLICK Game ではデータ追加前の平均スコアは1667点で、データ追加後は3933点である。HORIZONTAL&VERTICAL Game は、データ追加前の平均スコアは667点で、データ追加後は1600点である。深層学習用ライブラリ変更も含まれているが、データ追加前と追加後を比べると全てのゲームモードで平均スコアが上昇している。また、2025年1月8日に着目すると深層学習用ライブラリ PyTorch を変えずにデータを追加してときがあり、その部分を見ると純粋にデータ追加後の方のスコアが上回っている。

最後に結果に対する考察を述べる。1月8日の結果からデータ追加単体でもスコアの上昇はいえる。しかし、それ以外の期間では Keras と PyTorch の切り替え時期（別の論文[23]で詳しく述べた）も重なっているため、どの程度データ追加が全ゲームモードの平均スコア上昇に寄与しているのかさらなる調査が待たれる。平均スコアは、DOUBLE-CLICK Game でデータ追加前に比べて2.3倍、HORIZONTAL&VERTICAL Game でデータ追加前に比べて2.5倍と著しい。HORIZONTAL Game は、1.4倍と前者2つのゲームよりは伸びていない。これは、DOUBLE-CLICK Game と HORIZONTAL&VERTICAL Game は HORIZONTAL Game と比べて難易度が高いため伸びしろが大きかったと考えられる。

以上から、全ゲームモードでデータ追加によりスコアが上昇し難易度が高いゲームほど影響が大きいことが分かった。また、データ追加によりスコアの平均が上がっており矩形の動きの精度が上がった。つまり、動作識別精度が上がったといえる。

3.2　epoch 数の決定

次の節で学習時間の図表を示す。ここで epoch 数の基準を決定する。データ数やニューロン数を変化させた場合、収束するまでの時間が異なると考えられる。今回は追加サンプル数やニューロン数ごとに収束するまでの epoch 数を求めて学習時間を測定することはしない。純粋なデータ数やニューロン数増加による学習時間を測定する。そのため、epoch 数の基準が必要である。そこで、事前学習データが収束するのに十分な epoch 数を基準とする。基準 epoch 数は2.6節の表３、表４で示した値、Keras は2000epoch、PyTorch は10000epoch とする。付録図７、図８に Keras と PyTorch の収束具合を示す。

3.3　データ数およびニューロン数の増加と学習時間

1節で述べたように指標作成のため理論的な計算量と学習時間にフィットする関数の種類の対応関係を明らかにする。具体的には、学習時間のデータから関数の種類を特定し、計算量 $O(D)$ と $O(H^2)$ が一致するかを調査する。

ここで、データ数とライブラリ間の学習時間を付録表６に示した。

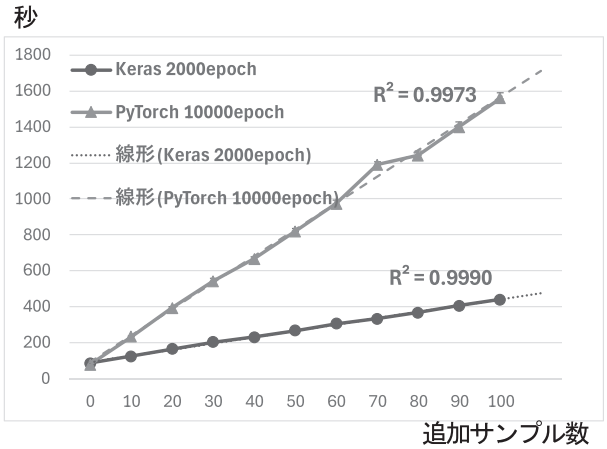

図９　データ数増加による学習時間の変化

図９に付録表６のデータをもとに、Keras と PyTorch の学習時間をプロットしたものを示す。

基準 epoch 数は3.2節で決定した。ニューロン数は1200で固定し、事前学習データに追加する標準データは、10、20、…、100の10パターンである。図のデータ点は、表３、表４のハイパーパラメータで追加サンプル数を変えながら10回学習時間を測定したときの平均値である。また、標準偏差を示すバーは、学習時間と比べてかなり小さいためほとんど見えない。実際の値は表６を参照してほしい。計算量は以降全て図の x 軸の範囲での検討とする。

図からデータ数の増加により学習時間が直線的に増加することが分かる。ここで、関数の種類を導き出すと Keras、PyTorch どちらも一次関数になっていると分かる。決定係数は $R^2=0.9990$（Keras）、$R^2=0.9973$（PyTorch）。これは、2.7節で挙げた O(D) の計算量と矛盾しない。

