卒業論文 2020 年度 (令和 02 年)

カーネル密度推定を用いた汎用的な活性化関数

村井・楠本・中村・高汐・バンミーター・植原・三次・中澤・武田 合同研究プロジェクト

慶應義塾大学 環境情報学部
神保和行

卒業論文要旨 - 2020 年度 (令和 02 年度)

カーネル密度推定を用いた汎用的な活性化関数

ニューラルネットワークにおけるニューロンの出力を決定する活性化関数は、モデルの表現できる関数の幅を広げるための重要な数学的要素の一つである。その活性関数には Sigmoid や ReLU といった関数が一般的に用いられ、機械学習の性能を飛躍的に向上させてきた。より良い活性化関数を調査する研究は現在も行われており、Swish(2017) や Mish(2019) といった新しい手法も精力的に見つけられている。しかしながら、より精度が高いモデルを作るために、活性化関数に何を用いるかという問題は経験的に判断されているだけで未解決のままである。本論文では統計の世界におけるセミパラメトリックモデルを応用した手法を用いて、モデルに応じた適切な活性化関数を推論しながら学習する手法を導いた。それにより、扱っている問題を意識することなく高い精度のモデルを制作することができるようになった。また、いくつかのパターンで既存の活性化関数よりも高い精度を導くことに成功した。また、そのような関数が関数全体から探査できるようになり、今まで用いてきた活性化関数がそのモデルに対して十分な表現を持っているかどうか判断できるようになった。このような成果により、分類や回帰といった学習データの形状を意識せずとも高い完成度でモデルの作成から学習まで行えることを示した。本研究が、ニューラルネットワークの汎用度の向上に繋がり、機械学習がより幅広く人々と関わっていく社会への貢献へとつながることを望む。

キーワード:
1. ディープラーニング, 2. 活性化関数, 3. ノンパラメトリック, 4. カーネル密度推定

慶應義塾大学 環境情報学部
神保和行

Abstract of Bachelor's Thesis - Academic Year 2020

Generic Activation Functions Using Kernel Functions

The activation function in neural networks is one of the important mathematical elements to expand the range of functions that neural networks can represent. Functions such as Sigmoid and ReLU are commonly used as its activation function, and have dramatically improved the performance of machine learning. Research to investigate better activation functions is still ongoing, and new methods such as Swish (2017) and Mish (2019) are being found vigorously. However, the question of what to use for the activation function in order to create a more accurate model remains unsolved, as it has only been determined empirically. In this paper, we applied the semiparametric model in the world of statistics to derive a method for learning while inferring the appropriate activation function for the model. By doing that, we were able to produce highly accurate models without being aware of the problem we were dealing with. We also succeeded in deriving higher accuracy than the existing activation functions in some patterns. In addition, such functions can now be explored from the entire function, and we can determine whether the activation functions we have been using have sufficient representation for the model. These results show that it is possible to create and train models with a high degree of completeness without being aware of the shape of the training data, such as classification and regression. We hope that this research will lead to the improvement of the versatility of neural networks, and contribute to a society where machine learning can be used more widely with people.

Keywords :
1. Deep lerning, 2. Activation Function, 3. Non parametric, 4. Kernel Function

Keio University Faculty of Environment and Information Studies
Kazuyuki Jimbo

目 次

図 目 次

表 目 次

記号

- 1 次元の値は通常の字体 x を用いる。ベクトルや行列など、複数の値を内部に持っていることを強調したい場合は $\mathbf{x}$ や $\mathbf{X}$ などの太字を用いる。
- $\mathbb{R}$ は実数の集合を表現し $\mathbb{R}^d$ は d 次元の実数のベクトル空間を表現する。
- $\mathrm{E}[x]$ は x の期待値を表す。
- e は自然対数の底で、指数関数は $\exp(x) = e^x$ のように表す
- $\mathcal{G}$ はガウス分布を表現する。
- 分布 $p(x)$ からサンプルを得ることを $x \sim p(x)$ と表記する。
- データセットの集合を $\mathcal{D}$ で表現しその要素を d_i と表現する。
- データセットの集合 $\mathcal{D}$ から n 個のデータをランダムにサンプリングすることを $X \sim_n \mathcal{D}(X)$ と表現する。

第1章　序論

本章ではまず本研究を取り巻く背景について述べる。そして本研究の解決する課題及び課題を解決する意義、解決するための手法を提示する。次に本論文がもたらした貢献と、解決した課題を述べ、最後に本論文の構成を概観する。

1.1　はじめに

背景

近年、画像認識や音声認識といった分野で機械学習の中でもディープラーニングと呼ばれる分野が急速発展し、自動運転やスマートスピーカーといったプロダクトとして人々の日常の中にも応用され始めている。機械学習はプログラミングを容易にする Pytorch [1] や Tensorflow [2] と呼ばれるライブラリの開発も積極的に行われているだけではなく株式会社ソニーの Neural Network Console [3] を筆頭とした、非エンジニアにも使いやすい GUI ベースのツールも多数開発され続けている。これらは機械学習におけるニューラルネットワークの構築を容易にするだけではなく、学習アルゴリズム自体も抽象化され複雑な式を理解せずとも使用できるようになっている。また、ニューラルネットワークを構築する重要な要素の一つに活性化関数がある。この活性化関数には、Sigmoid や ReLU [4] などの一般的に用いられてきた。そしてこれらは問題やデータに応じて最適な活性化関数を経験則に基づいて調整していた。また、ニューラルネットワークでは特に出力層のアウトプットを考慮した活性化関数の組み合わせを採択することが多く、例えば Resnet50 [5] では、問題が分類ということを考慮され、中間層は全て ReLU、出力層は Sigmoid などといった組み合わせが使用されている。

一方、統計学の分野では、リンク関数が未知の場合には、ノンパラメトリックな手法に基づきモデルを推論する幾つかの手法が考案されている。カーネル密度推定を用いてノンパラメトリックに推定するという手法が Ichimura(1993) [6] によって提案されている。

課題・手法

ニューラルネットワーク構築の際の課題として、問題に応じた最適な活性化関数が未だわかっていないことが挙げられる。全ての課題に同一の活性化関数を流用するのではなく、各課題においてその都度適切な活性化関数を導くことができればより良い精度が出せることが予想される。特に出力層に用いる活性化関数はデータセットや問題の種類を意識する必要があり、精度に非常に直結するだけでなく、初学者にとっての構築を難しくす

る。そこで本論文は事前に関数の形を指定しないカーネル密度推定を用いた活性化関数を提案する。本論文ではカーネル密度推定とトレーニングデータの少数の点を用いて活性化関数の推論をするアルゴリズムを構築する。提唱された Ichimura(1993) の方法では、トレーニングデータの全てを活性化関数の計算に使用する必要があったが、ディープラーニング等のデータ量が非常に大きい場合での応用を考え計算時間を現実的なものにするため、使用するデータ点を可変にすることが可能なアルゴリズムを提案した。

貢献

以降本研究で提案するカーネル密度推定を用いた活性化関数を K-AF と表記する。K-AF は実際のデータセットを用いて、ニューラルネットワークの出力層を本論文で提案する方法に置き換えることにより従来の活性化関数と同等かそれ以上の精度で予測できること示した。本研究における主な貢献を以下にまとめる。

- カーネル密度推定を用いた汎用的な活性化関数で実用的なものを完成させた。
- K-AF はさまざまなデータセットにおいて既存の活性化関数によりより良い精度を出すことを達成した。
- K-AF はいくつかのデータセットでは高い学習率でも安定した学習精度を出すことに成功した。
- K-AF を用いることによりデータセットに応じて出力層の活性化関数の形は従来のものではないことを示した。
- K-AF が勾配爆発 (2.3.6 節 参照) しないための条件を探求した。

以上の貢献により以下の一般的な課題を解決した。

- 出力層に用いる活性化関数を状況に応じた適切な形に変わる汎用的な関数を導として、高い精度を出せるようにした。
- 各問題やデータに応じた最適な活性化関数の形が模索できるようになった。
- そのような関数を用いることで、データセットの形等の専門的な知識を理解しなくて済むようになり、ニューラルネットの構築を容易にした。

1.2 本論文の構成

本論文における以降の構成は次の通りである.

