修士論文

SNN におけるスパイク応答関数近似による学習性能への影響

令和 5 年度

三重大学大学院　地域イノベーション学研究科

博士前期課程　地域イノベーション学専攻

安井　雅哉

目次

表目次

図目次

Influence of Response Function Approximation on Learning Performance of Spiking Neural Networks

Masaya Yasui
March 2024

1. Introduction

Spiking neural networks (SNNs) are neural networks that use spikes (pulses) to represent information. Spikes are used not only for input and output signals, but also for signal transmission between units that make up the network. Units themselves have an ability to handle time-series information and are expected to process complex time-series signals. In addition, SNNs are highly adaptable to chip manufacturing and can be implemented using event-driven processing [1]. So, they consume less power than conventional neural networks. However, the operation of SNN units is more complex than that of non-spiking neural network (NN) units. This complexity makes SNNs difficult to scale up compared to non-spiking NNs.

Many attempts have been made to handle large NNs (both spiking and non-spiking). In the case of non-spiking NNs, various attempts have been made: devising new learning methods, parallelization, and network model. Especially, network model affect on network performance (learning and generalization ability). For example, the activation function, which is used to determine the output of the units of non-spiking NNs, is replaced from the biological sigmoid function, which used in conventional non-spiking NNs, to a function such as ReLU [2]. The replacement improves both the computational cost of operation and the learning performance in large-scale networks. For SNNs, various network models were discussed. But, the discussion is mainly done from the view point of modeling biological neurons. So, it is necessary to discuss network models from the viewpoint of the performance in large-scale networks.

In this paper, we discuss affect of the similar replacement on SNNs. Concretely, we discuss the effect of replacement of the spike response function (SRF), which is similar to the activation function, from the viewpoint of the relationship between network size and learning performance. SRFs are used to determine the internal states of units in SNNs. Our research group has discussed the effect of the replacement of SRF by its approximation. In reference [3], it was shown that the approximation brings the speedup of network operation. Since both computation cost and learning performance are important for NNs, we should discuss the effect of the replacement from the view point of learning performance. For example, the replacement should be inefficient if it degrades the learning performance. To discuss the learning performance of large SNNs, we conducted experiments with various scale of SNNs and various approximated SRFs.

2. Result and discussion

We have previously shown that exponential SRF can be accelerated by using a polynomial approximation [3]. In this paper, we investigate the leaning performance of SNN with approximated SRF. We discuss the change in learning performance by approximated SRF through experiments. In experiments, following SRFs are used:

・Original SRF function(refer as “orig”)

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau_s} exp\left(1 - \frac{t}{\tau_s}\right) & if\ t \geq 0 \\ 0 & otherwize \end{cases} \quad (1)$$

・Linear approximation of orig (refer as “linear”)

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau_s} & if\ 0 < t \leq \tau_s \\ \frac{\tau_s + K - t}{K} & if\ \tau_s < t \leq K + \tau_s \\ 0 & otherwise \end{cases} \quad (2)$$

Experiments were conducted on two different types of data sets, NMNIST and oxford. NMNIST is a data set for digits recognition task. For NMNIST data set, we evaluate the learning performance by the recognition accuracy. Oxford is a data set for drawing a picture on time-units plane. For Oxford data set, we evaluate the learning performance by the loss, which is used for

evaluating the progress of learning process. The less the loss is, the better the result is.

The results of the experiment are shown in Figures 1 and 2. These figures show the learning performance for various network size (the number of layers) using each type of spike response function (orig and linear).

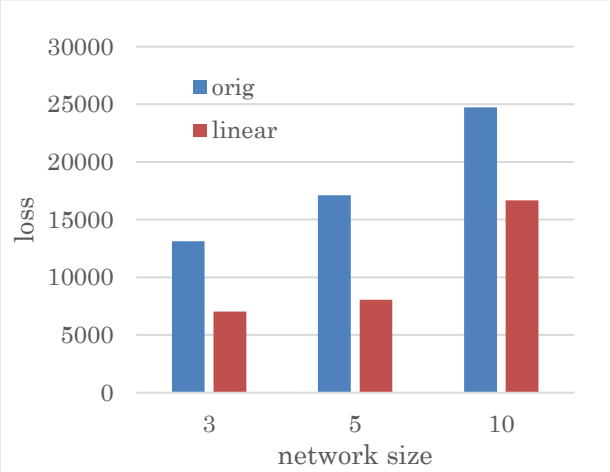

Figure1. Oxford experimental results

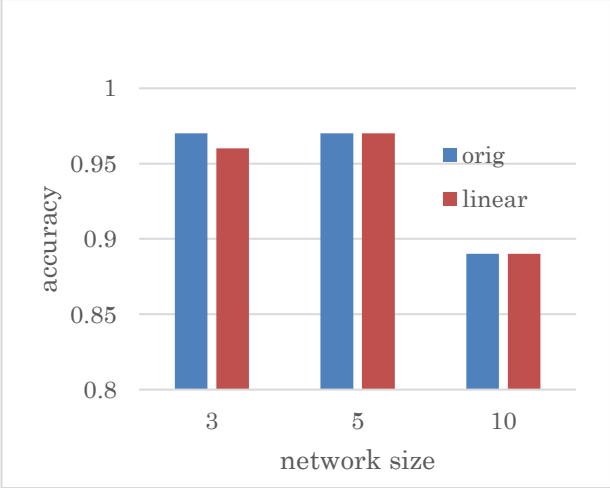

Figure2. NMNIST experimental results

From figure 1, all network sizes in the oxford dataset, using the approximation has lower values for the loss function and better learning performance than using the original function.

From figure 2, the 5-layer and 10-layer networks achieve the same learning accuracy with the approximation as with the original function, while the 3-layer network achieves a lower learning accuracy with the approximation than with the original, but the difference is 0.1%, which is almost the same.

These results show that the use of approximations does not degrade learning performance, and that some datasets, such as oxford, can achieve higher learning performance than using the original.

3. Conclusions

In this paper, we examined the effect of spike response function approximation on SNN in terms of learning performance. From the experimental results, it was found that the learning performance with the approximation of the spike response function was equivalent to or better than that with the original spike response function.

Combined with the reduction in computation time [3], the SRF approximation was found to be an effective method for constructing SNNs.

Acknowledgment

I would like to thank my advisor, Professor Haruhiko Takase, and the members of the Computer Engineering Laboratory for their advice and guidance in the preparation and presentation of this thesis. I would like to express my gratitude to them. Finally, I would like to express my gratitude to all the people involved in this research and this thesis.

References

[1] M. Pfeiffer and T. Pfeil: “Deep learning with spiking neurons: Opportunities and challenge”, Frontiers in Neuroscience, Vol.12, (2018).

[2] X. Glorot, A. Bordes and Y. Bengio: “Deep sparserectifier neural networks”, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Vol. 15, pp. 315–323 (2011)

[3] S. Ozaki, H. Takase, H. Kita, "Reducing computation time of Spiking Neural Networks by approximating response function - In the case of plural output spikes -", Proceedings of 52nd Tokai fuzzy workshop, S2-2 (2022).

[4] S. B. Shrestha and G. Orchard: “Slayer: Spike layer error reassignment in time”, Advances in Neural Information Processing Systems, Vol. 31, pp. 1419–1428 (2018).

[5] “bamsumit/slayerPytorch”, https://github.com/bamsumit/slayerPytorch, (last access on February, 2024).

第1章

序論

本研究では，近年注目を集めている情報処理技術の中でも，さまざまな分野で応用されているニューラルネットワーク（NN）に着目する．NN の中でもユニット間の信号のやり取りをスパイク（パルス）で行うスパイキングニューラルネットワーク（SNN）に着目し，ネットワークの大規模化を目指す．

1.1

はじめに

近年 AI 技術の発展により，機械学習分野に注目が集まっている．機械学習とは，データを分析する方法の一つであり，データからコンピュータが自動で学習し，データの背景にあるルールやパターンを発見する方法であり，AI を実現するためのデータ分析技術の一つである．機械学習は，画像認識や音声認識，自然言語処理など幅広い活用ができ，そのため医療や農業，マーケティング，ゲームなど多種多様な分野で活用することができる[1-4]．近年の話題になっている AI 技術としては，OpenAI 社による GPT という大規模言語モデルがある．GPT は簡単に言うと AI が大量のテキストデータを学習し，文章の生成や言語理解の能力を身につけた次世代言語モデルである．そして，GPT のような大規模言語モデルの社会的な影響は大きく，80%の労働者が，彼らの持つタスクのうち少なくとも 10%が大規模言語モデルの影響を受け，高賃金の職業，参入障壁の高い業界ほどその影響は大きくなると予測されている[5]．

このように，AI 技術やその応用は近年著しく発展しており，社会的なインパクトも大きい．本研究ではそんな AI 技術を支えている機械学習法の一つである NN について取り扱う．次節では，NN について紹介を行う．

1.2

ニューラルネットワーク（NN）

NN は 1943 年に McCulloch と Pitts による形式ニューロンと呼ばれるニューロンモデルの発表が起源としている[6]. 1958 年, Rosenblatt は形式ニューロンを基に単純パーセプトロンと呼ばれる初の NN が提案された[7]. 現在の NN が入力層, 中間層, 出力層で構成されているのに対して, 単純パーセプトロンは, 入力層と出力層の 2 層で構成されたシンプルな構造であるのにも関わらずシンプルなパターン認識問題などを解くことができたため注目を集めた. しかし, 単純パーセプトロンは線形分離不可能な問題を識別できないことを Minsky らによって指摘された[8]. この出来事により NN は一時的に世間の関心を失ったが, その後も研究は続けられた. そして, 1986 年に Rumelhart らによって NN の誤差逆伝搬法が発表された[9]. これは, 実際の出力と望みの出力（教師データ）を比較し誤差を計算し, その誤差を入力側の層へ伝搬していくことによってネットワーク全体のパラメータを学習（調整）する方法であった. この方法によって入力層と出力層の間に中間層を導入することが可能となり, 非線形分離問題を扱うことができるようになった. これにより, 再び NN に注目が集まるようになり, 4 層以上の深層 NN に誤差逆伝搬法を適応する試みがなされたが, 勾配消失や過学習, 局所解への収束などの問題が発見され, 90 年代後半には再び下火になった[10]. その後, 2006 年の Hinton らの層ごとの学習[11]などにより, 再び深層 NN の研究が盛んになり, 2012 年に Krizhevsky らの深層 NN が画像認識で従来手法を大きく上回る性能を示した[12]. これにより, 深層 NN の有効性が広く認知され, 画像認識や音声認識, 自然言語処理など現在でも広く活用されている分野へと拡大していった.

現在の一般的で広く用いられている NN として神経細胞の平均発火率の概念に基づいたニューロンモデルが用いられている. それとは別に神経細胞のダイナミクスに着目し, より生体の神経細胞を模したモデルであるスパイキングニューロンと呼ばれるニューロンモデルが研究されてきた. 代表的なものには, 1952 年に Hodgkin と Huxley による Hodkin-Huxley モデルがある[13]. その後も研究が続けられ神経細胞をモデル化する際の抽象化の程度が異なるさまざまなモデルが提案されている[14,15]. 次節ではスパイキングニューロンモデルが基礎となったスパイキングニューラルネットワークについて述べる.

1.3

スパイキングニューラルネットワーク（SNN）

NN の中でも，スパイキングニューロンによってユニット間の信号のやり取りをスパイク（パルス）で行う SNN が注目を集めている[16,17].

生体の脳は，ニューロン間の情報伝達をスパイク（パルス）状の信号によって行っており，スパイクを受け取ったニューロンは内部電位が変化する．このような動作をモデル化した NN が SNN である．SNN はスパイク（パルス）信号を伝達し動作を行い，スパイク信号を受け取ったユニット（スパイキングニューロン）は，応答関数に従い内部電位を上昇させ，入力が繰り返され内部電位があるしきい値に達したときにスパイクを出力するという動作をする．従来の NN が実数値で情報伝達を行っているのに対し，SNN は情報伝達をすべてスパイク信号で動作し従来の NN より実際の脳の動作の模倣度が高い NN となっている．そのため従来の NN より高度な情報処理能力が期待されており，実際にスパキングニューロンモデルは従来のシグモイド型のニューロンモデルより計算能力があるとする報告がある[18]．また，SNN はネットワーク内の情報伝達がすべてスパイクによって行われるためノイズに強く，ハードウェアでの実装に有利でチップ化との親和性が高くなっている．さらに，SNN はイベント駆動型での実装も可能であり，低消費電力や小型化での動作も期待されている[19].