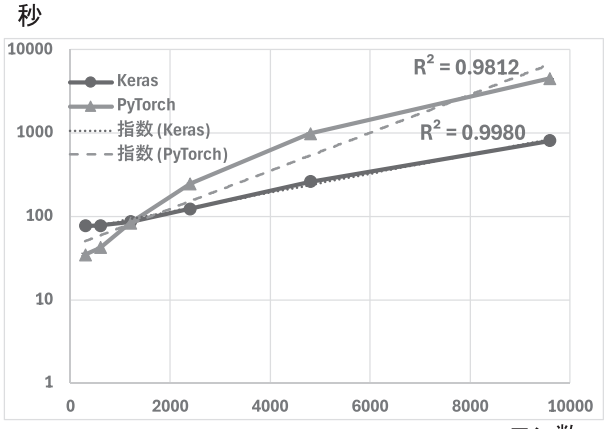

図10　ニューロン数増加による学習時間の変化

一方、ニューロン数増加における学習時間の変化を図10に付録表７の値をもとに x 軸を線形目盛、y 軸を対数目盛（片対数グラフ）にして、Keras と PyTorch の学習時間をプロットしたものを示す。事前学習データを使い、データ数を固定した。ニューロン数は、300、600、1200、2400、4800、9600の6パターンである。図のデータ点は表3、表4のハイパーパラメータでニューロン数を変えながら10回学習時間を測定したときの平均値である。また、標準偏差を示すバーは、学習時間と比べてかなり小さいためほとんど見えない。実際の値は表7を参照してほしい。

点線は Keras のプロットに対する近似曲線（指数近似）、破線は PyTorch のプロットに対する近似曲線（指数近似）を表している。決定係数は $R^2=0.9980$（Keras）、$R^2=0.9812$（PyTorch）である。

Keras のプロットはほぼ点線と重なり、指数関数的な増加である。一方、PyTorch のプロットは上に凸となり指数関数的な増加とはいえない。

PyTorch の曲線にフィットする関数を探索したいが目視では難しい。そこで、シンボリック回帰（PySR ライブラリ1.5.5）を用いた。

過学習しておらず、その中で最もスコアが高い関数（PySR の出力、Complexity＝5、Loss＝3.579e＋03、Score＝2.276e＋00）は、

$$y=(x_0*x_0)*4.861e\text{-}05$$

である。ここで x_0 は、ニューロン数を表す。
展開して整理すると、

$$y=4.851\times10^{-5}x^2$$

となる。式から明らかに2次の項が支配的であり、$O(H^2)$ がいえる。

４．制限事項

プログラムの検査を行った結果、PyTorch の学習において softmax＋CrossEntropyLoss（softmax が含まれる）の冗長な実装が見つかった。ただし、学習自体が不安定となる処理を行っても、データ追加による動作識別精度が向上したため、結果と考察ではそのままのデータを掲載する。冗長な処理を削除し、ゲーム操作を行ったが同様の点数であった。また、データ数およびニューロン数を変化させたときの学習時間を再測定した。学習時間は減少したものの関数の種類は同じであった。

５．結論

今回、ゲームを作成しスコア推移から客観的に動作識別精度向上を示した。スコアの推移を見ると、データ追加によるゲーム操作精度の向上、つまり動作識別精度の

向上が認められる。ただし、深層学習用のライブラリの切り替えと重なっていることから、その影響も考えられる。また、難易度の高いゲームほどスコア推移を比較しやすい。

次に、計算量と学習時間の関係を述べる。Keras の学習時間の関数の種類は、データ追加の計算量 O(D)は一致したが、ニューロン数の計算量に関して指数関数的な増加となり $O(H^2)$と一致しなかった。一方、PyTorch の学習時間の関数の種類はデータ追加の計算量 O(D)とニューロン数の計算量 $O(H^2)$で一致した。これらから、Keras と PyTorch の示されたデータ範囲での計算指標が得られた。