2 章では、ノンパラメトリックモデル、カーネル密度推定などといった本研究へとつながる背景の解説し、これらの手法における課題を洗い出す。 3 章では、本研究におけるカーネル密度推定を用いた活性化関数についての解説を行い、提案手法の解説の詳細を述

べる。 4 章では、 3 章で述べた手法の実装及び、実装における留意点を述べる。 5 章では、2 章で求められた課題に対しての評価を行い、考察する。 6 章では、実験の結果に対する考察を行い, 本研究を行う上で浮上した提案手法の限界を示し, 今後の研究方針についてまとめる。

第2章　背景

本章では本研究の背景について述べる。まず機械学習における活性化関数の役割について明確にする。また、統計学において活性化関数に相当する概念がどのように応用されてきたか述べる。次に、活性化関数の他に、ニューラルネットワークにおける精度を向上させるいくつかの構成要素について述べる。本研究の問題点の解決に必要な、セミパラメトリックモデルとその具体例であるカーネル密度推定を導入する。最後に、本研究に必要な機械学習を行う上での社会的観点も踏まえた問題点を述べる。

2.1　活性化関数

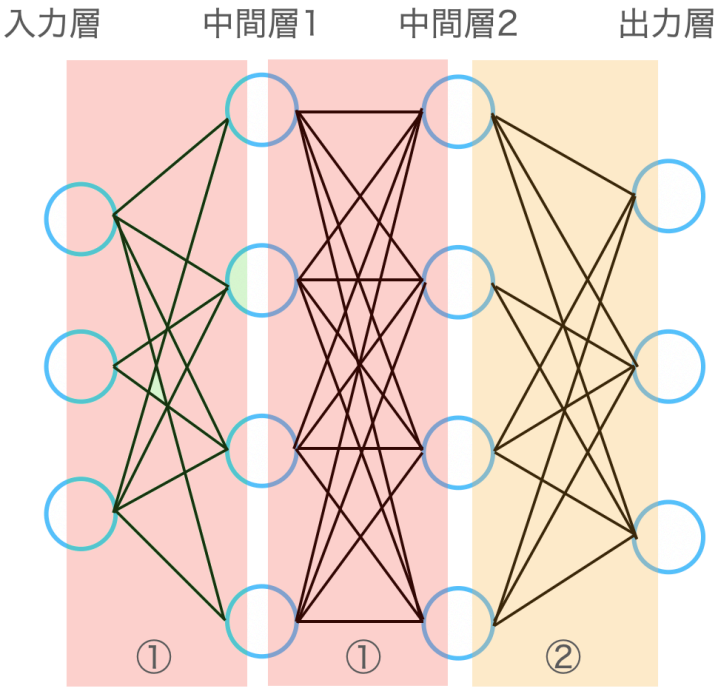

図 2.1: 活性化関数の領域。①、②の領域によって使われる活性化関数が変わることが多い。

また、概要を述べるにあたってニューラルネットワークの用語を定義する。活性化関数

は図 2.1 の①の部分で使用する活性化関数を「中間層の活性化関数」、②の部分で使用する活性化関数を「中間層の活性化関数」と記述することとする。

2.1.1 活性化関数とは

活性化関数とは端的に表現すると、ニューラルネットワークの表現できる関数の空間を広げるための数学的な関数である。ニューラルネットワークは図 2.1 のように多数の層に別れていて、その層同士の結合部分に活性化関数を置くことで、ニューラルネットワーク自体の精度を向上させることができる。

2.1.2 さまざまな活性化関数とその歴史

表 2.1: 活性化関数の種類

活性化関数の式	式
Sigmoid	$\dfrac{1}{1+\exp(x)}$
Tanh	tanh(x)
ReLU	$\text{output} = \begin{cases} 0 & \text{when } x < 0 \\ x & \text{when } x \geq 0 \text{ else} \end{cases}$
Swish	$x \cdot \text{sigmoid}(\beta x)$
Mish	$x \cdot \tanh(\log(1+\exp(x)))$

活性化関数に関する研究は近年も精力的に行われており、実務的にも有用な多くの関数が見つけれている。 [7]。

その始まりは、Sigmoid という統計学的にも馴染み深いロジスティック回帰の参考にしたモデルから始まった。

その後 Tanh や Xavier の ReLU(2011) [4] などといったより計算に適した活性化関数が発見されてきた。特に ReLU に関しては現在のディープラーニングなどの深層ニューラルネットワークにおいても未だ応用されており、実用的にもその有用性が示されている。ReLU には勾配消失が発生するという欠点があるが、解決するモデルとして、Leaky ReLU [8]、ELU [9]、SELU [10] なども考案されている。また、それまで以上に精度が良いモデルを探すために強化学習を用いた手法で、Prajit Ramachandran の Swish(2017) [11] や Diganta. Misra の Mish(2019) [12] という活性化関数も発見された。Swish や Mish は ReLU や Sigmoid の前提にあった単調増加性という性質の前提を覆し、大きく性能を向上

させるきっかけとなった。これらの活性化関数の式を表 2.1 に記す。時代が進むにつれて式より高度で複雑化していることが確認できる。

2.2 統計学における位置付け

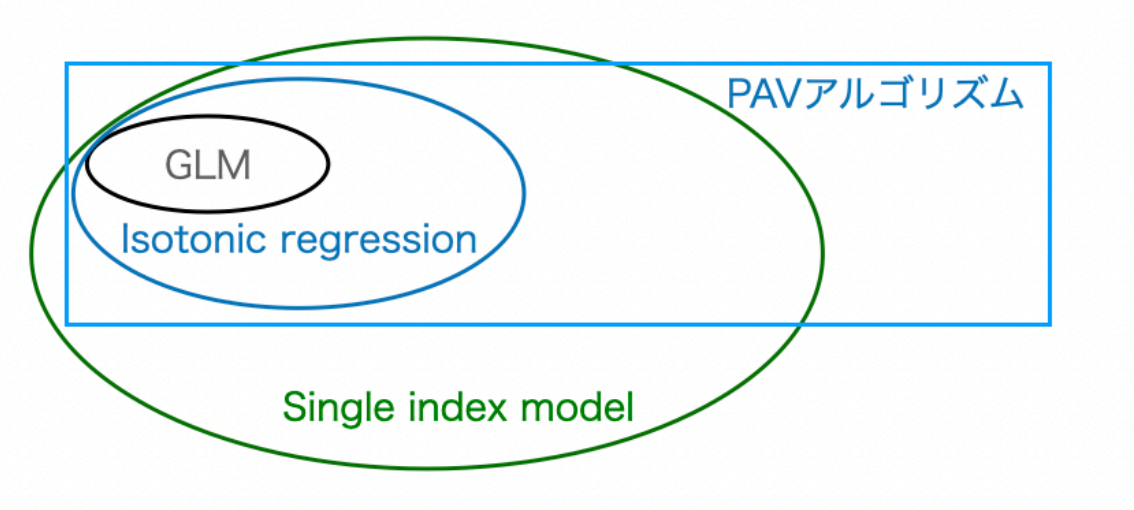

図 2.2: 機械学習と統計学の繋がり

活性化関数の概念は統計学におけるリンク関数と呼ばれるモデルの汎用性を高める動きに始まり、機械学習へと応用されている。本項目ではその理解に必要な知識を述べていく。

2.2.1 一般化線形モデルとは

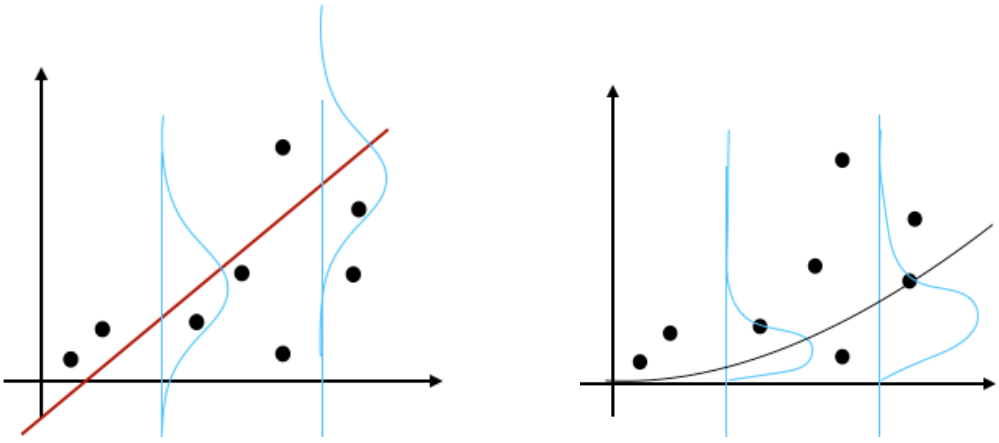

図 2.3: 一般化線形モデルの必要性

説明変数のベクトルを $\mathbf{X}$, パラメータのベクトルを $\mathbf{W}$ で表現し、従属変数を Y、誤差を $\epsilon \sim \mathcal{G}(0, \sigma^2)$ で表現すると、一般線形モデルは以下の式で表現することができる。

$$\mathrm{Y} = \mathbf{X} \cdot \mathbf{W} + \epsilon \tag{2.1}$$

またこれは Y の期待値を使って表現すると上記は

$$\begin{aligned} \mathrm{E}[\mathrm{Y}] &= \mathrm{E}[\mathbf{X} \cdot \mathbf{W} + \epsilon] && (2.2)\\ \mathrm{E}[\mathrm{Y}] &= \mathrm{E}[\mathbf{X} \cdot \mathbf{W}] + \mathrm{E}[\epsilon] && (2.3)\\ \mathrm{E}[\mathrm{Y}] &= \mathbf{X} \cdot \mathbf{W} && (2.4) \end{aligned}$$