SNN の情報表現手法は大きく 2 種類に分けることができる[20]．出力スパイクの密度や数により情報を表現する手法（rate coding）と出力スパイクの発火時刻により情報を表現する手法（temporal coding）である．応答速度の観点においては，密度を観測する時間がかからない後者の方が好ましいとされる[21]. Temporal coding を用いた SNN としてさまざまなモデルや学習法が提案されてきた[22-24]．その中の一つである Sumit らによる SLAYER という学習法について本研究では着目する[25]. SLAYER は一般的な誤差逆伝搬法を用いた学習法である．しかし，従来の誤差逆伝搬法と異なり SLAYER は誤差信号を過去に遡って再分配し学習を行う．これにより，時間的な性質を持つ SNN の特徴をさらに活かした形となり学習結果としても最先端の結果を記録している．また，SLAYER の著者らによって提供されている実装においては，GPU による並列計算などが用いられており高速化のための実装も実際にされている．

このように SNN における学習性能や高速化はさまざまな工夫がされ向上しており，ネットワークの大規模化に適応できそうな土台はできつつある．しかし，ネットワークが大規模

化するほどその複雑さは増していき，まだまだ性能向上の余地があるのが現状である．

1.4 研究目的

前節でも述べたように, SNN の大規模化については，さまざまな工夫がされている．また，従来の NN は活性化関数を置き換えることでネットワークの大規模化が実現されてきた．

そこで本研究では, SNN における応答関数近似による学習性能への影響の調査を目的とする．従来の NN は同期的に処理を行うため 1 回の学習につき各ユニットでの計算は 1 回で完了するが, SNN は非同期的に処理を行う性質上ユニットの計算を時々刻々と行う必要がある．このように SNN は従来の NN より複雑な計算を行っており，ユニットが層状に並び構成されその層を深くするほど処理能力が向上する NN の性質上，ユニットの計算量は処理時間に大きく影響する．そのため，ユニットの動作する上での計算量を削減することによって SNN 全体の計算量削減に大きく貢献することができると考えた. SNN 全体の計算量の削減は実際にハードに実装する上において大きく有利になってくる．そこで，ユニットの内部電位を計算に用いられる応答関数（スパイク応答関数）に着目しその近似を行った．先行研究において，応答関数近似により SNN の動作の高速化が可能であることが分かった[26]．しかし一般には，関数の近似は計算コストの削減には有効であるが性能を損なうという考えがある．そこで，学習性能の観点から応答関数近似が SNN をもたらす影響について検討を行う．

第 2 章では SNN の動作や学習法について説明を行い，第 3 章で本研究の検討手法である応答関数の近似と先行研究の結果を述べる．第 4 章で検討手法を基にした実験と結果を示し，第 5 章で本研究についてまとめを行う．

第2章

スパイキングニューラルネットワーク(SNN)

本研究ではSNNを構成するユニットがスパイクを処理するときに用いられる応答関数を近似することによる学習性能の観点からの評価を行う．そこで，本章では実際にSNNのユニットがどのように応答関数を用いて計算を行うのか,SNNの学習の仕方について本研究で用いるSLAYERについて説明しながら解説する．

2.1節ではSNNの構成や動作の仕方について，2.2節では今回用いるSLAYERという学習法を基にSNNの学習の仕方とSLAYERの特徴について述べる．

2.1

SNNのモデルと動作

まず,SNNの概形の一例やその動作について簡単に述べる．

SNNのネットワークは主にユニットが入力層，出力層，その間の中間層に分かれて層状に並びユニット同士が相互に接続して構成されている（図2.1）．

ユニット同士の結合間にはパラメータが付与されており，パラメータには信号の強さをコントロールする結合荷重や信号の入力タイミングをコントロールする時間遅れがある．入力や出力だけでなくネットワーク内のすべてのユニット間の情報伝達をスパイク信号で行う．

SNNを構成する各ユニットは，入力されたスパイクをスパイク応答関数に変換する（図2.2）．そして，そのスパイク応答関数は結合荷重などのパラメータなどによってその値を調整されたのちユニットの内部電位として時々刻々と加算される．内部電位があるしきい値に達したとき，ユニットはスパイクを出力する．その後，内部電位は不応期関数という関数によって急激に減少し，再び入力によって内部電位が加算される（図2.3）．

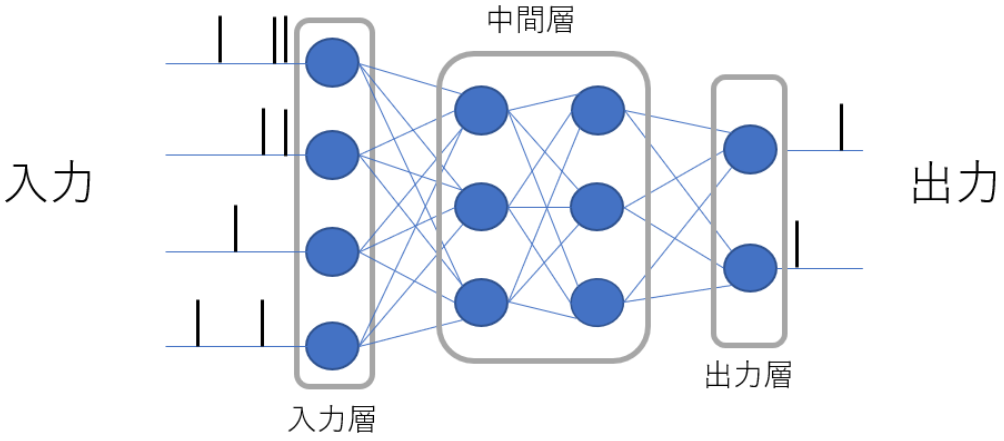

図 2.1 SNN のネットワーク構成

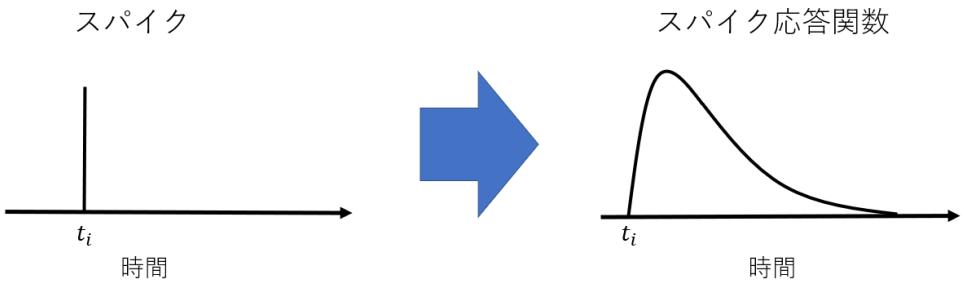

図 2.2 スパイクからスパイク応答関数への変換

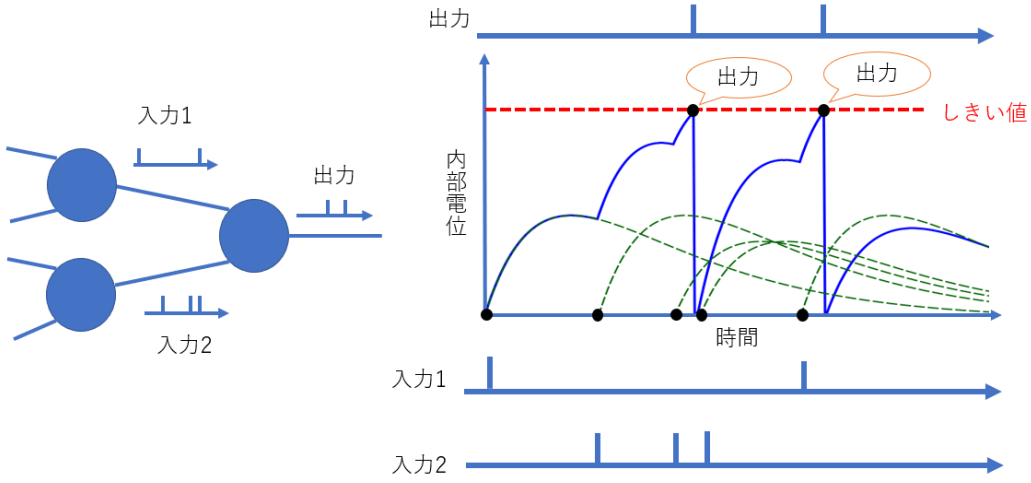

図 2.3 SNN のユニットのスパイクを入出力する際の動作

次に，ユニットの動作を詳細に示す.

SNN で情報伝達に使用されるスパイク列は式(2.1)のように定義される．ここで，i は i 番目のユニット，$\delta(\cdot)$ はディラック関数の δ 関数，$t_i^{(f)}$ はスパイク列を構成する f 番目のスパイクの時刻である.

$$s_i(t) = \sum \delta\left(t - t_i^{(f)}\right). \tag{2.1}$$

各ユニットは第 i 入力でスパイク列 $s_i(t)$ を受け取り，これを $\varepsilon(\cdot)$ で応答に変換した後，結合荷重 ω_i により重みづけする．$\varepsilon(\cdot)$ は，スパイク応答関数であり式(2.2)で表される.

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau_s} exp\left(1 - \frac{t}{\tau_s}\right) & if\ t \geq 0 \\ 0 & otherwize \end{cases} \tag{2.2}$$

また，連続した発火（出力）を防ぐため，自身の出力スパイク列に対して，内部電位を抑制する応答を生成する．この応答を生成する関数 $\nu(\cdot)$ を不応期関数と呼び式(2.3)のように表す．ユニットの内部電位 $u(t)$ は，これらの関数と畳み込み演算子 $(*)$ を用いて式(2.4)のように表される.

$$\nu(t) = \begin{cases} -2\theta \frac{t}{\tau_r} exp\left(1 - \frac{t}{\tau_r}\right) & if\ t \geq 0 \\ 0 & otherwize \end{cases} \tag{2.3}$$

$$u(t) = \sum w_i(\varepsilon * s_i)(t) + (\nu * s)(t) \tag{2.4}$$

各ユニットは，この内部電位があらかじめ定められたしきい値 θ を超えたときに発火し，スパイクを出力する．このように内部電位からスパイク列への変換を行う関数を $f_s(\cdot)$ とする．また，ユニット間の信号伝達には時間遅れが発生する．そこで，式(2.5)の $\varepsilon(t)$ は，時間遅れ d を含んだ $\varepsilon_d(t) = \varepsilon(t - d)$ を用いる.

多層 SNN において，第 l 層のユニット数を N_l 個，第 l 層から第 $(l+1)$ 層への結合

荷重を $\mathbf{W}^{(l)} \in R^{N_{l+1} \times N_l}$ とすると，ネットワークの 1 層分の動作は以下のように表される．

$$a^{(l)}(t) = \left(\varepsilon_d * s^{(l)}\right)(t) \tag{2.5}$$

$$u^{(l+1)}(t) = W^{(l)} a^{(l)}(t) + \left(\nu * s^{(l+1)}\right)(t) \tag{2.6}$$

$$s^{(l+1)}(t) = f_s\left(u^{(l+1)}(t)\right) \tag{2.7}$$

2.2

SNN の学習法

本節では，本研究で用いた SNN の学習法である SLAYER について述べる．

SLAYER が SNN を学習するときに用いる誤差関数 E は学習時間 $t \in [0, T]$ において式(2.8)で表される．

$$\mathrm{E} = \int_0^T L\left(s^{(n_l)}, \hat{s}\right) dt = \frac{1}{2} \int_0^T \left(e^{(n_l)}\left(s^{(n_l)}, \hat{s}(t)\right)\right)^2 dt \tag{2.8}$$

$L\left(s^{(n_l)}, \hat{s}\right)$ は時間 t における損失を表し，$e^{(n_l)}\left(s^{(n_l)}, \hat{s}(t)\right)$ は出力層における誤差信号を表す．以降，$e^{(n_l)}\left(s^{(n_l)}, \hat{s}(t)\right)$ は $e^{(n_l)}(t)$ とする．$e^{(n_l)}(t)$ は分類問題を学習する場合は式(2.9a)，それ以外は式(2.9b)を用いる．

$$e^{(n_l)}(t) = \left(\int_{T_{int}} s^{(n_l)}(t) dt - \int_{T_{int}} \hat{s}(t) dt\right), t \in T_{int} \tag{2.9a}$$

$$e^{(n_l)}(t) = \varepsilon * \left(s^{(n_l)}(t) - \hat{s}(t)\right) \tag{2.9b}$$

SLAYER の誤差逆伝搬法の学習は以下のパイプラインに従い行う．

$$e^{(l)} = \begin{cases} \dfrac{\partial L(t)}{\partial a^{(n_l)}} & if\ l = nl \\ \left(W^{(l)}\right)^{\mathsf{T}} \delta^{(l+1)}(t) & otherwize \end{cases} \tag{2.10}$$

$$\rho(t) = \frac{1}{\alpha} exp(-\beta|u(t) - \Theta|) \tag{2.11}$$

$$\delta^{(l)}(t) = \rho^{(l)}(t) \cdot \left(\varepsilon_d \odot e^{(l)}\right)(t) \tag{2.12}$$

$$\nabla w^{(l)} E = \int_0^T \delta^{(l+1)}(t) \left(a^{(l)}(t)\right)^{\mathsf{T}} \mathrm{dt} \tag{2.13}$$

$$\nabla d^{(l)} E = -\int_0^T \dot{a}^{(l)}(t) \cdot e^{(l)}(t) dt \tag{2.14}$$

式(2.12)の演算子 $\odot$ は時間方向における要素毎の相関演算, 式(2.11)の α, β は内部電位としきい値の差が学習に与える影響の大きさを調整する定数を意味している. 式(2.13), 式(2.14)を ADAM などの一般的な勾配降下法を用いてそれぞれのパラメータの最適化を行う.

ここまで学習法 SLAYER についての学習規則について述べた. SLAYER は文献[25]で学習データとして NMNIST や TIDIGITS, DVS Gesture などのさまざまな種類のデータセットを用いて学習を行った結果, 他の学習法と比べても高い学習精度を達成できている.

2.3

応答関数の問題点

ここまで SNN の動作の仕組みや学習方法について述べてきた. 情報伝達をすべてスパイク列で行い, 実数値で動作する従来の NN と比較してより生体の脳に近い動作をすることによって高い処理能力が期待されているが問題点もある. それは, 従来の NN と比べユニットごとの計算量が多いということである. 従来の NN は同期的にユニットの計算を行い, ユニ

ットは一回の学習において一回の計算で済む(図2.4). それに対し, SNN ではユニットが非同期的に動作し, ユニットにスパイク列が次々に入力され時々刻々と内部電位が変化するため, 時間毎にすべてのユニットの計算を行う(図 2.5).

さらに, ユニットの内部電位を計算する上でスパイクが入力されるごとに式(2.2)のような計算量の多い指数関数を含むスパイク応答関数を計算する必要がある. そのため, 従来の NN と比べ動作が複雑になり計算量が多くなるといった問題点がある.

それに対して, 従来の NN でも同様にユニットの内部状態を計算する活性化関数がネットワークの大規模化の障害となっており, 従来のＮＮでは活性化関数を簡略化することで大規模化を実現している.