今回の知見を利用し、引き続き困っている人に役に立つ道具を作る研究開発を進めていきたい。

謝辞

この研究は1人では成り立ちませんでした。助けてくださった先生の皆様、センサ企業の担当者様、そして家族に感謝申し上げます。

参考文献

[1] 伊藤正剛，北本卓也（2023）「筋電位を利用した深層学習による手の動作識別について」山口大学教育学部研究論叢 第72巻 227-235

[2] 伊藤正剛，北本卓也（2024）「様々な機械学習を用いた手の動作識別について」山口大学教育学部研究論叢 第73巻 91-100

[3] 伊藤正剛，北本卓也（2025）「筋電位での機械学習によるリアルタイム動作識別について」山口大学教育学部研究論叢 第74巻 163-171

[4] C.Mビショップ著，元田浩，栗田多喜夫，樋口知之，松本裕治，村田昇監訳（2012）『パターン認識と機械学習 上』丸善出版

[5] Yutaro Hiyoshi, Yuta Murai, Yoshiko Yabuki, Kenichi Takahana, Soichiro Morishita, Yinlai Jiang, Shuta Togo, Shinichiro Takayama, Hiroshi Yokoi, (2018), “Development of Wireless Assistive Interface for Myoelectric Prosthetic Hands for Children”, Front Neurorobot. Aug 2: 12: 48

[6] 田中宏樹，奥村大，淡野公一（2007）「表面筋電位をFFT処理しないで動作識別する方法の検討」，電子情報通信学会論文誌，9 Vol.J90-D No.9

[7] 平岩明，内田典佳，下原勝憲，曽根原登（1994）「筋電操作ハンドの制御のための皮膚表面筋電位信号のニューラルネットによる認識」，計測自動制御学会論文集Vol.30，No.2，216/224

[8] 小野哲，松尾芳樹，上野祐樹（2016）「深層ニューラルネットを用いた筋電位に基づく前腕の動作識別」，ロボティクス・メカトロニクス講演会講演概要集，セッションID 1A2-13a6，p.1A2-13a6-

[9] 斎藤祐樹，伊藤桃代，伊藤伸一，福見稔（2022）「手首筋電に基づくタッピング動作の認識」，人工知能学会全国大会論文集，JSAI2022巻，第36回，セッションID 3F3-GS-9-04，p. 3F3GS904

[10] Kentaro Nagata, Keiichi Ando, Kazushige Magatani, Masafumi Yamada (2007) “Development of the hand motion recognition system based on surface EMG using suitable measurement channels for pattern recognition”, Annu Int Conf IEEE Eng Med Biol Soc 5214-7.

[11] Tianao Cao, Dan Liu, Qisong Wang, Jinwei Sun, (2020) “Surface Electromyography-Based Action Recognition and Manipulator Control”, Applied Sciences, No17,p.5823

[12] 西川大亮，兪文偉，横井浩史（2001）「表面筋電位からの動作識別システムにおけるオンライン型学習データ管理機構」，電気情報通信学会論文誌，J84-D-II，2634-2643

[13] 辻敏夫，市延弘行，伊藤弘司，長町三生（1993）「エントロピーを用いた誤差逆伝播型ニューラルネットワークによるEMGからの前腕動作識別」第29巻第10号 1213-1220

[14] Meng Z , Kang J (2023) “Continuous joint velocity estimation using CNN-based deep learning for multi-DoF prosthetic wrist for activities of daily living”, Front. Neurorobot. 17:1185052.

[15] Angkoon Phinyomark, Erik scheme (2018) “EMG Pattern Recognition in the Era of Big Data and Deep Learning”, Big Data Cogn. Comput. 2(3), 21

[16] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta, (2017) “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era”, arXiv: 1707.02968v2

[17] Jared Kaplan , Sam McCandish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, Dario Amodei, (2020) “Scaling Laws for Neural Language Models”, arXiv: 2001.08361v1

[18] 藤原健之，大見士，大村廉，石原尚子（2022）「深層学習を用いた保育士の行動認識における学習データ量と認識精度の関係」第84回全国大会講演論文集，(1)，167-168

[19] Viswanath Sivakumar, Jeffrey Seely, Alan Du, Sean R Vittner, Adam Bernzweig, Anuluwapo

Bolarinwa, Alexandre Gramfort, Michael I Mandel (2025) “emg2qwerty: A Large Dataset with Baselines for Touch Typing using Surface Electromyography”, arXiv:2410.20081v3 [cs.LG] 20 Mar
[20] emg2qwertyデータセットサイト https://github.com/facebookresearch/emg2qwerty（2025年4月30日アクセス）
[21] Ninapro（Non-Invasive Adaptive Hand Prosthetics）データセットサイト https://ninapro.hevs.ch/（2025年5月2日アクセス）
[22] UC2018 DualMyoデータセットサイト https://zenodo.org/records/1320922（2025年5月29日アクセス）
[23] 伊藤正剛，北本卓也（2026）「筋電位でのジェスチャー操作におけるKerasとPyTorchの性能比較」山口大学教育学部研究論叢 第75巻 投稿予定