である。しかしながら、上記の式の展開では実際は図 2.3 左のように ϵ が正規分布に従うことを想定した、すなわち従属変数 Y がガウス分布に従うことを仮定したが、実際は図 2.3 右のように、誤差の分布にガウス分布を仮定すると、正確さが失われることがある。そこで、従属変数をある関数 G で変換してからモデル化することでモデルの正確さが向上する。すなわち、$\mathrm{E}[\mathrm{Y}|\mathbf{X}] = \mathbf{X} \cdot \mathbf{W}$ に対して $G(\mathrm{E}[Y|\mathbf{X}]) = \mathbf{X} \cdot \mathbf{W}$ となるような G を取り入れる。またこの G の逆関数 G^{-1} をリンク関数と呼ぶ。一般線形モデルに対して、リンク関数を加えた式を以下に記す。

$$\mathrm{E}[\mathrm{Y}|\mathbf{X}] = G^{-1}(\mathbf{X} \cdot \mathbf{W}) \tag{2.5}$$

一般に誤差構造が決まれば、リンク関数も自動的に決まる。ガウス分布の場合のリンク関数は $G(U) = U$ である。これらの結果は G^{-1} を単調増加な任意の関数に置き換えることでさまざまなモデルを表現することが可能になる。

2.2.2 Sigmoid 関数とロジスティック回帰

$$G(\mathrm{Y}) = \mathbf{X} \cdot \mathbf{W} \tag{2.6}$$

とした時、

$$G = \log\Big(\frac{\mathrm{Y}}{1-\mathrm{Y}}\Big) \tag{2.7}$$

とする。一般的にこれはロジット関数と呼ばれる。これを左辺が Y になるように変形すると,

$$\log\Big(\frac{\mathrm{Y}}{1-\mathrm{Y}}\Big) = z \tag{2.8}$$

$$\mathrm{Y} = \frac{1}{1+\exp(-z)} \tag{2.9}$$

右辺を z を関数にすると

$$g(z) = \frac{1}{1 + \exp(-z)} \tag{2.10}$$

となり、これは Sigmoid 関数である。$g(z)$ はロジスティック関数でもあり、

$$\mathrm{Y} = \frac{1}{1 + \exp(\mathbf{X} \cdot \mathbf{W})} \tag{2.11}$$

より、ロジスティック回帰であることも示される。

これらにより、ニューラルネットワークに出てくる Sigmoid 関数が潜在的に統計学の世界でも出てくることが確認できる。

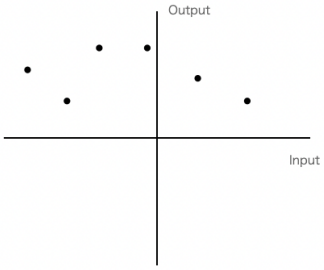

図 2.4: カーネル密度推定に用いるデータ点の集合

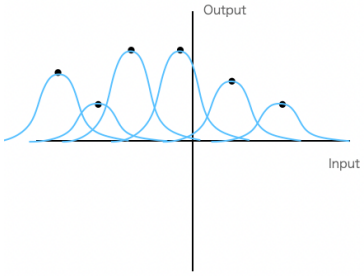

図 2.5: カーネル関数、今回はガウス関数でその周辺ごと近似する。

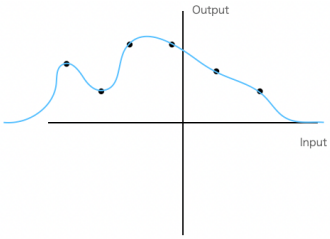

図 2.6: 点の周辺のカーネル関数を足し合わせた時にできる関数

2.2.3 ノンパラメトリックとカーネル密度推定

統計学において、パラメータで表現されるモデルや確率分布を使用するものをパラメトリックな手法として分類するが、パラメータを使用せずモデルを表現する手法をノンパラ

メトリック手法という。ノンパラメトリックを代表する手法の一つにカーネル密度推定と呼ばれる手法がある [13]。これは、ある母集団のデータが与えられたとき、カーネル関数を用いてその関数を推定する手法である。カーネル関数とは、与えられた領域内で積分した時に 1 となり、対称性を持つものとしてイメージして良い。カーネル関数の代表例としてガウス関数があげられる。

K をカーネル関数 $u \in \mathbb{R}$ とした時、カーネル関数の定義は以下である。

- $\int_{-\infty}^{+\infty} K(u)du = 1$
- $K(-u) = K(u)$

この時、カーネル密度推定法とは、x_n をデータ、推定すべき関数を f、カーネル関数を K、バンド幅を h としたとき、以下の式で表現することができる。

$$f(x) = \frac{1}{nh}\sum_{i=1}^{n} K\big(\frac{x - x_i}{h}\big) \tag{2.12}$$

図 2.4 のようにまばらに存在するデータ点の周辺に、図 2.5 のようにカーネル関数をおき、任意のバンド幅で足し合わせ近似していくようなものである。

2.2.4 セミパラメトリックモデルと SingleIndexModel

統計学の世界では、セミパラメトリックモデルというノンパラメトリックな手法とセミパラメトリックな手法を組み合わせた手法が存在する。その中の一つの代表的な手法の中に SingleIndexModel(SIM) と呼ばれる手法が存在する。SIM とは、未知の関数 g、従属変数を Y、説明変数のベクトルを $\mathbf{X}$、パラメータのベクトルを $\mathbf{W}$、誤差項 ϵ と置いた時、以下のように表される式である。

$$Y = g(\mathbf{X} \cdot \mathbf{W}) + \epsilon \tag{2.13}$$

SIM は未知の関数 g を推定しながらパラメータのベクトル $\mathbf{W}$ を求めていく問題に帰着されるため、ノンパラメトリックとパラメトリックが混ざった手法であるセミパラメトリックモデルとして表現される理由である。この g は、一般化線形モデルのリンク関数 G^{-1} をさらに一般化した単調増加性を無くしたモデルである。SIM の有名なモデルの一つに isotonic regression と呼ばれるものがある。

isotonic regression は単調増加性等の制約を仮定した SIM の一つで、化学分野や経済分野に応用されている。

2.2.5 セミパラメトリックモデルと機械学習

セミパラメトリックにモデルを推定する手法は統計学では Ichimura(1993) [6] から始まり、PAV アルゴリズムとして Adam Tauman Kalai(2008) [14] によりその有用性が確かめられた。

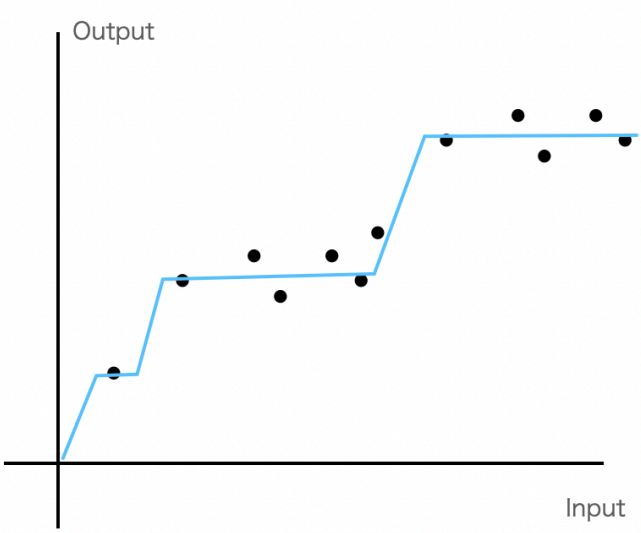

図 2.7: SingleIndexModel の例の一つの isotonic regression

Ichimura の手法

SIM の未知の関数を leave one out 法を用いたカーネル密度推定法と最尤推定により導く試みは Ichimura(1993) [6] や Klein(1993) [15] によって提案され始めた。