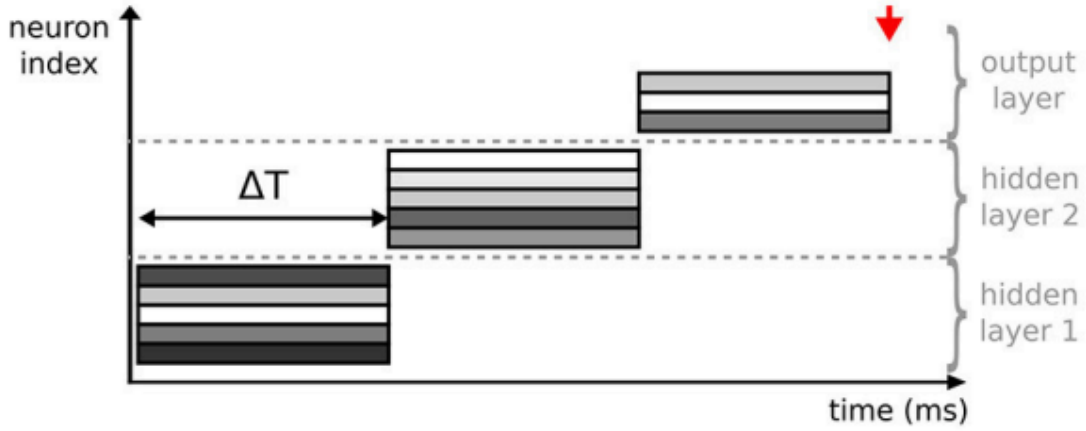

図 2.4 従来の NN での処理の様子 [19]

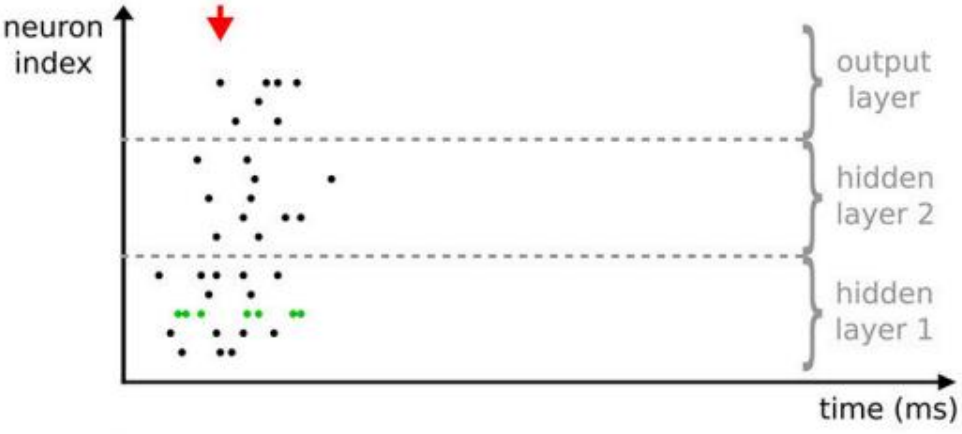

図 2.5 SNN での処理の様子 [19]

第3章

スパイク応答関数の近似

これまで, SNN の動作の仕方と学習方法と問題点について述べてきた. 我々は, 従来の NN が活性化関数を近似したのと同様に SNN の応答関数を近似することでネットワークの大規模化に貢献できるのではと考えた. そこで, 本章では SNN におけるスパイク応答関数の近似について, 先行研究[26]における提案手法とその結果について述べ, 検討を行う.

3.1

スパイク応答関数の近似による高速化

前章で, SNN における動作の複雑さと応答関数の関係について述べてきた. そこで, 先行研究[26]では, 動作の複雑さの一因である応答関数を近似することによる SNN の動作の簡略化, ひいては動作の高速化を目指した. 以下にその概略を示す.

続いて近似式について説明する. スパイク応答関数の近似式は式(3.1)(3.2)が提案された. 元々のスパイク応答関数は式(2.2)で原点から最大となる時刻 τ_s まで曲線的な単調増加で, その後 0 に漸近していき, 式(3.1)ではこの概形を保つように近似している. 具体的には, (1) 原点を通過する, (2) 時刻 τ_s で最大値 A となる, (3) 時刻 τ_s で傾きが 0 となる, (4) 時刻 K で 0 となる, (5) 時刻 K で傾きが 0 となる, 以上の五つの条件を満たす近似を行った. その結果, 式(3.1)は時刻 t の 4 次関数の近似式となった. しかし, 導出の過程から, K には式(3.1)で示すように有効範囲が限られてしまっている. そこで, 有効範囲が限られないようにするために, (1) 原点を通過する, (2) 時刻 τ_s で最大値 A となる, (3) 時刻 K で 0 となる, の三つの条件を満たす近似式 (3.2) を考案した. 式(3.2)は時刻 τ_s まで直線的な単調増加し, それ以降は時刻 K に対して直線的に 0 まで減少する概形となっている(図 3.2).

式(3.2)は, 近似前の関数である式(2.2)と比較するとかなり単純化してあることが分かるが, 従来の NN において活性化関数をシグモイド関数から ReLU 関数に変更することで学習精度の向上がみられた事例などがあり[27], 必ずしも関数の単純化が学習精度の低下を

招くわけではないと考えた.

$$\varepsilon(t) = \frac{A(3-K)}{\tau_s^4(k-1)^3} t(t-K\tau_s)^2 \left(t - \frac{2\tau_s(2-K)}{3-K}\right) \qquad (2 \le K \le 4, K \ne 3) \qquad (3.1)$$

$$\varepsilon(t) = \begin{cases} \dfrac{t}{\tau_s} & if\ 0 < t \le \tau_s \\ \dfrac{\tau_s + K - t}{K} & if\ \tau_s < t \le K + \tau_s \\ 0 & otherwise \end{cases} \qquad (3.2)$$

スパイク応答関数に式(3.2)を用いて 3 層のネットワークで oxford と NMNIST のデータセットを用いた SNN の処理時間と式(3.2)に近似する前との比較を表 3.1 に示す.

表 3.1 近似による高速化の検証 [26]

学習データ	近似前学習時間(A)	近似後学習時間(B)	近似前 tensor メソッドの占有時間(C)	近似後 tensor メソッドの占有時間(D)	近似による減少時間(E=C-D)
oxford	100	74.2	65.7	39.7	26.0
MNIST	235	210	47.7	27.3	20.4

表 3.1 より, oxford というデータセットにおいては約 26%, NMNIST というデータセットにおいては約 11%の高速化が実現された. また, 先行研究においては, スパイク応答関数だけでなく, 不応期関数の近似も行われており, 二つの近似を組み合わせることで学習時間が oxford では約 50%の短縮, NMNIST では約 15%の短縮が実現できている. このような先行研究の結果から, スパイク応答関数などのユニット内部の計算において近似を用いて簡素化できることによって SNN 全体の計算コストが減少し, 学習時間の高速化ができることが分かった.

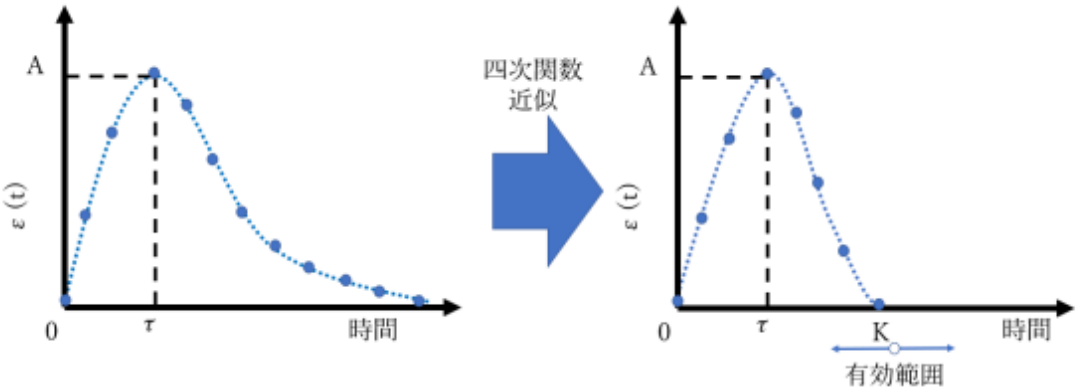

図 3.1 式(3.1)への近似前後のスパイク応答関数の比較 [26]

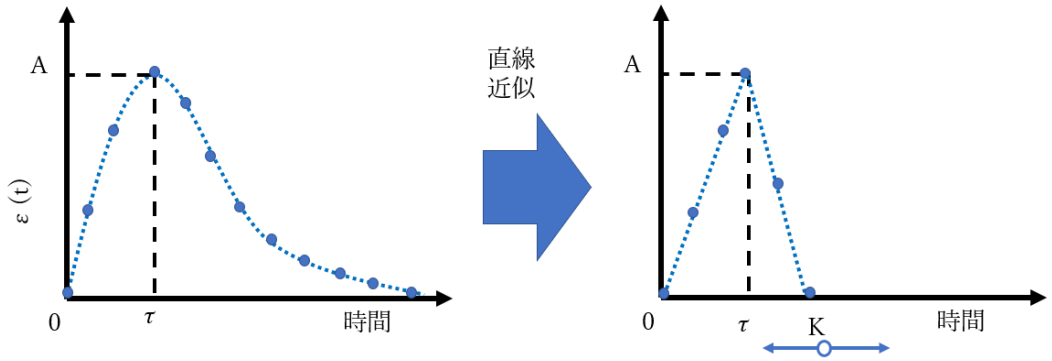

図 3.2 式(3.2)への近似前後のスパイク応答関数の比較

3.2

近似による学習性能の変化

ここまで, SNN の応答関数の近似によって高速化が実現できることについて述べてきた. しかし, SNN の大規模化において高速化だけではなく学習性能についても検討する必要がある. そこで, 本研究ではスパイク応答関数の近似を用いたときの SNN への影響を学習性能の観点から調査, 検討を行う. また, スパイク応答関数の近似については先行研究で用いた式(3.2)を用いる.

一般的に, 関数の近似は概形の類似度が下がるほどその影響が大きくなると考えられる. 式(3.1)は, 元のスパイク応答関数をなめらかな関数で近似しており, 元の応答関数の形状が, 式(3.2)と比べ残っている. 実際, 式(3.1)と式(3.2)を元の関数である式(2.2)と比較したときに, 式(3.1)はゆるやかに最大値に達し, 定数 K の値までゆるやかに 0 へ漸近していくが, 式(3.2)は直線で最大値に達し, 定数 K の値まで直線に 0 へ向かっていくため, 式(3.2)の方が式(2.2)との類似度が明らかに低いことが分かる. その式(3.2)を用いたときの方が, 式(3.1)を用いたときよりも学習性能の劣化が考えられる. そこで, 単峰性の関数であると言う特徴だけを残して近似した, 式(3.2)をスパイク応答関数の近似として用いる.

また, 先行研究においては, スパイク応答関数以外に不応期関数についても近似を行っているが, 本研究においてはネットワークサイズやパラメータの組み合わせ上実験の実施回数が膨大になってしまうため, スパイク応答関数のみの近似について検討を行った. SNN の動作上, ユニットの計算においてスパイク応答関数の方が不応期関数に比べ用いられる回数が圧倒的に多い点やスパイク応答関数のみの近似でも学習時間の短縮が実現できている点, スパイク応答関数の方が学習に及ぼす影響が大きいという点を考慮して, このような選択に至った.

第4章

実験方法

本章では, SNNの内部電位計算で用いられているスパイク応答関数を近似することによるSNN の学習性能について検討する. 一般的に近似を行うと計算コストは向上するが性能が落ちると考えられている. スパイク応答関数の近似がもたらす計算コストへの影響については向上することが分かっている. そこで, 今回はスパイク応答関数の近似による学習性能への影響について, さまざまなネットワーク規模において実験と考察を行う.

4.1

実験条件

学習データやネットワークパラメータについて述べる. 本研究では SLAYER の実装に付随して配布されている oxford と NMNIST の二つのデータセットを用いて実験を行う.

Oxford は, 図 4.1 のポアソン分布に従うスパイク列を入力として受け取り横軸に時間, 縦軸に出力ユニットを並べてできる平面上にネットワーク出力によって図 4.2 のような建物の絵を描くように学習するデータである. 図 4.1 と図 4.2 において, 横軸はスパイクの入力時刻・出力時刻であり, 縦軸は入力・出力ニューロンの番号を示している.

NMNIST は, 手書き文字認識用のデータセットである MNIST の画像をスパイクに変換しSNN用にした文字認識の問題のデータセットである. 34×34 の画素により描かれた文字を Dynamic Vision Sensor により撮影したものを, 画素毎に出現・消失の入力を用意し, 該当する時刻にスパイクを入力する. 図 4.3 は”5”を入力した際のある時刻における入力層への入力の例である. 赤色が出現・青色が消失・黒色が無入力に対応している. 出力は, 300ms の入力期間に対して, 当該文字種の出力ユニットは60個のスパイクを出力し, それ以外の出力ユニットは10個のスパイクを出力するように学習する. そして, 実際の出力において最もスパイク数が多かった出力ユニットに該当する文字種（数字）が分類結果となる.

Oxford と NMNIST のデータセットを用いて実験を行う際の固定のネットワークのパラメータはそれぞれ表 4.1, 表 4.2 を用いる. また, 本研究ではネットワーク規模を変更しての

実験を行っているがその際のネットワークにおける中間層のユニットの数は各データセットにおいてすべて統一して行った. 学習回数は Oxford では 10000 回, NMNIST では 300 回に統一して実験を実施した.

学習性能については, NMNIST はどれだけ正確に文字を認識できたかという認識精度, oxford は教師データと実際の出力のズレの大きさを表す損失関数（式(2.8)）の値を基に評価を行った.