付録

```
Epoch 1988/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0404
Epoch 1989/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0436
Epoch 1990/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0494
Epoch 1991/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0349
Epoch 1992/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0442
Epoch 1993/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0369
Epoch 1994/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0474
Epoch 1995/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0529
Epoch 1996/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0416
Epoch 1997/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0427
Epoch 1998/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0469
Epoch 1999/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0466
Epoch 2000/2000
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0341
```

図７　Keras（2000epoch）の学習状況

```
Epoch [9988/10000], [1/2], Accuracy: 1.0000, Loss: 0.5517
Epoch [9988/10000], [2/2], Accuracy: 1.0000, Loss: 0.5521
Epoch [9989/10000], [1/2], Accuracy: 1.0000, Loss: 0.5525
Epoch [9989/10000], [2/2], Accuracy: 1.0000, Loss: 0.5519
Epoch [9990/10000], [1/2], Accuracy: 1.0000, Loss: 0.5529
Epoch [9990/10000], [2/2], Accuracy: 1.0000, Loss: 0.5521
Epoch [9991/10000], [1/2], Accuracy: 1.0000, Loss: 0.5540
Epoch [9991/10000], [2/2], Accuracy: 1.0000, Loss: 0.5518
Epoch [9992/10000], [1/2], Accuracy: 1.0000, Loss: 0.5534
Epoch [9992/10000], [2/2], Accuracy: 1.0000, Loss: 0.5525
Epoch [9993/10000], [1/2], Accuracy: 1.0000, Loss: 0.5527
Epoch [9993/10000], [2/2], Accuracy: 1.0000, Loss: 0.5519
Epoch [9994/10000], [1/2], Accuracy: 1.0000, Loss: 0.5521
Epoch [9994/10000], [2/2], Accuracy: 1.0000, Loss: 0.5517
Epoch [9995/10000], [1/2], Accuracy: 1.0000, Loss: 0.5521
Epoch [9995/10000], [2/2], Accuracy: 1.0000, Loss: 0.5517
Epoch [9996/10000], [1/2], Accuracy: 1.0000, Loss: 0.5522
Epoch [9996/10000], [2/2], Accuracy: 1.0000, Loss: 0.5540
Epoch [9997/10000], [1/2], Accuracy: 1.0000, Loss: 0.5516
Epoch [9997/10000], [2/2], Accuracy: 1.0000, Loss: 0.5519
Epoch [9998/10000], [1/2], Accuracy: 1.0000, Loss: 0.5521
Epoch [9998/10000], [2/2], Accuracy: 1.0000, Loss: 0.5518
Epoch [9999/10000], [1/2], Accuracy: 1.0000, Loss: 0.5520
Epoch [9999/10000], [2/2], Accuracy: 1.0000, Loss: 0.5524
Epoch [10000/10000], [1/2], Accuracy: 1.0000, Loss: 0.5528
Epoch [10000/10000], [2/2], Accuracy: 1.0000, Loss: 0.5522
```

図８　PyTorch（10000epoch）の学習状況

表６　データ数とライブラリ間の学習時間
（10回測定の平均値±標準偏差）

追加データ数	Keras学習時間(秒) 平均±標準偏差	PyTorch学習時間(秒) 平均±標準偏差
0	84.3±1.6	77.0±3.6
10	123.5±3.1	232.0±4.8
20	163.8±3.5	391.8±7.9
30	201.9±4.2	542.1±14.9
40	230.3±4.5	664.3±14.2
50	265.6±5.9	818.1±17.8
60	304.5±7.8	971.8±20.1
70	333.1±11.2	1190.7±17.1
80	366.8±8.4	1242.7±24.4
90	405.1±12.0	1400.5±27.6
100	439.6±9.9	1559.6±30.4

表7　ニューロン数とライブラリ間の学習時間
（10回測定の平均値±標準偏差）

隠れ層ニューロン数	Keras学習時間(秒) 平均値±標準偏差	PyTorch学習時間(秒) 平均値±標準偏差
300	76.6±2.0	34.5±1.2
600	77.9±3.0	42.4±0.5
1200	86.7±1.6	83.2±1.2
2400	123.4±1.9	248.3±8.0
4800	262.8±3.1	987.5±52.9
9600	812.6±6.7	4514.6±65.5