Ichimura(1993) の手法は SingleIndexModel のノンパラメトリック関数を以下の式で近似する手法である。

$$G(\mathbf{X}_i\mathbf{W}) = \frac{\sum_{i\neq j} K\left(\frac{\mathbf{X}_j\mathbf{W}-\mathbf{X}_i\mathbf{W}}{h}\right)\mathbf{Y}_j}{\sum_{i\neq j} K\left(\frac{\mathbf{X}_j\mathbf{W}-\mathbf{X}_i\mathbf{W}}{h}\right)} \tag{2.14}$$

ここで、K はカーネル関数である。$i \neq j$ とすることにより、$\mathbf{X}_i$ を入力した時の値が $\mathbf{Y}_i$ へと過剰適合しないようにするためである。

PAV アルゴリズム

SIM や isotonic regression を機械学習に応用する試みは Adam Tauman Kalai [14] の PAV アルゴリズムと呼ばれる手法で、分類問題の応用へと繋がった。isotonic regression 自体は回帰問題として発明された手法であったが、これによりアルゴリズム的に分類問題がセミパラメトリックな手法を用いて解くことが可能であることが発見された。この手法をベースに Sham Kakade(2011) [16] や Ravi Ganti(2015) [17] などによってより高速で汎用的な isotonic regression を応用したセミパラメトリックモデルの分類問題の解法のアルゴリズムが導かれた。

2.3 勾配法と学習における知識

機械学習の問題の多くは学習の際の最適なパラメータを探索する。最適なパラメータとは損失関数が最小値を撮る時の値のことである。勾配法とは関数の勾配方向に閾値を移動させることで、関数の最小値を見つける方法のことである。特にニューラルネットにおいては最小値を見つけるために、勾配法がよく用いられる。損失関数を E, $\mathbf{W_i}$ を i ステップ目のパラメータとした時、勾配法を式で表すと以下のようになる。

$$\mathbf{W_{i+1}} = \mathbf{W_i} - \mu \frac{\partial \mathrm{E}}{\partial \mathbf{W}} \tag{2.15}$$

複雑な損失関数を最小化させるためのテクニカルな手法として、学習率、初期パラメータ、正則化などと言ったものが挙げられる。本項ではこれらについて必要な概念を述べる。

2.3.1 LearningRate（学習率）

LearningRate とはハイパーパラメータの一つで、式 2.15 の μ に相当する部分である。LearningRate は大きいほど収束の可能性は小さくなり、小さいほど学習が遅くなる。また、高い LearningRate でも安定して学習できる活性化関数が求められている。

2.3.2 Initializer(重みの初期化)

ニューラルネットワークの学習効率は、重みの初期値によって大きく変わることが知られている。例えば初期値を全て 0 に固定すると、逆誤差伝播の影響で重みが均一になることにより、重みを多く持つ意味がなくなる。この問題を解消するために、学習のテクニックとして以下のような手法がある。また、重みの初期化手法を総称して Initializer と記述する。

Xavier initializer

Xavier(2010) [18] によって提唱された重みの初期化手法の一つで、現在最もニューラルネットワークの学習で利用されてる手法である。ニューラルネットワークの i 層の重みの数を n_i、重みを w_i、一様分布を U で表現した時

$$w_i \sim U\big[-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}\big] \tag{2.16}$$

で初期化する手法である。これにより初期化すると、重みの分布が広がりを持つことが知られている。活性化関数が Linear、Sigmoid、Tanh の時に有効に働く。

KaimingUniform

Kaiming He(2015) [19] によって提案された初期化の手法で、以下の分布から重みをサンプリングする。

$$w_i \sim U\big[-\frac{\sqrt{6}}{\sqrt{n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_{i+1}}}\big] \tag{2.17}$$

主に ReLU に有効になる。

2.3.3 Regularizer（正則化）

ニューラルネットワークは訓練データを過剰に学習すると未知データへの予測精度が落ちることがある。これはモデルが訓練データに対して過剰に学習したために、はずれ値やノイズまで学習してしまうことに起因する問題である。このような現象を過学習というが、この過学習を防ぐためにパラメータに対して一種の罰則をかけるようなことが一般的に行われている。

損失関数を $\mathrm{E}(w)$ とした時に、最適化する関数を $\mathrm{E}(w)$ の代わりに以下の式を用いる。

$$\mathrm{E}(w) + \mu\frac{1}{p}\|w\|_p \tag{2.18}$$

ここで w はパラメータベクトルで　$\|\cdot\|_p$ は L1 ノルム (p=1) や L2 ノルム (p=2) などである。μ はハイパーパラメータである。

L1 ノルム

L1 正則化は余分なパラメータ（説明変数）を省くことを目的とした手法である。2.18 の式を考察すると、モデルに必要ないパラメータを 0 にすることが損失関数の最小化につながることが確認できる。この結果は、主に次元の圧縮などに対して応用することも可能である。

L2 ノルム

L2 ノルムの値を $L2$、L1 ノルムの値を $L1$、またパラメータの合計値 $(\sum_i w_i)$ が等しい場合においては

$$L2 < L1 \tag{2.19}$$

であることが導ける。これは L1 ノルム同様に一つのパラメータを 0 に対することの影響度が関数全体に対して小さいことを表している。以上により L2 ノルムの最小化は、均一

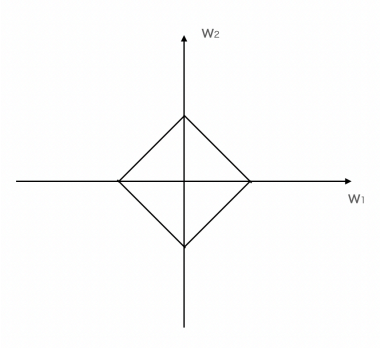

図 2.8: パラメータが二つの時の L2 ノルムのイメージ図。パラメータが二つある時、その合計値 $(\sum_i w_i)$ が 1 の点を取ると、一つのパラメータを 0 にすることが最も大きくなる。

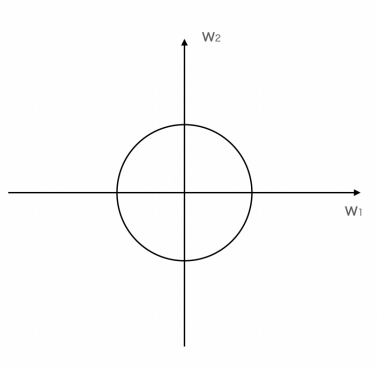

図 2.9: パラメータが二つの時の L2 ノルムのイメージ図。パラメータの合計値その合計値 $(\sum_i w_i)$ が 1 の時は $w_1 = w_2 = 0.5$ の時が最も値が小さくなるので、より高次元の場合ではこの値を大きくするのは均一的な重み分布であることが望ましい。

的にパラメータを小さくすることで、最小化を図ることができる。以上により L2 ノルムの罰則を足した合わせた損失関数により最小化したニューラルネットワークはパラメータ数が多いため、”表現力に優れている”と表現することが可能である。これは過学習の回避に用いることができる。

2.3.4 データセットの選択

ニューラルネットワークの最適化の際に学習データセットをどのように用いるかによって性能が大きく変わることが知られている。データセットの集合を $\mathcal{D}$ 大きく分けて以下の三つの方法があることが知られている。

表 2.2: 実験に用いるデータ集合の表現方法

扱うデータ	名称	説明
$\mathcal{D}_i \subseteq \mathcal{D}$	ミニバッチ学習	データを部分的にランダムで取り出して学習を行う
$d_i \in \mathcal{D}$	オンライン学習	データを一つ取り出して学習を行う
$\mathcal{D}$	バッチ学習	全てのデータを用いて学習する