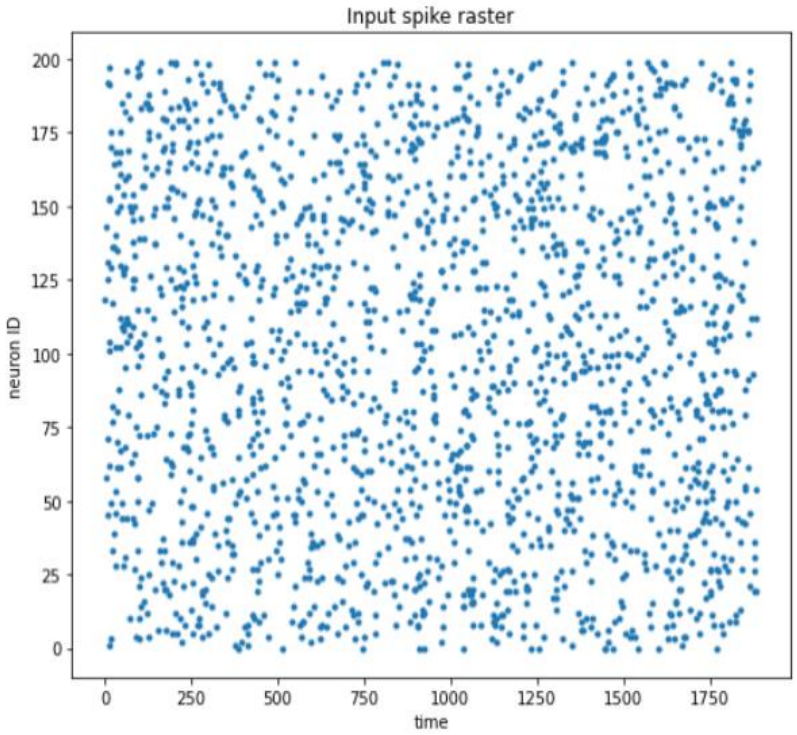

図 4.1 Oxford の入力スパイク

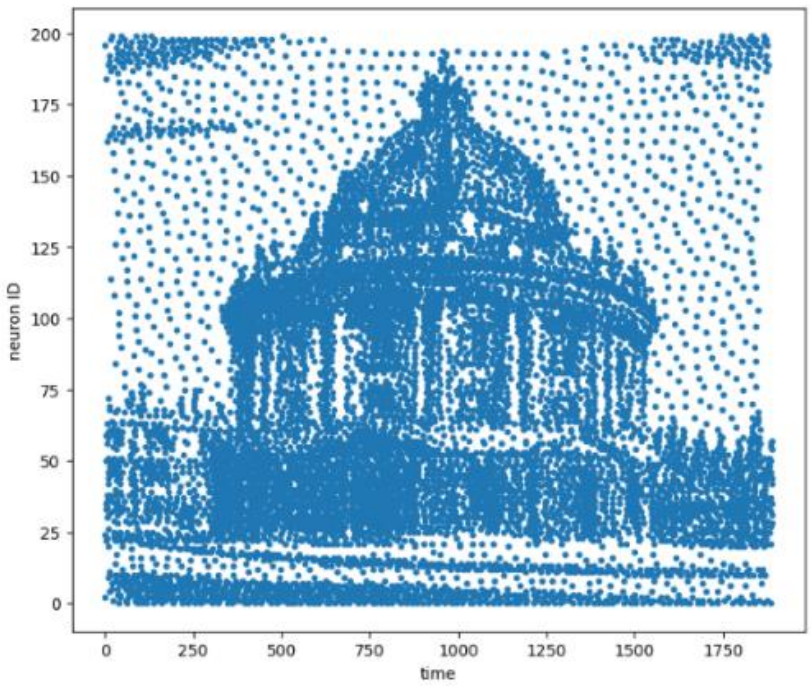

図 4.2 Oxford の望みの出力スパイク

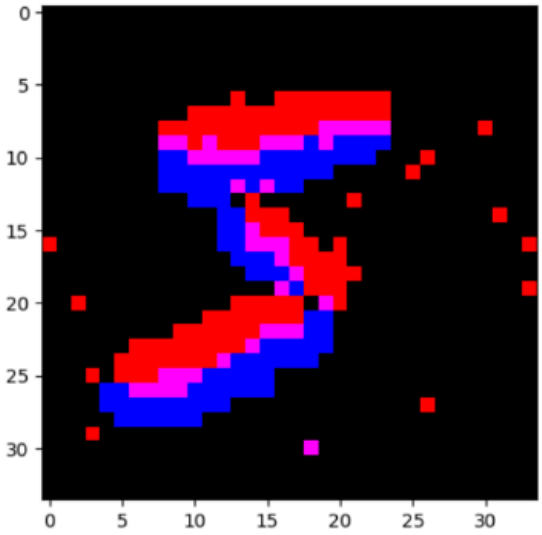

図 4.3 NMNIST の入力例

表 4.1 ネットワークパラメータ（oxford）

入力層ユニット数	200
中間層ユニット数	256
出力層ユニット数	200
テーブル間隔	1
処理時間	1,900
閾値Θ	10
スパイク応答関数 パラメータA	1
τ_r	1

表 4.2 ネットワークパラメータ（NMNIST）

入力層ユニット数	34×34×2
中間層ユニット数	512
出力層ユニット数	10
テーブル間隔	1
処理時間	300
閾値Θ	10
スパイク応答関数 パラメータA	1
τ_r	1

4.2

実験１：スパイク応答関数のパラメータ　τ_s の決定

本節では，スパイク応答関数の置き換えでは，式(2.2)において最も性能が良い時定数のものを置き換え対象としたい．そこで，その値を決定するために実験を通じ，検討を行う．

スパイク応答関数は式(2.2)で表され，時定数 τ_s によってその形を大きく変える．そして，スパイク応答関数は各ユニットの出力を行う際に用いられるためその形は学習に大きく影響を与える．また，学習データやネットワークの規模によって最適なスパイク応答関数の形は異なってくる．そこで，本節の実験では 3, 5, 10 層のネットワークの規模における最適な時定数 τ_s のパラメータの値を決定するため，それぞれのネットワーク規模において時定数 τ_s の値を変更させながら実験を行った．なお，認識精度は高い方が良く，損失関数の値は小さい方が良い．

4.2.1

実験方法

Oxford と NMNIST の二つのデータセットにおいてそれぞれ 3, 5, 10 層のネットワーク規模で τ_s の値を変化させながら各ネットワーク規模において 15 回ずつ実験を行う．実験は τ_s を 1～20 の範囲で変化させて NMNIST は認識精度を, oxford は教師データと実際の出力の差を表す損失関数の値を基にして評価を行った．また，損失関数はその性質上 τ_s の値によって大きく値が変化する．そのため異なる τ_s でも一律に評価するために，評価を行う際にはすべて τ_s＝10 で統一して行った．

4.2.2

実験結果

本節では，実験１の oxford と NMNIST のデータセットを用いた実験の結果を述べる．

4.2.2.1

Oxford

Oxford のデータセットを学習させたときのそれぞれのネットワーク規模における実験結果を図 4.4～6 に示す. これらの図は横軸が τ_s の値で, 縦軸が損失関数の値, 各 τ_s の値での損失関数の値をプロットした図である. 図 4.4 は 10 層, 図 4.5 は 5 層, 図 4.6 は 3 層のネットワーク規模においての結果である　　.

図 4.4～6 が示すように, 時定数τ_sの値が変化することによって学習結果が大きく変わることが分かる. スパイク応答関数の時定数を変化させることは, スパイク応答関数の形を大きく変化させることと同様であり, 図 4.4～6 の結果よりスパイク応答関数の形は学習性能に大きく影響を与えることが確認できる.

また, この結果からそれぞれの層におけるもっとも損失関数の値が低かったパラメータ（時定数）の値を表 4.3 に示す.

表 4.3 各ネットワーク規模における近似で用いる τ_s の値

ネットワーク規模	τ_sの値	Loss の最小値	Loss の平均値
3	5	12365	13111
5	1	14940	17100
10	2	13820	24727

表 4.3 よりベストなτ_sでの損失関数の最小値は層の大きさに関わらず大きく変化しないことが分かる. しかし, 損失関数が最小になるτ_sの値の損失関数の平均値はネットワーク規模が拡大するほど大きくなっている.

また, 図 4.4～6 より τ_s がベストの値でないとき以外の損失関数の値はネットワーク規模が拡大するほど大きくなっていることから, ネットワークが大規模になるほど適切な学習性能を保てるパラメータの範囲が狭くなっていくことが考えられる.

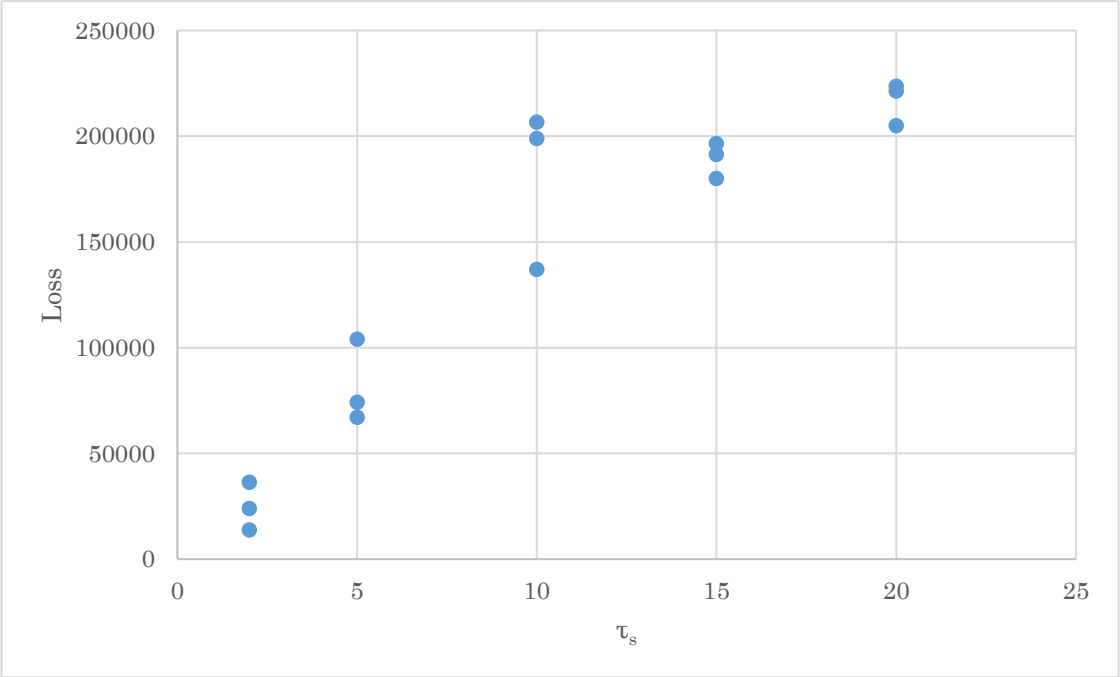

図 4.4 10 層における τ_s を変化させたときの損失関数の値

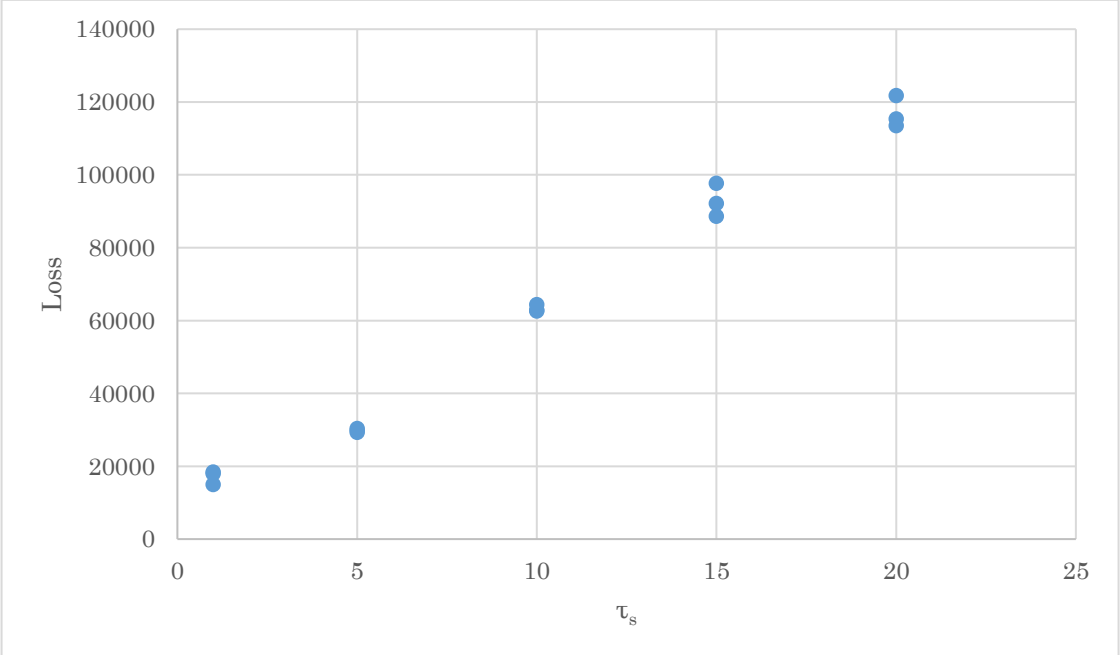

図 4.5 5 層における τ_s を変化させたときの損失関数の値

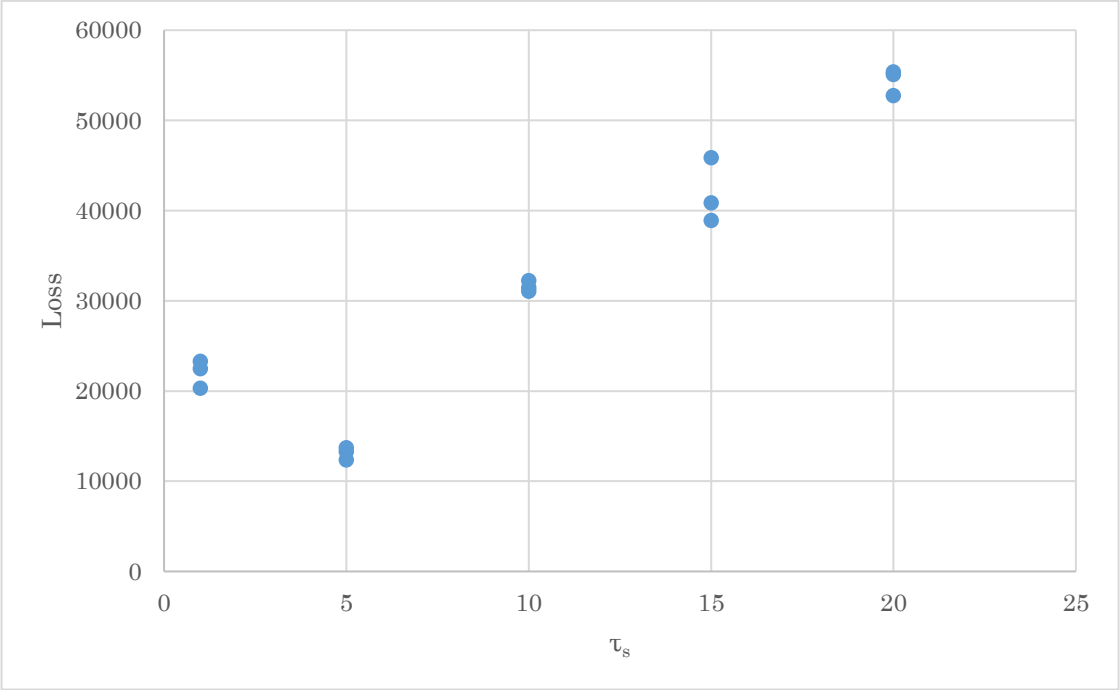

図 4.6 3 層における τ_s を変化させたときの損失関数の値

4.2.2.2

NMNIST

NMNIST のデータセットを学習させたときのそれぞれのネットワーク規模における実験結果を図 4.7～9 に示す. これらの図は横軸が τ_s の値で, 縦軸が認識精度の値, 各 τ_s の値での認識精度の値をプロットした図である. 図 4.7 は 10 層, 図 4.8 は 5 層, 図 4.9 は 3 層のネットワーク規模においての結果である.

図 4.7～9 が示すように, NMNIST という分類のデータセットを学習するさいにおいてもスパイク応答関数の形は学習性能に大きく影響することが分かる. 特に 10 層のネットワークの場合最大で認識精度が約 40%も開きがあり, 3 層と 5 層における認識精度の開きが約 10%くらいであるのと比較するとかなり大きな差であることが分かる. このことから, oxford の実験と同様に層が大きくなるほどネットワークが複雑になり学習が難しくなっていると言える.

また, この結果から決定したそれぞれの層におけるパラメータ（時定数）の値を表 4.4 に示す.

表 4.4 各ネットワーク規模における近似で用いる τ_s の値

ネットワーク規模	τ_s の値	認識精度の最高値	認識精度の平均値
3	1	0.98	0.97
5	5	0.99	0.97
10	5	0.91	0.89

表 4.4 から 3 層と 5 層のネットワークは最大で 98～99%の精度で学習できているのに対し, 10 層は 91%と約 10%の差があることからも層が大きくなるほど学習が難しくなっているということが言える.

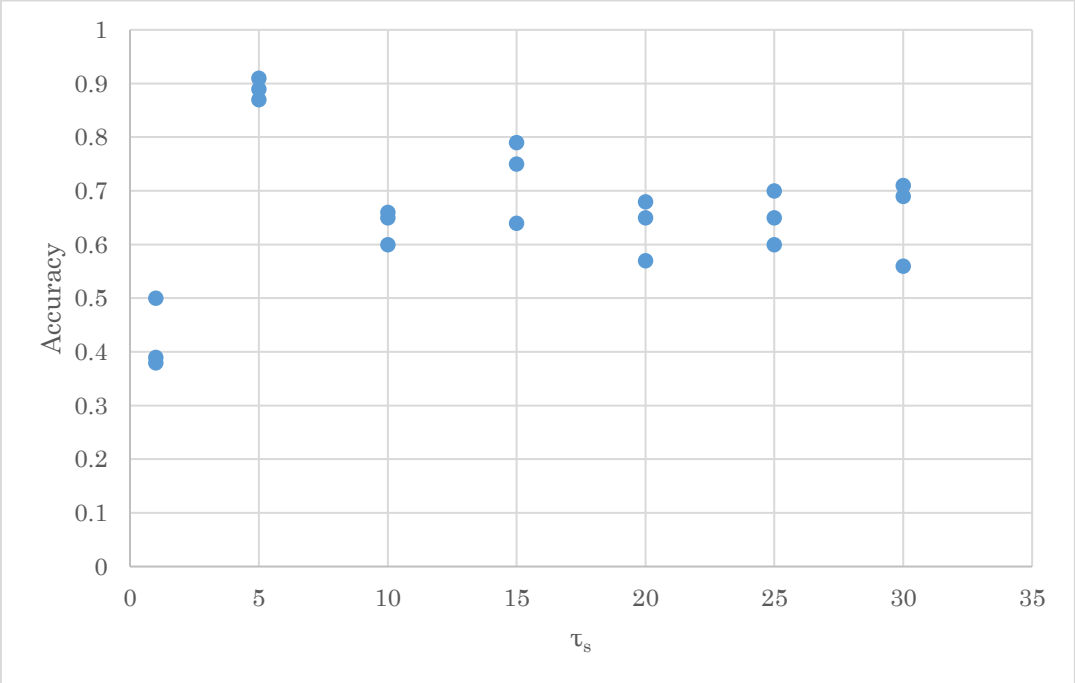

図 4.7 10 層における τ_s を変化させたときの認識精度

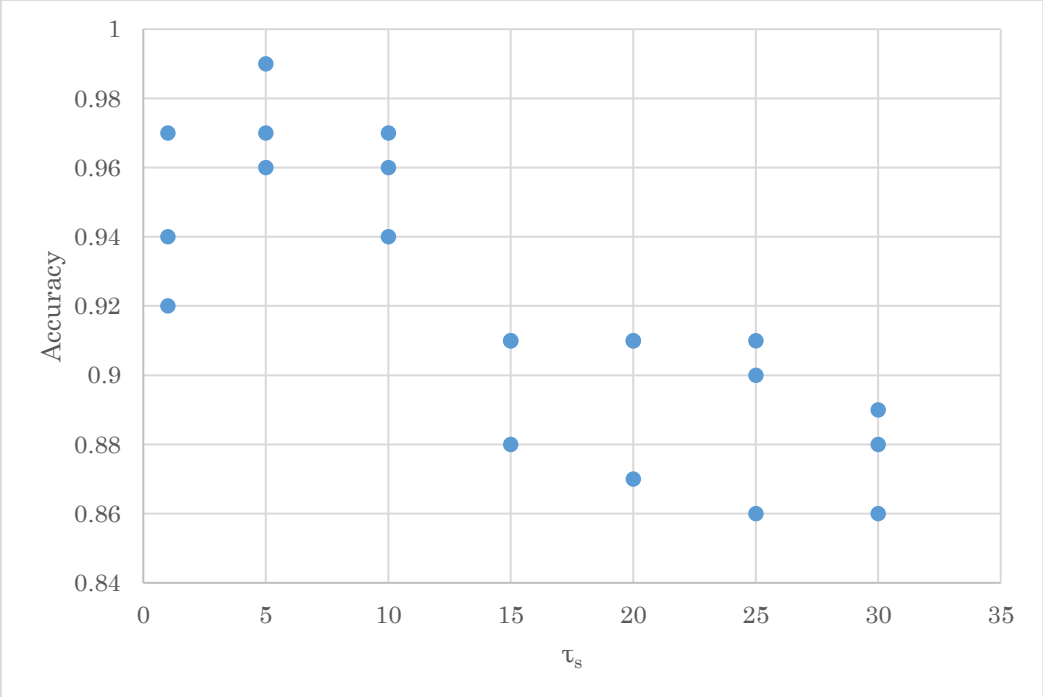

図 4.8 5 層における τ_s を変化させたときの認識精度

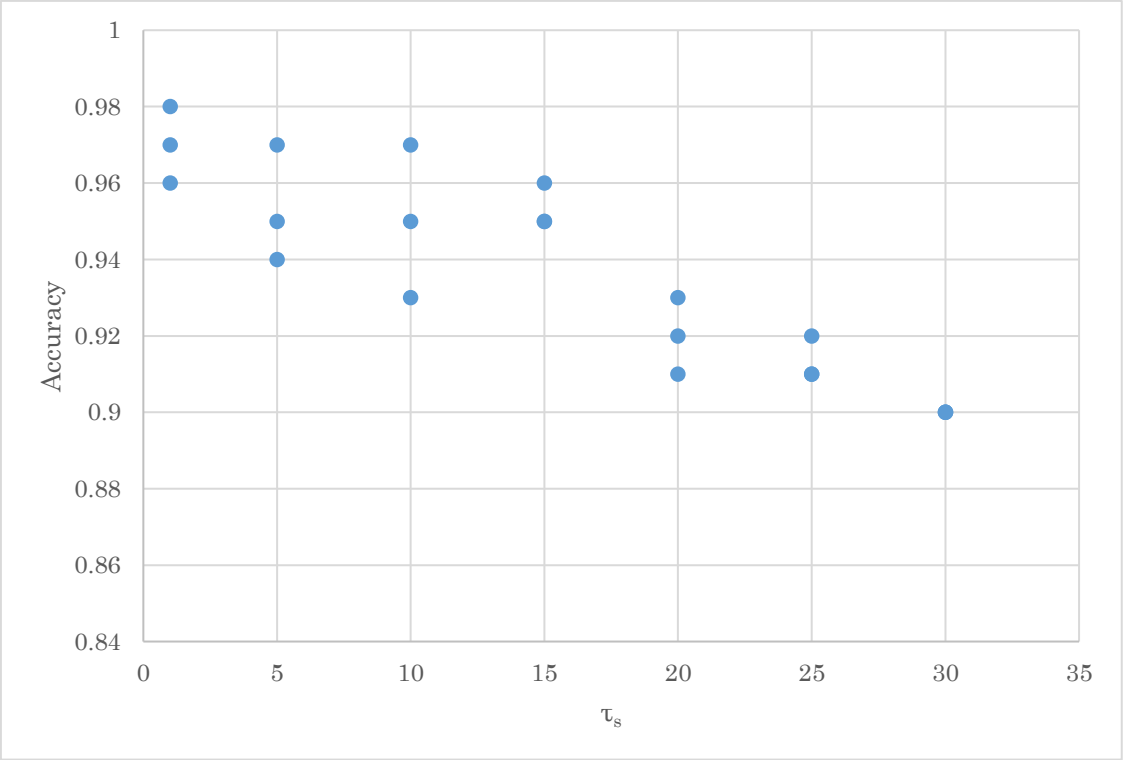

図 4.9 3 層における τ_s を変化させたときの認識精度

4.2.3

考察

図 4.4〜4.6, 図 4.7〜4.9 から表されているようにスパイク応答関数の時定数(パラメータ)を動かして形を変更しながら実験を行った結果 NMNIST と oxford の種類が違うどちらのデータセットにおいてもスパイク応答関数の形が学習性能に大きな影響があることが分かった. ネットワークの規模が大きくなるほどベストなパラメータの値から外れたときの学習性能が大きく落ち込み, 10 層のネットワークにおいて oxford では損失関数の値の差が最大 15 倍以上, NMNIST では認識精度が 40%以上の差が出ていることが分かる. これは, ネットワークが大きくなるほど動作が複雑になり, 適切なパラメータの範囲が狭くなっていると考えられ, SNN のネットワークの大規模化は依然として難しくなっているといえる.

また, 表 4.3 より oxford の最も学習性能が高い τ_s の値が, 層が大きくなると小さくなるのに対し, NMNIST は表 4.4 より層が大きくなると最も学習性能の高い τ_s の値が大きくなることから適切なスパイク応答関数の形は学習させるデータセットの種類によっても大きく変わることが分かる.

4.3

実験 2：スパイク応答関数の置き換えの学習性能への影響

本節では, 実験 1 で決定した τ_s を基にスパイク応答関数を式(2.2)から式(3.2)に置き換えて実験を行いスパイク応答関数の置き換えによる学習性能への影響を調査する.

式(3.2)における τ_s は実験 1 で決定した値に設定し, K を変化させていくことによってスパイク応答関数の形を変えていき, 直線で構成された式(3.2)が SNN の学習性能に与える影響を実験 1 と同様に 3, 5, 10 層のネットワークでそれぞれ学習させて実験を行った.