バッチ学習は安定した学習が行える一方で欠点は大きく以下の二つがある。

- 損失関数の形が変わらないため、最適化の手法によっては学習が停滞してしまう。

- データ数が大きい場合には損失関数が大きくなり、メモリ不足等の問題が発生する。

また、オンライン学習は局所界に陥りにくいというメリットがあるものの、以下欠点が存在する。

- 最初より最後のデータに過剰に適合してしまう。
- 外れ値にも反応しやすいため、パラメータの収束が不安定になる。

以上により一般的に両者の欠点を抑えたミニバッチ学習が実務では使用されることが多い。

2.3.5 Optimizer

ニューラルネットワークの学習の目的は、損失関すの値をできるだけ小さくするパラメータを見つけることである。これは最適なパラメータを見つける問題であり、その問題を解くことを最適化という。

2.15 で表現した手法も最適化の一つである。これは確率的勾配効果法 (SGD) と言って単純な方法であるが、パラメータ空間を闇雲に探すよりは遥かに効率的な方法である。しかしながら、SGD 以外にもパラメータをよりよく最適化する方法は多く研究されている。

SGD

SGD は式 2.15 でも記したように以下の式の形で一般的に広く知れ渡り、実装が簡単な手法として認知されている。

$$\mathbf{W} \leftarrow \mathbf{W} - \mu \frac{\partial \mathrm{E}}{\partial \mathbf{W}} \tag{2.20}$$

$$\tag{2.21}$$

ここで、$\mathbf{W}$ は各パラメータ w_i をベクトルで表現したものである。

しかしながら式からも導けるように SGD には欠点とした以下の二つが広く認知されている。

- 勾配が 0 の点では学習が進まなくなる。
- 勾配の方向が本来の最小値の方向を指していないことがある。

これらの理由により近年では実用的には使用されていない。

Momentum

Ning Qian の Momentum(1999) [20] は SGD の勾配の方向が本来の最小値ではないという考えから、物理の法則を応用するような形で生まれた勾配法の一つである。

Momentum という手法は、式で次のように表される。

$$\mathbf{v} \leftarrow \alpha\mathbf{v} - \mu\frac{\partial \mathrm{E}}{\partial \mathbf{W}} \tag{2.22}$$

$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v} \tag{2.23}$$

ここで新しく $\mathbf{v}$ という変数が登場する。これは一つ前の勾配の速度のようなものを記録しており、勾配が急なところでは大きな値になり、小さなところでは値が小さくなる。これにより SGD に比べると更新するときの”ジグザグ度合い”のようなものが軽減され、学習が安定し高速化することが知られている。

AdaGrad

ニューラルネットワークの学習では学習係数の値が重要になる。これを初めは大きく学習し、次第に小さ学習する、学習係数の減衰 (learning rate decay) という方法がよく使われる。これを発展させた方法に John Duchi の AdaGrad(2011) [21] というものがある。AdaGrad は以下の式で表現できる。

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + \frac{\partial \mathrm{E}}{\partial \mathbf{W}} \odot \frac{\partial \mathrm{E}}{\partial \mathbf{W}} \tag{2.24}$$

$$\mathbf{W} \leftarrow \mathbf{W} - \mu\frac{1}{\sqrt{\mathbf{h}}}\frac{\partial \mathrm{E}}{\partial \mathbf{W}} \tag{2.25}$$

ここで $\odot$ は行列の要素ごとの掛け算を意味する。パラメータ更新の際に $\frac{1}{\sqrt{\mathbf{h}}}$ を乗算することで、学習スケールを調整するという手法である。これにより、よく動いた学習パラメータは次第に小さくなる。

Adam

物理的なテクニックを応用する Momentum と学習係数を調整する AdaGrad を掛け合わせたせるのが Adam(2014) [22] である。機械学習の世界では最も頻繁に用いられる。

2.3.6 勾配爆発問題

勾配爆発問題とは、ニューラルネットワークの設計において、勾配が発散することで学習が進まなくなる技術的な問題のことである。勾配爆発の問題は RNN [23] などの行列の積が絡む問題や GAN [24] などの誤差関数が不連続のな場合では簡単に発生する問題であ

る。これを対処するためには簡単な手法としてパラメータの勾配の上限値を決めるクリッピングという操作を用いる方法などが挙げられる。勾配を g、勾配の上限値を M で表現した場合に、勾配を以下式 (2.26) のように変換する。

$$\dot{g} = \frac{M}{\|g\|} g \tag{2.26}$$

2.4 問題点

本節で現状の機械学習および活性化関数の周辺に存在する社会的観点も踏まえた問題点を述べる。

2.4.1 実社会における学習の問題点

ディープラーニングを GUI で簡易的に扱えるツールは様々な企業が積極的に開発しており、50 以上のツールがあることが確認されている [25]。しかしながらこれらのツールの共通として存在する問題点は画像分類に強いなどといった特化型となっており、それぞれの問題に応じて使い分ける必要が出てくるということである。機械学習では問題は大きく分けて回帰と分類の二つが存在するが、初学者にはそのような問題に応じたモデルの構築や学習を各ツールと状況に応じて使い分けることは非常に困難である。このような点からコスト的面でも、初学者向けのツールであってもデータセットの形や目的を意識せずとも利用できるような学習アルゴリズムが求められている。

2.4.2 汎用的な活性化関数の問題点

ReLU や Swish, Mish といった活性化関数は、どれも実験的に精度が向上すると言う理由で選択されたものである。そのため、あらゆるパターンにおいて最適かどうかは未知数である。このような問題を解決するため、Alberto Marchisio(2018) [26] が提唱した手法では、既存の活性化関数の中から最適なもの見出し精度を向上させる方法を提唱した。しかし、この方法にも欠点があり既知の活性化関数が問題に応じた適切な活性化関数かどうか判断する方法は存在しない。Garrett Bingham(2020) [27] は進化的アルゴリズムを用いて x^2 や $\sin(x)$ といった原始的関数を組み合わせ最適な活性化関数を見つけることが提唱されている。しかしこの方法は、計算量が重く、関数全体の空間を探査できるかどうかは原始的関数の組み合わせに依存してしまう。

第3章　提案手法

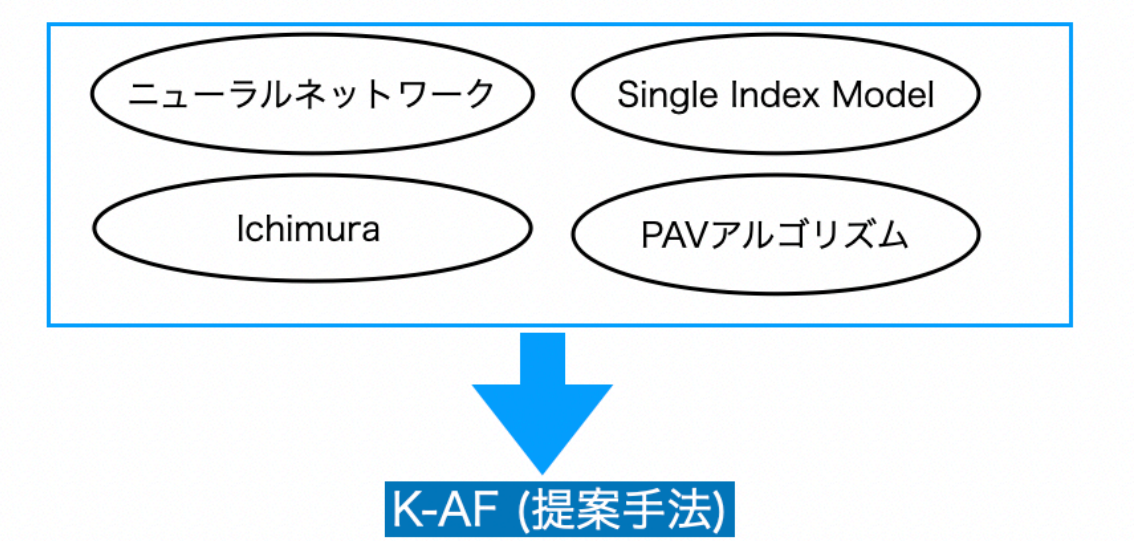

図 3.1: 提案手法のイメージ。統計の世界で用いれてきた手法を機械学習の世界に組み合わせる。

本章では、まず 3.1 節で本研究が取り組むべき問題を背景を整理しながら述べる。3.2 節ではその問題を踏まえ、課題を明確に記す。3.3 節では 3.2 節で述べた課題を解決する本論文の提案手法の着想点と歴史的位置付けについて述べる。3.4 節では本研究の研究的な位置付けについて解説する。3.5 節では K-AF の式について解説し、活性化関数として使用できることを示す。最後に 3.6 節では K-AF のアルゴリズムについて概説し、実装の理解を深める。