4.3.1

実験方法

実験 1 と同様に oxford と NMNIST の二つのデータセットにおいてそれぞれ 3, 5, 10 層のネットワーク規模で, K の値を変化させながら各ネットワーク規模において 15 回以上ずつ

実験を行う．本節の実験ではスパイク応答関数に式(3.2)を組み込み，式(3.2)における τ_s は実験１で決定した各ネットワークにおけるそれぞれの値（表 4.3-4）を用い，K を変化させることによって式(3.2)が 0 になる地点を変更させていった．それにより，直線で構成されたスパイク応答関数への置き換えが SNN にもたらす学習性能の影響を調査した．K の値は NMNIST では 0～40, oxford では 0～25 の範囲で実験を行った．

また，実験の評価としては実験 1 と同様に NMNIST では認識精度を，oxford では損失関数の値を基にし，損失関数においては τ_s = 10 で統一して行った．

4.3.2

実験結果

本節では，実験２の oxford と NMNIST のデータセットを用いた実験の結果を述べる．

4.3.2.1

Oxford

Oxford のデータセットを学習させたときのそれぞれのネットワーク規模における実験結果を図 4.10～12 に示す．これらの図は横軸が K の値で，縦軸が損失関数の値であり，各 τ_s の値での損失関数の値をプロットした図である．図 4.10 は 10 層，図 4.11 は 5 層，図 4.12 は 3 層のネットワーク規模においての結果である．

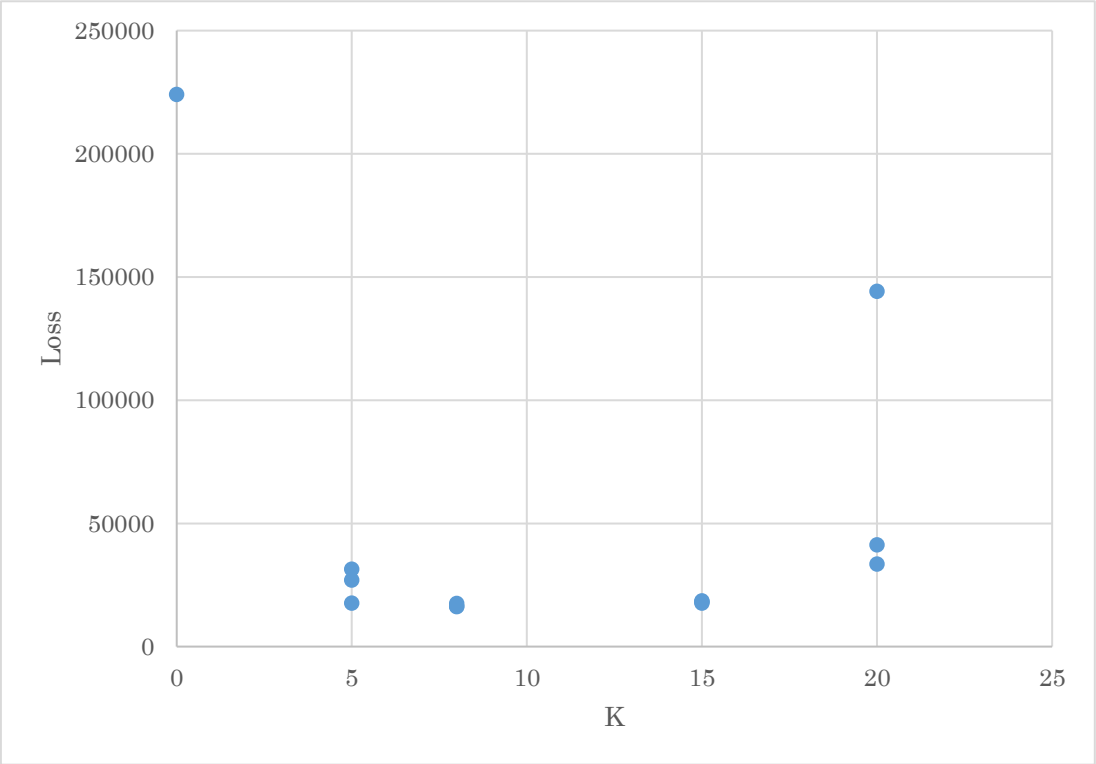

図 4.10 10 層における K を変化させたときの損失関数の値

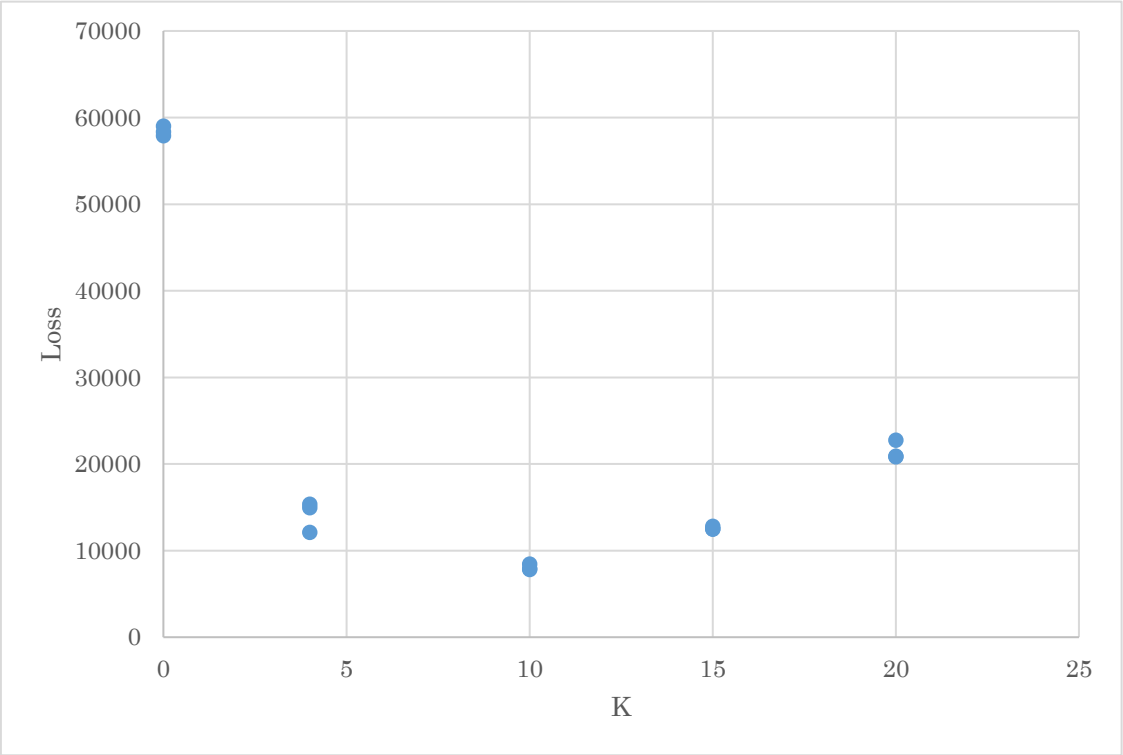

図 4.11 5 層における K を変化させたときの損失関数の値

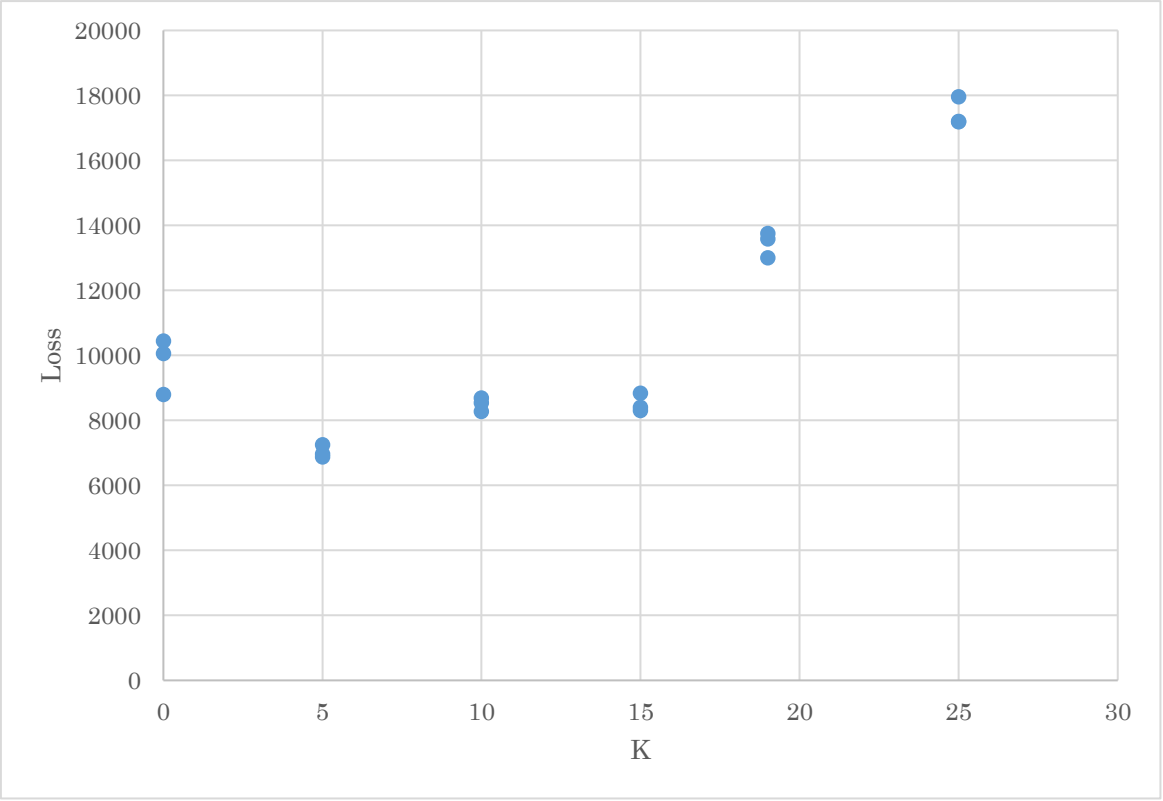

図 4.12 3 層における K を変化させたときの損失関数の値

また, それぞれのネットワーク規模における損失関数の値が最小になった K の値とその時の損失関数の最小値, 平均値を表 4.5 に示す.

表 4.5 各ネットワーク規模における損失関数が最小になる $\boldsymbol{K}$ と
その時の最小値と平均値

ネットワーク規模	K の値	Loss の最小値	Loss の平均値
3	10	8280	8504
5	10	7807	8050
10	8	16204	16656

図 4.10～12 から, 実験 1 と同様にスパイク応答関数の形を変数 K で変化させていくと損失関数が低く抑えられる K の範囲があり, ネットワークの規模が大きくなるほどその範囲から外れたときの損失関数の値が増大する傾向がある.

表 4.5 から, 損失関数の最小値は 3 層と 5 層のネットワークは大きく変わらないが 10 層は 3 と 5 層のときと比べてかなり大きくなっている. また, 損失関数の平均値を見ると最小値から大きく差がないことが分かり, これは学習結果におけるバラつきが少なく再現性が高くなっていることが言える.

4.3.2.2 NMNIST

NMNIST のデータセットを学習させたときのそれぞれのネットワーク規模における実験結果を図 4.13～15 に示す. これらの図は横軸が K の値で, 縦軸が認識精度の値であり, 各 τ_s の値での認識精度の値をプロットした図である. 図 4.13 は 10 層, 図 4.14 は 5 層, 図 4.15 は 3 層のネットワーク規模においての結果である.

図 4.13 から oxford と同様に直線で構成されたスパイク応答関数において K によって認識精度が大きく変動している. しかし, 図 4.14, 4.15 から 3 層, 5 層のネットワークにおいては認識精度の差に大きな差は見られなかった.

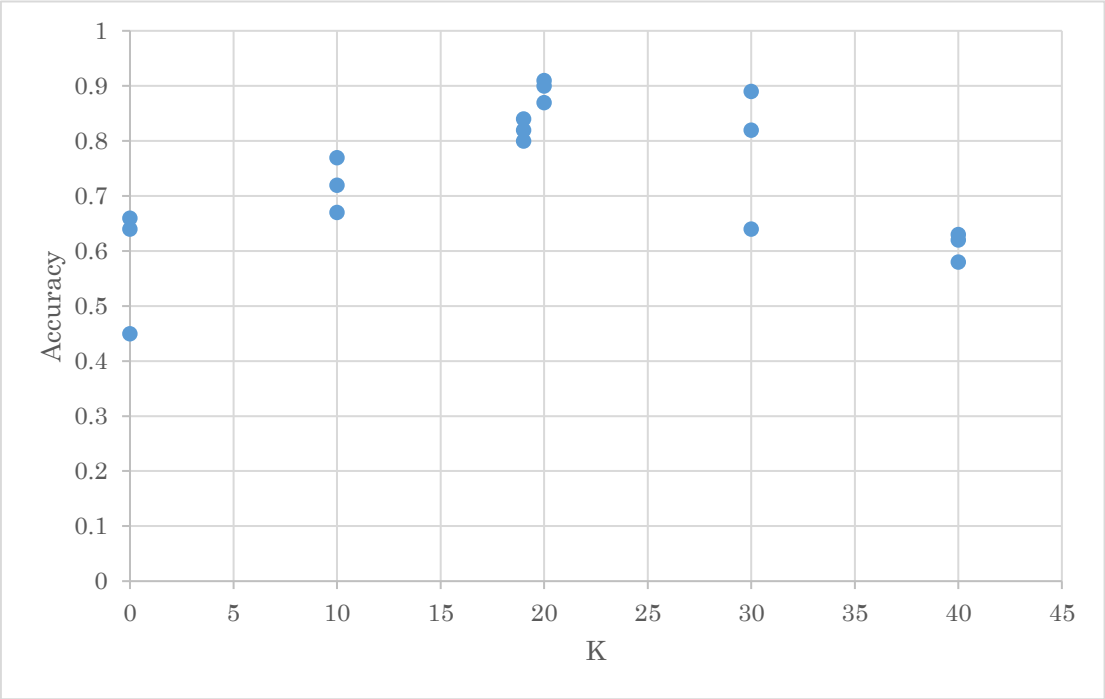

図 4.13 10 層における ***K*** を変化させたときの認識精度

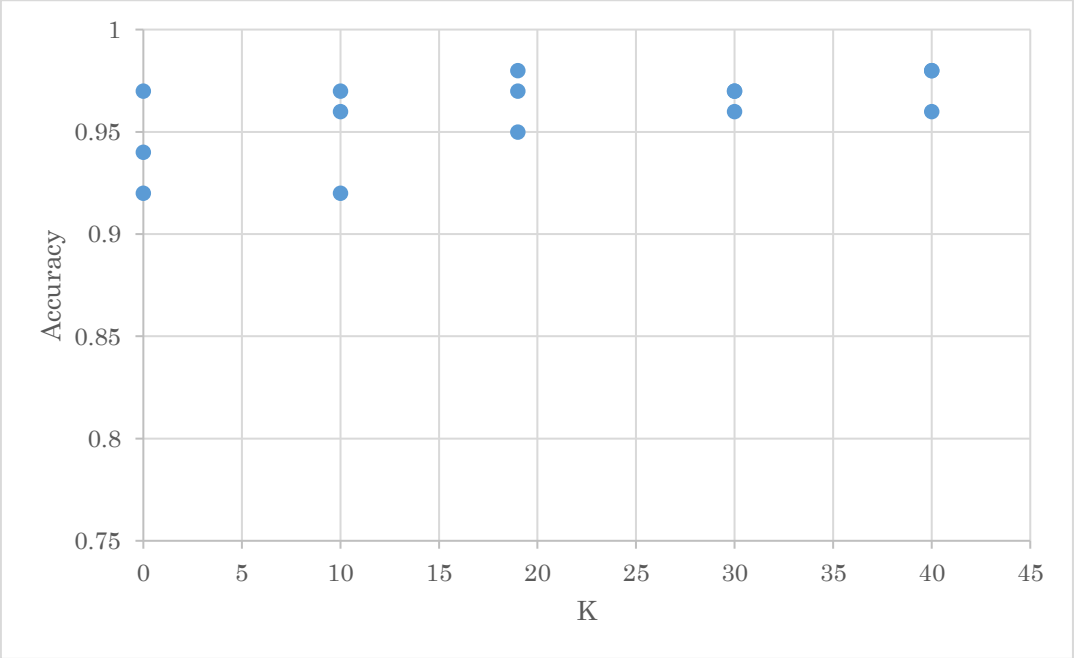

図 4.14 5 層における ***K*** を変化させたときの認識精度

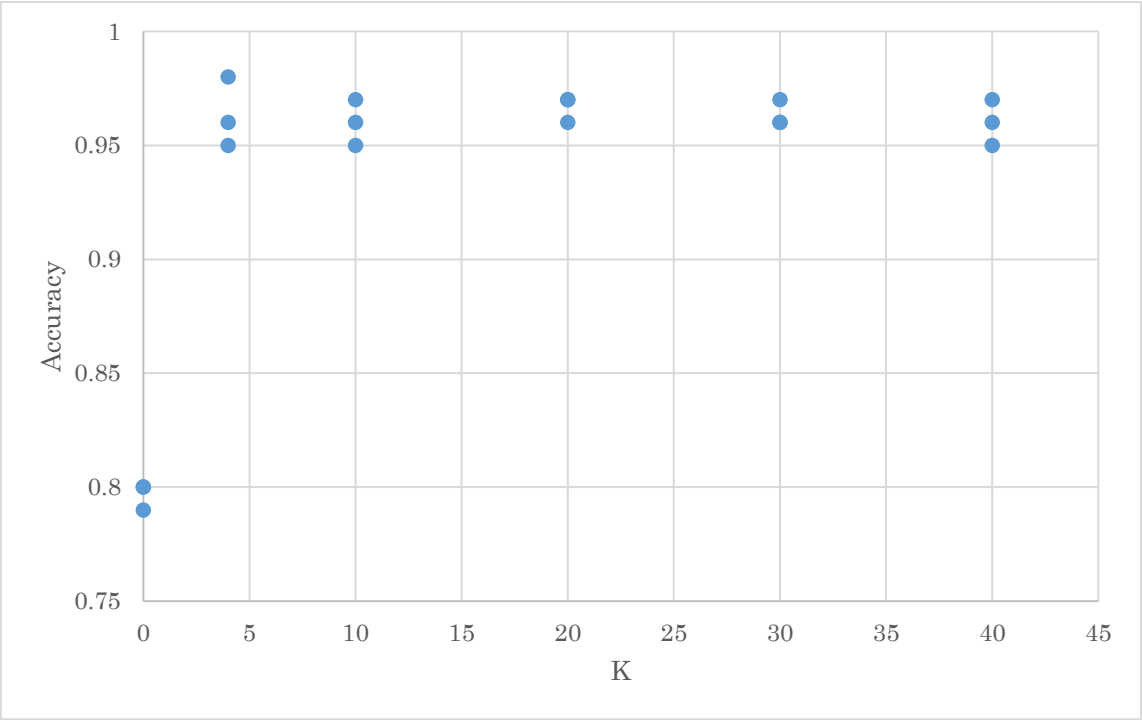

図 4.15 3 層における K を変化させたときの認識精度

また, それぞれのネットワーク規模における認識精度の値が最大になった K の値とその時の認識精度の最大値, 平均値を表 4.6 に示す.

表 4.6 各ネットワーク規模における認識精度の最高値を出した $\boldsymbol{K}$ とその平均値

ネットワーク規模	K の値	認識精度の最高値	認識精度の平均値
3	4	0.98	0.96
5	40	0.98	0.97
10	20	0.91	0.89

表 4.6 から各ネットワーク規模の認識精度の最大値については, 近似したにも関わらず表 4.4 と比較しても差が 1%に収まっていることが分かる.

4.3.3

考察

SNN における学習性能は実験 1 よりスパイク応答関数の形が大きく影響していることが分かっている. そこで, 実験 1 で一番学習性能が良かった時定数を用いたスパイク応答関数の概形に最も近似した式(3.2)における K を求め、その K の値を用いた近似と元の関数の学習性能の比較を行う. K の値は式(2.2)と式(3.2)の積分値の差を K を変更させながら, 最もその差が小さくなるものを求めた（表 4.7）. また, あくまで今回はオリジナルの式(2.2)に近い概形が取れればよかったため K は整数で求めた.

表 4.7 式(2.2)の時定数（τ_s）にもっとも近似した式(3.2)におけるそれぞれの $\boldsymbol{K}$ の値

τ_sの値	K の値
1	4
2	8
5	19

表 4.7 の結果から, 実験 2 における採用した時定数に応じた K の値のときの学習性

能と実験 1 での元の関数式(2.2)を用いたときの学習性能の比較は図 4.16, 17 のようになった. 図 4.16, 17 の横軸はネットワークの規模であり縦軸は図 4.16 では最小の損失関数の平均値, 図 4.17 では最も高い認識精度の平均値を示している.

図 4.16 から, oxford での実験では 3 層では損失関数の値の平均値の差が約 3%とほぼ同等であるのに対し, 5 層では約 18%, 10 層では約 33%の差で近似したスパイク応答関数を用いた方が損失関数の値の平均値が小さくなっていることが分かる.

図 4.17 から, NMNIST での実験では 3 層と 5 層では認識精度の平均値の差が 1%以内に収まっており近似を用いてもほぼ同等の学習性能であるといえる. 10 層においては元の関数を用いたときとの認識精度の平均値の差が 7%と他の層と比較すると大きくなっているがそれでも 80%以上の比較的高い認識精度を有していることが分かる.

次に, 実験 2 の中で最も学習性能が高かった時と実験 1 で最も学習性能が最も学習性能が高かった時の比較を図 4.18, 4.19 に示す. 図 4.18, 19 の横軸はネットワークの規模であり縦軸は図 4.18 では最小の損失関数の平均値, 図 4.19 では最も高い認識精度の平均値を示している.

図 4.18 から, oxford での実験において 3 層では損失関数の値の平均値の差が約 47%, 5 層では約 53%, 10 層では約 33%の差で近似したスパイク応答関数を用いた方が損失関数の値の平均値が小さくなっている. 特に 3 層では, 図 4.16 では近似を用いた損失関数の値が 3%元の関数を用いたときより高かったのに対し, ベストな K のパラメータでは近似を用いた方が元の式(2.2)を用いたときよりも半分近く損失関数の値を抑えることができている.

図 4.19 から, NMNIST での実験では 3 層では図 4.17 と同様に認識精度の平均値の差が 1%以内に収まっており近似を用いてもほぼ同等の学習性能であるといえる. また, 5, 10 層においては元の式(2.2)を用いたときと同じ認識精度の値を示している. 特に, 10 層においては図 4.17 では約 7%以上の差があったのに対し, 図 4.19 では同じ認識精度に達している.

以上のような結果から, oxford と NMNIST の種類の異なるどちらのデータセットにおいても, 5 層までの小規模なネットワークにおいては元の式(2.2)に近い概形の式(3.2)の近似を用いると式(2.2)を用いたときと同等かそれ以上の学習性能を有していることが分かった. また, 式(2.2)に近い概形の式(3.2)の近似がベストというわけではなく, さらに学習性能が向上する場合もあることから, 元の関数の形にこだわる必要がないこと

が示された.

これらのことから, スパイク応答関数を式(2.2)のような複雑な関数ではなく, 式(3.2)のような直線で構成された近似した関数を用いても学習性能は元の式(2.2)と同程度かそれ以上の学習性能を有しており, 近似したからといって多くの場合は学習性能が大きく損なわれるということはなかった.