3.1　問題の背景

現在ニューラルネットワークの構築に使われている活性化関数は組み合わせは問題に応じて、表 3.1 のような組み合わせで構築することが多い。

しかし、これらの組み合わせは経験的であるという問題を抱えているだけではなく、問題やデータに対する人間の知識が事前に必要とされる。

表 3.1: 問題に応じた活性化関数の選択

問題の種類	中間層に使う活性化関数	出力層に使う活性化関数
分類	ReLU など	Sigmoid など
回帰	ReLU など	ReLU など

3.2 本研究が取り組むべき課題

第 2 章より、活性化関数は今もなお汎用的で精度が向上するような手法が模索されている。また、これまで用いてきた活性化関数が問題に対して最適だったか確認する手立てが求められている。本研究ではそれを踏まえて以下の 2 つの課題に取り組む。

- 高い汎用度で活性化関数が自動で推論でき、既存のものより安定的で精度がよくなること。
- 関数全体から活性化関数が推定でき、扱っていた活性化関数との差分を確認できるようにすること。

3.3 提案手法の着想点

本研究で提案する活性化関数は、Swish や Mish などがの活性化関数の単調増加性の仮定を外したところから着想を得た。2.2.2 節より、Sigmoid はロジスティック回帰から生まれたものであり、ReLU 自体も実験的に精度が良いと導出されただけのものであったため、単調増加性の仮定は必要ないものだったのである。図 3.2 のように、活性化関数は Swish や Mish などといった形へと進化する中で、単調増加性を仮定する必要がなくなった。本研究ではそのような歴史と、SIM などのセミパラメトリックモデルを参考に、カーネル密度推定を用いた活性化関数を提案する。カーネル密度関数という汎用的な関数で、学習とともに活性化関数自体も推論することでモデルにあった活性化関数が構築でき、より精度の高い結果を導き出すことができると予想される。これにより活性化関数を既存の関数から選択するのではなく、関数空間全体から活性化関数を推定することができる。また、これまで機械学習で課題とされてきた活性化関数の選択の問題を解決も、カーネル密度関数の形を考察することで導けるようになることが予想される。加えて、これまで選択してきた活性化関数が実験的に正しいかどうか判定することも可能となる。

3.4 提案手法の位置付け

本研究の提案手法の位置付けをまとめた図を 3.1 に示した。統計学の世界で SingleIndexModel における Ichimura(1993) の手法を多次元化し、学習アルゴリズムに PAV アルゴリズムを組み合わせた手法をニューラルネットワークの活性化関数に応用する。

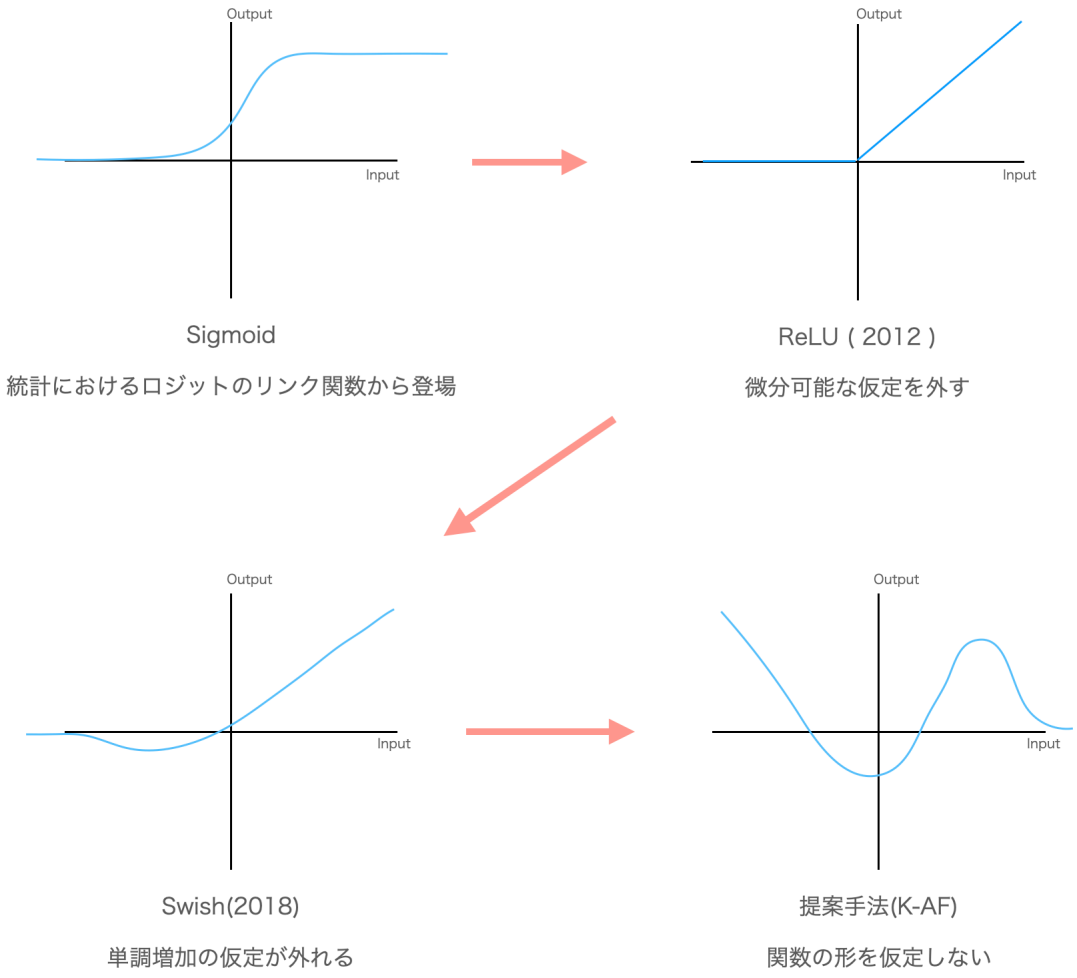

図 3.2: 活性化関数の歴史

3.5 K-AF

本論文で提案する活性化関数を式 3.1 で表現する。

$$\tilde{G}(\mathbf{X}_i, \mathbf{W}, \mathbf{X}^{calc}, \mathbf{Y}^{calc}) = \frac{\sum_{i \neq j} K\left(\frac{\mathbf{X}_j^{calc}\mathbf{W} - \mathbf{X}_i\mathbf{W}}{h_{calc}}\right) \mathbf{Y}_j^{calc}}{\sum_{i \neq j} K\left(\frac{\mathbf{X}_j^{calc}\mathbf{W} - \mathbf{X}_i\mathbf{W}}{h_{calc}}\right)} \tag{3.1}$$

$\mathbf{X}_j^{calc}$ 及び $\mathbf{Y}_j^{calc}$ は計算用に用いるデータ点である。現在の機械学習における問題の多くは学習に用いるデータセットが非常に大きい。そのため、Ichimura(1993) [6] ではデータセットの数だけで表現していたが、一部を省略することにより少ないデータ点で表現する。少ないデータ点で記述することは、活性化関数の形の単純化と計算量を減らすことにも直結する。

詳細な式の導出は appendixA.1 で述べた。

3.5.1 バンド幅推定

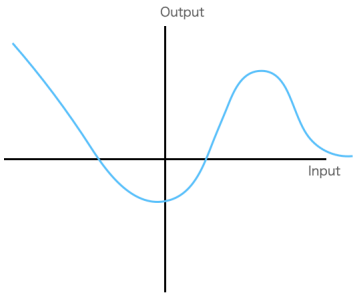

図 3.3: バンド幅が大きいと、K-AF で表現できる活性化関数の数は減る。

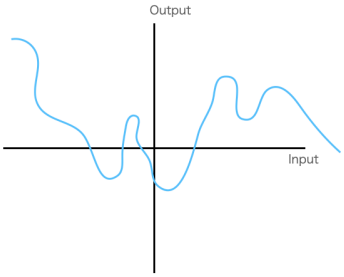

図 3.4: バンド幅が小さいと、K-AF で表現できる活性化関数の形は大きくなる。

カーネル密度推定は関数の形状を推定するにあたってバンド幅の大きさが非常に重要になる。バンド幅は大きいほど関数の自由度が減り、小さいほど自由度が大きくなることが容易に推定できる。

図 3.4 の方が表現の幅が広いが、勾配爆発の問題や過学習の問題が考えらる。本実験ではこのバンド幅も学習のパラメータに含めることで最適なバンド幅を推定できるようにした。

3.6 アルゴリズム

K-AF のアルゴリズムの概要を **Algorithm 1** に記述した。

入力の次元を d　出力の次元を e とする。m をミニバッチのサイズ、$\mathbf{W}^t$ を t ステップ目の重みのパラメータとする。E を損失関数とした時、以下のアルゴリズムで最適化を表現することができる。

Algorithm 1 K-AF

Input: data $\langle(\mathbf{X}_i, \mathbf{Y}_i)\rangle_{i=1}^{m} \in \mathbb{R}^d \times \mathbb{R}^e$, $G : (\mathbb{R}^d, \mathbb{R}^{d\times n}, \mathbb{R}^{e\times n},) \to \mathbb{R}^e$.
$\mathbf{X}^{calc} \sim_n \mathcal{D}_x$;
$\mathbf{Y}^{calc} \sim_n \mathcal{D}_y$;
for $t = 1, 2, \ldots$ **do**
　$g^t := \tilde{G}(\mathbf{W}^t, \mathbf{X}^{calc}, \mathbf{Y}^{calc})$;
　$\min_w \mathrm{E}(w) = \frac{1}{2}\sum_m (\mathbf{Y}^{true} - g^t(\mathbf{X}_i))^2$
end for

- データセット $\mathcal{D}_x$,$\mathcal{D}_y$ から任意個数の $\mathbf{X}^{calc}$ と $\mathbf{Y}^{calc}$ から取り出す。

- 現状のパラメータ $\mathbf{W}^t$, $\mathcal{D}_x$, $\mathcal{D}_y$ を用いて、リンク関数 g^t のカリー化を行う。
- そのリンク関数を用いて $\mathbf{Y}^{true}$ の値との最小二乗誤差を取り $\mathbf{W}^{t+1}$ を求める。

python での実装については appendixB に記述した。

第4章　実装

3.2 節を踏まえつつ 3 節で解決に近づけるために、本章では 3 つの実験とその設定内容について述べる。

本章では本研究における実装環境, 提案手法の実装, 提案手法の評価に用いるデータセットについて述べる．4.1 節では本研究における実験のための実行環境及び事前知識について述べる。4.2 節では、K-AF の性能を最も引き出す可能性がある学習の設定を調査する。実験 1 (4.3 節) では K-AF の性能を既存の活性化関数と比較する実験を行う。また、関数の形状を学習すると同時にどのように損失関数が小さくなるか ValidationLoss のグラフを取得する。実験 2 (4.4 節) では各データセットにおいての活性化関数の形を調査する実験を行う。実験 3 (4.5 節) では、K-AF の性能を最も引き出す可能性がある、学習の設定を調査する。

これら 3 つの実験を踏まえて、K-AF の課題に対しての実用性だけでなく、K-AF の性質についても調査し、今後の発展のための鍵となる結論へとつなげる。

4.1　実装環境

本研究において利用した実装環境を 表 4.1 に記す。また、本手法を動かすために使用したライブラリのバージョンを表 4.2 に記す。PyTorch [1], Chainer [28], Tensorflow [2] は計算グラフの自動微分ライブラリであり、深層ニューラルネットワークの研究や開発にも用いられる。その中でも Pytorch は実装コストが低く研究領域に適用しやすいことから、本研究で用いた。

表 4.1: 本研究の実行環境

項目	仕様
CPU	2.6GHz Intel core i7
メモリ	16GB 2400 MHz DDR4
ストレージ	Macintosh HD 256GB
OS	macOS High Sierra 10.13.6

表 4.2: 本研究のに使用したライブラリのバージョン

環境	バージョン
Python	3.8.5
scikit-learn	0.23.2
numpy	1.18.5
pytorch	2.5.0

4.2 本実験での共通項目

実験に用いる活性化関数の形を図 4.1 に示す。

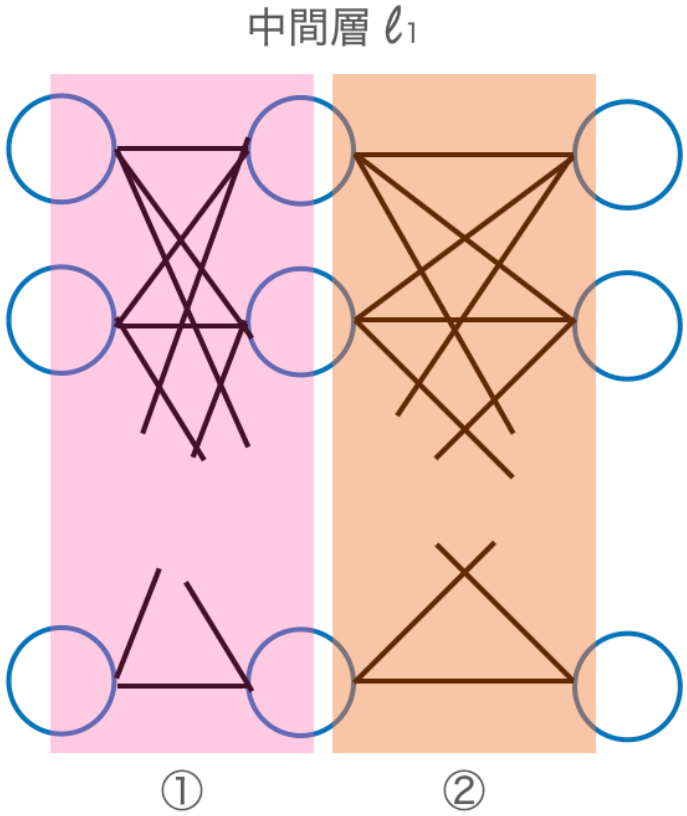

図 4.1: 実験で用いるニューラルネットワークの概要図

今回の実験では、簡易的なモデルで活性化関数の性能を試していく。中間層の数を l_1 とし、①には ReLU を用いる。②の部分を可変的にさまざまな活性化関数へと変えていく。本論文で提案する、K-AF は②の部分において性能を評価する。

比較用の活性化関数には以下の表 4.3 に記述してあるものを用いる。

表 4.3: 本実験で使用する活性化関数リスト

ReLU
Sigmoid
Tanh
Mish
Swish
K-AF(本手法)

活性化関数の性能の比較実験のために、以下の表項目 4.4 を変えながら実験する。データセットには sckit-learn [29] のライブラリのデータセットに対してデフォルトで入ってるものを想定する。各種データセットの詳細については `https://scikit-learn.org/stable/datasets/toy_dataset.html` こちらのページが参考になる。

表 4.4: 学習においての使用するニューラルネットワークの学習アルゴリズム

学習においての変更項目	変更のリスト
LearningRate	[10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6} ,10^{-7}]
Initializer	[Xavier, KaimingUniform]
Regularizer	[何もなし, L1 ノルム, L2 ノルム]
Optimizer	[SGD, Momentum, AdaGrad, Adam]
データセット	[iris, digits, wine, boston, breast_cancer]

4.2.1 比較データ

他の活性化関数と適当に比較するために、以下の条件を比較して実験を行う。

表 4.5: 実験 1 に用いるデータセットの情報

データセット名	サンプル数	入力の次元	出力の次元	出力の形式	中間層の数 l_1
iris	150	3	3	分類	4
digits	1797	3	64	分類	100
wine	178	3	13	分類	40
boston	506	13	1	回帰	20
breast_cancer	569	30	2	分類	30

4.2.2 実装における留意点

実験のために必要なハイパーパラメータとして以下のパラメータを実験前に設定することとした。

表 4.6: 本実験において使用するハイパーパラメータ

ハイパーパラメータ	記号	説明
epoch 数	epoch_num	学習の速度、最終的な ValidationLoss を確認するために十分な epoch 数を取る。
実験回数	exp_num	数回実験を行なった平均をとり、結果を保証する。
カーネル密度推定に用いるデータ数	calc_num	$\mathbf{X}^{calc}$ に用いるデータ数を表す。