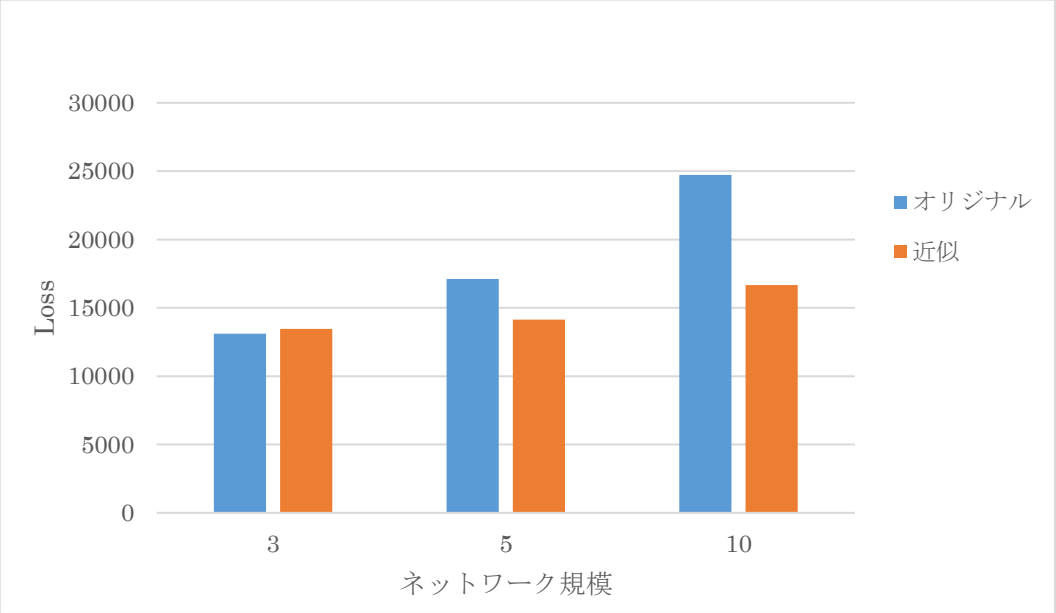

図 4.16 Oxford における式(2.2)の概形に最も近い式(3.2)の $\boldsymbol{K}$ の値を用いたときの損失関数の平均値とその比較

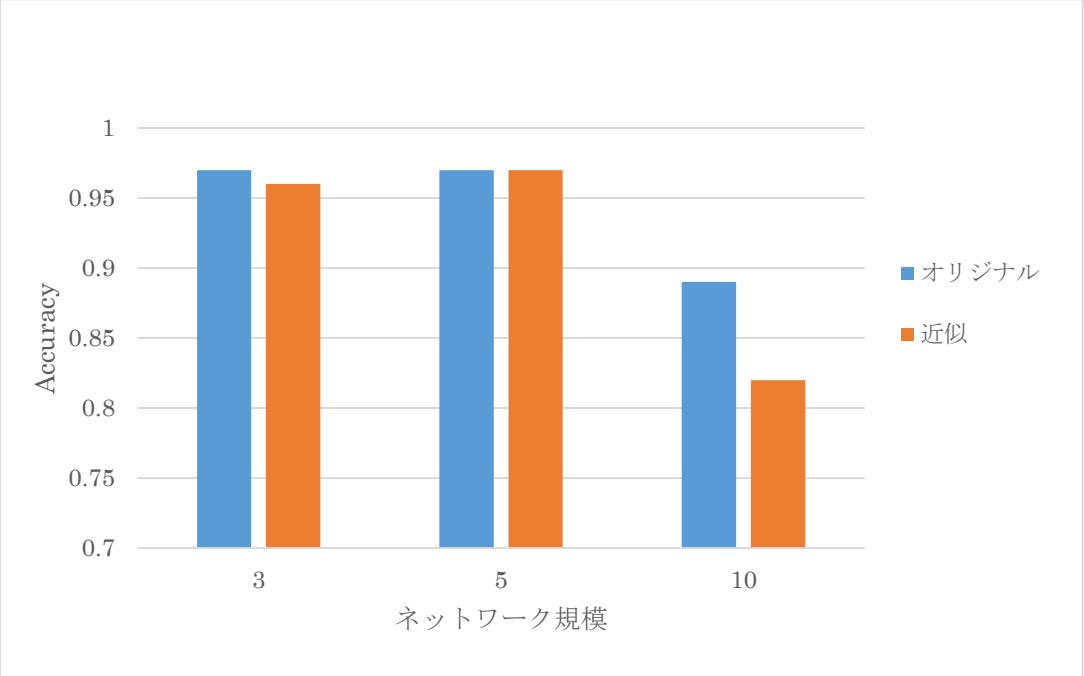

図 4.17 NMNIST における式(2.2)の概形に最も近い式(3.2)の $\boldsymbol{K}$ の値を用いたときの損失関数の平均値とその比較

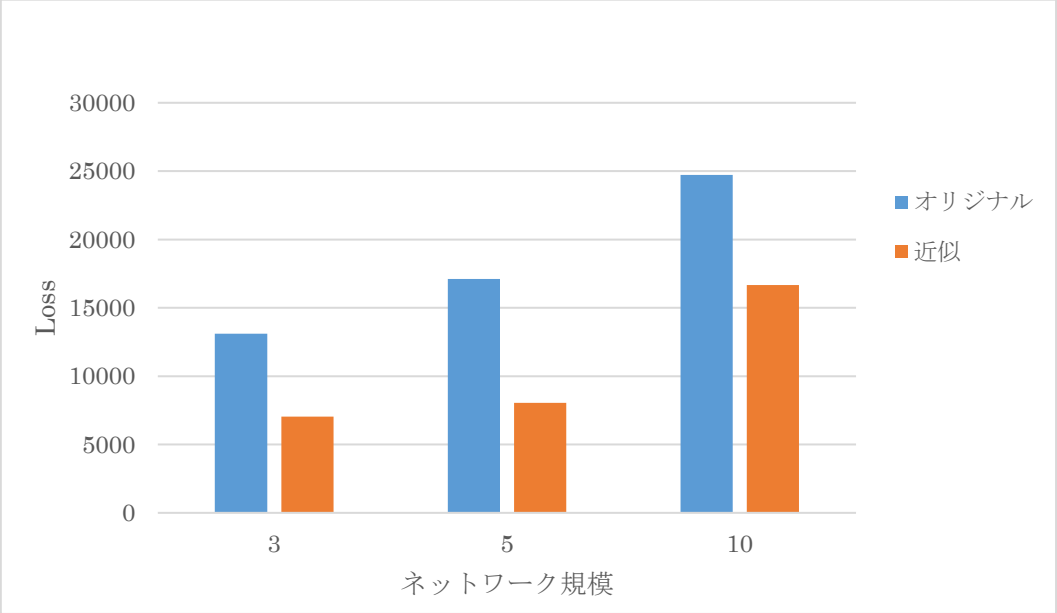

図 4.18 Oxford における式(2.2)と近似した式(3.2)を用いたときの損失関数が最も低かった値の平均値の比較

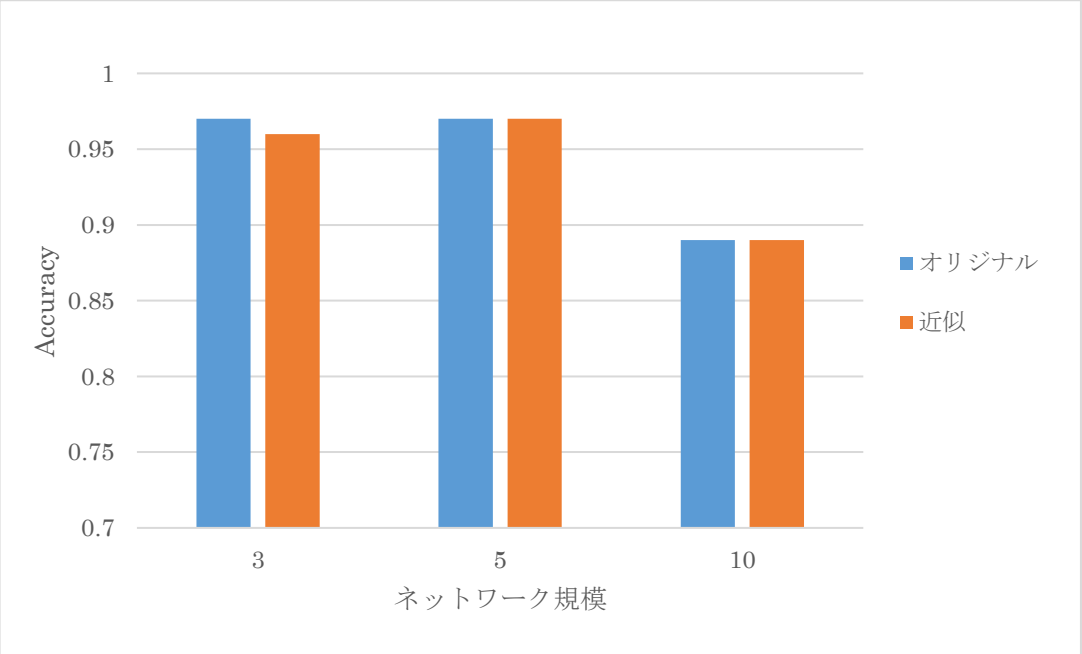

図 4.19 NMNIST における式(2.2)と近似した式(3.2)を用いたときの認識精度が最も高かった値の平均値の比較

4.4

まとめ

SNN はその歴史から生体により近いモデルにすることによって高い学習能力を発揮するため計算量が多かったり動作が複雑になったりしており実用化に至ることが中々できていない.

先行研究では SNN の計算量を多くしてしまっている要因の一つであるスパイク応答関数を単純な直線を用いた関数にすることにより計算量を削減し高速化が実現された. そこで本研究では, スパイク応答関数の近似をすることによって学習性能にどのような影響がでるのか, オリジナルの関数を用いたときと同程度の性能を発揮できるかを調査した.

実験の結果として, 種類の違う 2 つのデータセットを学習させたときにおいて, 近似を用いてもオリジナルの関数を用いたときと比較しても性能の劣化は見られず同程度かそれ以上の学習性能を発揮できることが分かった. これにより, 生体に近いニューロンモデルではなく, 単純な直線を合わせたスパイク応答関数でも学習性能を損なわないことが分かった. これは計算量の削減だけでなく, 実際にハードに実装する上でアドバンテージになりうる. また, スパイク応答関数の微分は SNN の学習法を考えるうえで長年大きな課題となっていた[25]. 直線を組み合わせによるスパイク応答関数は, その微分が簡単になり新たな学習法を生み出す可能性がある. 以上より, SNN のネットワークを構築する上でスパイク応答関数の近似は有効であるといえる.

今回は, 学習法として SLAYER を採用し実験を行ったが, 得られた結果は他の学習法の場合にも適用できると考える. 現在提案されている学習法は, 誤差逆伝搬法に基づくものが主流である. これらの方法は, 出力から求めた損失関数から, スパイク応答関数及びその導関数を基にパラメータの更新量を求め調整を繰り返すものである. そのため, スパイク応答関数の変更が及ぶ範囲は同程度であり, SLAYER 以外の学習法においても有効であると考えることができる.

第5章

まとめ

本研究では, SNN のネットワークの大規模化に貢献するため, 応答関数近似が SNN にもたらす影響を学習性能の観点から調査, 検討を行った.

SNN はその歴史から生体により近いモデルにすることによって高い学習能力を発揮すると考えられていた. そのため計算量が多かったり動作が複雑になったりしており大規模な実用化の事例は少ない. 特に, ネットワークの構成要素であるユニットのモデル化に関しては, これまで LIF ニューロンモデルや Izhikevich モデルなどの生体のモデルを数理的にモデル化したものが研究初期では用いられていた, それに対して, 生体の動きとは割り切って数学的計算量を優先したモデルもあった. しかし, これらも生体のモデル化を主眼とした研究の流れで用いられており, 大規模な情報処理システムの実現という観点からは, 単純化は十分に進んでいない. そこで, 本研究では既存のニューロンモデルを, 生体のモデル化という枠を越えて近似することを試みる. そのために, このような近似の SNN への影響を調査した. これは, 従来の NN でも ReLU のニューロンモデルを用いると計算量が減少し大規模化が実現できた例などから有効であるのではないかと考えたためであり, 具体的にはニューロンモデルの中の ReLU と同様にニューロンの内部状態を決定するスパイク応答関数についての近似を行った.

スパイク応答関数に近似を用いることにより SNN の動作の高速化が実現できることが確認されている[26]ので, 本研究においては学習性能の観点から近似が SNN にもたらす影響を調査, 検討を行った. また, 近似式についてはより学習性能への影響が懸念される直線の組み合わせによる近似を用いた.

実験は, 直線の組み合わせで近似した関数と元の関数のそれぞれをスパイク応答関数として SNN に組み込み, 3, 5, 10 層のネットワーク規模においてそれぞれ用意したデータセットを学習させることによって実施した. データセットについては, 確率的に分布した入力に対して複雑な建物の絵を出力で描く oxford というデータセットと手書き文字データを分類する NMNIST の種類が異なる二つのデータセットを用いて学習を行い, 評価を行った.

実験結果として, すべてのデータセットにおいてスパイク応答関数を直線の組み合わせで

近似したものを用いたとしても，ネットワーク規模に関わらず学習性能が大きく劣化することはなく同等以上の学習性能を有していることが分かった．

近似を用いることによる動作の高速化については先行研究で確認されている．この結果と，本研究の結果と合わせるとスパイク応答関数の近似は SNN の構築において，性能を落とさずに計算量を低減できるという意味で有効であるといえる．

また，従来の式(2.2)のような生体を数理的にモデル化したスパイク応答関数においてスパイク応答関数の微分は SNN の学習時に勾配消失などの問題を引き起こしていたが，式(3.2)のように線形な直線の組み合わせをスパイク応答関数に用いることによって微分が簡単になり，新たな学習法を生み出すことも可能であり，そういった点でもスパイク応答関数の近似は SNN において有利に働く可能性がある．

NN は，ネットワークの規模を大きくして学習をするディープラーニングが盛んに導入されている．これにより，学習精度の向上や複雑なデータであっても学習が可能になるが，学習コストの増大や，計算時間の増大などの問題が生じる．NN の一つである SNN においても学習性能や高速化においてさまざまな工夫がされることで，ディープラーニングの実現に向けて技術的な基盤が充実しつつある．しかし，従来の NN と比べより生体に近い処理を行う SNN においてはネットワークが大規模化するほどその複雑さは増大しており，まだまだ工夫の余地が残っている．このような状況の中で，本研究の結果からスパイク応答関数の近似をすることによって学習性能を維持しながら計算コストを削減すること示されたことは，SNN の大規模化におおいに貢献する．

本研究では，ベンチマークのデータセットを用いた実験しか実施していないため，今後は実際の問題に適応したデータセットに対しての効果を検討する．

謝辞

本研究を進めるにあたり，指導教授の高瀬治彦教授，計算機工学研究室の皆様方には，本論文の作成，研究発表において，助言，ご指導を頂きました．感謝いたします．最後に，本研究，本論文に関わっていただいたすべての方々に感謝の意を表します．

参考文献

[1] Igor Kononenko, "Machine learning for medical diagnosis: history, state of the art and perspective", Artificial Intelligence in Medicine, Vol. 23, pp. 89–109, 2001.

[2] Konstantinos G. Liakos, Patrizia Busato, Dimitrios Moshou, Simon Pearson, Dionysis Bochtis, "Machine Learning in Agriculture: A Review", Sensors, Vol. 18, No. 8, 2018.

[3] Vinicius Andrade Brei, "Machine Learning in Marketing: Overview, Learning Strategies, Applications, and Future Developments", Foundations and Trends® in Marketing, Vol. 14, pp. 173–236, 2020.

[4] Maxim Mozgovoy, Marina Purgina, Iskander Umarov, "Believable self-learning AI for world of tennis", IEEE Conference on Computational Intelligence and Games, 2016

[5] Tyna Eloundou, Sam Manning, Pamela Mishkin, Daniel Rock, "GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models", arXiv:2303.10130, 2023

[6] W. S. McCulloch, and W. Pitts. "A logical calculus of the ideas immanent in nervous activity", The bulletin of mathematical biophysics, Vol. 5, No. 4, pp. 115–133, 1943.

[7] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", Psychological Review, Vol. 65, No. 6, pp. 386–408, 1958.

[8] M. Minsky, and S. Papert, "Perceptrons", MIT Press, 1969.

[9] D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning representations by back propagating errors", Nature, Vol. 323, pp. 533–536, 1986.

[10] Jürgen Schmidhuber, "Deep learning in neural networks: An overview", Neural Networks, Vol. 61, pp. 85-117, 2015.

[11] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets", Neural computation, Vol. 18, No. 7, pp. 1527–1554, 2006.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", Advances in Neural Information Processing Systems 25, pp. 1097–1105, 2012.

[13] A. L. Hodgkin, and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve", The Journal of physiology, Vol. 117, No. 4, pp. 500–544, 1952.

[14] E. M. Izhikevich, "Simple model of spiking neurons", IEEE Transactions on Neural Networks, Vol. 14, No. 6, pp. 1569–1572, 2003.

[15] W. Gerstner, "Time structure of the activity in neural network models", Physical Review E, Vol. 51, No. 1, pp. 738–758, 1995.

[16] W. Maass, and C. M. Bishop, "Pulsed Neural Network", The MIT Press, 1998.

[17] F. Ponulak, and A. Kasi n ski, "Introduction to spiking neural networks: Information processing, learning and applications", Acta neurobiologiae experimentalis, Vol. 71, No. 4, pp. 409–433, 2010.

[18] W. Maass, "Noisy Spiking Neurons with Temporal Coding have more Computational Power than Sigmoidal Neurons", M. Mozer, M.I. Jordan, T. Petsche (Eds.), Advances in Neural Information Processing Systems, MIT Press, Vol. 9, pp. 211–217, 1997.

[19] Michael Pfeiffer and Thomas Pfeil, "Deep Learning With Spiking Neurons: Opportunities and Challenges", Frontiers in Neuroscience, Vol. 12, Art. 774, 2018.

[20] 銅谷 賢治, ほか編, “脳の情報表現 ―ニューロン・ネットワーク・数理モデル―”, 朝倉 書店, 2002.

[21] S. M. Bohte, “The evidence for neural information processing with precise spike-times: A survey”, Neural Computing, Vol. 3, No. 2, pp. 195–206, 2004.

[22] M. C. W. van Rossum, G. Q. Bi, and G. G. Turrigiano, “Stable Hebbian Learning from Spike Timing-Dependent Plasticity”, Journal of Neuroscience, Vol. 20, pp. 8812-8821, 2000.

[23] O. Booij, H. tat Nguyen, “A gradient descent rule for spiking neurons emitting multiple spikes”, Information Processing Letters, Vol. 95, pp. 552–558, 2005.

[24] Filip Ponulak and Andrzej Kasñski, “Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting”, Neural Computation, Vol. 22, no.2, pp.467-510, 2010.

[25] Sumit Bam Shrestha, Garrick Orchard, “SLAYER: Spike Layer Error Reassignment in Time”, Advances in Neural Information Processing Systems, pp. 1417–1426. 2018.

[26] 尾崎 舜, “スパイキングニューラルネットワークの応答関数近似による高速化”, 三重大学大学院工学研究科修士論文, 2022

[27] X. Glorot, A. Bordes and Y. Bengio: “Deep sparse rectifier neural networks”, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Vol. 15, pp. 315–323, 2011

研究実績

[1] 安井雅哉，他，スパイキングニューラルネットワークの応答関数近似による学習精度の変化に関する一考察，第 53 回東海ファジィ研究会，P1-10，2023

[2] 安井雅哉，他，スパイキングニューラルネットワークの応答関数近似による学習性能への影響，第 39 回ファジィシステムシンポジウム，pp.137–140，2023

[3] Masaya Yasui, et.al., Influence of Response Function Approximation on Learning Performance of Multilayer Spiking Neural Networks, The 15th International Workshop on Regional Innovation Studies, pp.63–66, 2023