また本論文の実験では学習の手法として、オフライン学習（バッチ学習）により全ての実験を行なった。そのため epoch 数は学習の際にループで回した回数に相当する。

4.3 実験 1 既存の活性化関数との比較実験

4.3.1 実装手法

表 4.4 に記述した 5 つのデータセットを軸に 4.3 の活性化関数での比較実験を LearningRate、Optimizer、Initializer、Regularizer を変更しながら実験した。K-AF の性能が有利にならないように、可能な限り任意の組み合わせで実験を行い性能の評価を行う。

4.3.2 評価手法

回帰問題と分類問題におけるれぞれの評価時における精度指標（Metrics）として、回帰問題では平均二乗誤差（MSE）を、分類問題では正解率（Accuracy）を用いる。これらは、最も基礎的で一般的な評価関数である。

また本実験では ValidationData は TrainingData と等しいものとして実験を進める。

MSE の式

評価に用いる推論したデータを $\hat{y}_i$、正解のラベル $y_i^{true} \sim \mathcal{D}_y$ とした時

$$\mathbf{MSE} = \frac{\sum_n \|\hat{y}_i - y_i^{true}\|}{n} \tag{4.1}$$

で計算することとする。

Accuracy の式

評価に用いる推論したデータを $\hat{y}_i$ と正解のラベル $y_i^{true} \sim \mathcal{D}_y$ とした時

$$\mathbf{Acc} = \frac{\sum_n \|\hat{y}_i == y_i^{true}\|^2}{n} \tag{4.2}$$

ここで、$\|\hat{y}_i == y_i^{true}\|$ とは $\hat{y}_i$ と y_i^{true} が等しければ 1、等しくなければ 0 を表現する式とする。

4.3.3 各種データセットでの比較実験の詳細

各種データセットでは Accuracy による性能の比較だけでなく ValidationLoss のグラフから学習速度を評価する。また、step 数が 0 に近い場合活性化関数によってはかなり大きな値が出力されてしまいグラフ全体が可視化に際に評価しにくくなるので、100 から表示することとする。

iris での比較実験

iris はアヤメの分類問題であり、統計学の分野で最も一般的に性能評価がおなわれるものである。optimzer に SGD を用い Regularizer を特に使わない理由は iris は単純なデータセットのため、収束することを前提に考えているからである。実験に用いる設定と環境を表 4.7 に示した。

表 4.7: iris での実験と設定

設定名	設定 1	設定 2
LearningRate	10^{-1}	10^{-2}
Initializer	KaimingUniform	KaimingUniform
Optimizer	SGD	SGD
Regularizer	なし	なし
epoch_num	1000	1000
exp_num	3	3
calc_num	30	30

digits での比較実験

digits は数字の画像の分類問題であり、機械学習の分野で一般的に性能評価の対象として使用されるものである。digits は次元数が高いデータセットであるため、Optimizer には Adam と SGD, レギュライザーには L1 ノルムと L2 ノルムを使用した。実験に用いる設定と環境を表 4.8 に示した。

表 4.8: digits での実験と設定

設定名	設定 1	設定 2
LearningRate	10^{-2}	10^{-3}
Initializer	KaimingUniform	KaimingUniform
Optimizer	SGD	SGD
Regularizer	なし	なし
epoch_num	2000	10000
exp_num	3	3
calc_num	36	36

wine での比較実験

wine とはワインの種類の判別を 13 個の入力次元の性質を用いて 3 つにカテゴライズするデータセットである。主に決定木の性能評価に使われることが多い。本実験では以下のデータセットを用いて行う。実験に用いる設定と環境を表 4.9 に示した。

表 4.9: wine での実験と設定

設定名	設定 1	設定 2
LearningRate	10^{-3}	10^{-4}
Initializer	KaimingUniform	KaimingUniform
Optimizer	SGD	Adam
Regularizer	なし	なし
epoch_num	3000	3000
exp_num	3	3
calc_num	27	36

boston での比較実験

boston はボストンの住宅価格を回帰で推定する問題である。K-AF が回帰問題に対しても有効であることを示すため、この実験を行う。実験に用いる設定と環境を表 4.10 に示した。

また、boston の実験では K-AF が勾配爆発することがあったが、今回は性能評価のため、勾配爆発しなかった場合で MSE を求めた。

表 4.10: boston での実験と設定

設定名	設定 1	設定 2
LearningRate	10^{-5}	10^{-5}
Initializer	KaimingUniform	Xavier
Optimizer	SGD	Adam
Regularizer	なし	なし
epoch_num	1000	1000
exp_num	5	5
calc_num	26	26

breast_cancer での比較実験

breast_cancer は乳がんのデータセットである。実験に用いる設定と環境を表 4.11 に示した。

表 4.11: breast_cancer での実験と設定

設定名	設定 1	設定 2
LearningRate	10^{-3}	10^{-3}
Initializer	KaimingUniform	KaimingUniform
Optimizer	SGD	SGD
Regularizer	なし	なし
epoch_num	1000	1000
exp_num	5	5
calc_num	29	285

また、breast_cancer の実験では K-AF が勾配爆発することがあったが、データセットによる問題であることに気づいたので、発散した場合も含めた Accuracy を結果に記した。

4.4 実験 2 K-AF の関数形状の調査及び損失関数の詳細調査

2 つ目の実験では推論した活性化関数の形を表 4.13 に記載したデータセット観測し、既存の活性化関数との違いを定性評価を行う。活性化関数の種類を分割すると以下のようになっている。また、ニューラルネットワークの設定は表 4.3 の設定で各データセットごとに二つずつ結果を取得する。

表 4.12 を軸に、K-AF の形状を分析し、既存の活性化関数と比べて何が違うか評価を行う。

表 4.12: 活性化関数の関数的な意味

単調増加関数か	勾配が 0 の点があるか	上限値があるか
○ or ×	○ or ×	○ or ×

またそのような活性化関数を計算するためのニューラルネットワークの設定を以下の表に示した設定で行う。

表 4.13: 実験 2 に用いるデータセットの情報及び LearningRate の設定

データセット名	LearningRate	Optimizer	Initializer	Reguralizer	中間層の数	calc_num
iris	10^{-1}	SGD	KaimingUniform	なし	4	30
digits	10^{-1}	SGD	KaimingUniform	なし	100	36
wine	10^{-2}	SGD	KaimingUniform	なし	40	36
boston	10^{-5}	SGD	KaimingUniform	なし	20	26
breast_cancer	10^{-2}	SGD	KaimingUniform	なし	30	285

また、本実験は勾配爆発が起きていない場合の活性化関数の形状を記録し、評価する。

4.5 実験 3 K-AF の性能が上がる条件探査

K-AF は実験の中で勾配爆発問題が発生することが発覚した。特に boston や breast_cancer では適当なパラメータで学習を開始すると、モデルが発散する可能性が高いことが実験の中から判明した。breast_cancer の場合に実際に発散したことは本論文でも明らかになっている。また、勾配の爆発問題はパラメータの勾配を上限値でクリッピングすることである程度減らすことが知られている。本実験では、二つの実験を行う。

4.5.1 実験 3.1 ニューラルネットワークの設定変更により精度向上の条件調査

K-AF が一般的にどのようなニューラルネットワークの設定の場合に性能を出しやすいか表 4.4 での設定をランダムに組み合わせて、評価する。本実験ではデータセットとして boston を用いて性能評価を行う。

その際のニューラルネットワークの設定は実験 1 で用いた表 4.5 の boston の欄を軸に設定する。他に実験において必要なパラメータを以下の表 4.14 に設定する。

epoch_num が少ない理由は発散するときは初期のタイミングで発散するからである。1000 回実験する中で勾配爆発頻度の回数を記録する実験をこなう。そして、各ニューラルネットの構成で記録しどの組み合わせが一番勾配爆発の回数が少ないか調査する。また、実験

表 4.14: 実験 3 で使うハイパーパラメータ

ハイパーパラメータ	値
epoch_num	5
exp_num	1000
calc_num	50

3.1 では LearningRate は小さい方が勾配爆発の確率が減ることは自明なため LearningRate は 10^{-5} で固定するものとする。また、calc_num が大きい場合も発散の確率が低いことが 5.5 でも明らかになるため、今回は 50 で固定をする。

4.5.2 実験 3.2 クリッピングを用いた勾配爆発の制御

実験 3.2 では表 4.10 の設定 1 の条件に対してクリッピングを用いる場合とそうでない場合に勾配爆発の回数がどのように変化するか実験をする。また学習性能についての評価を行うため、勾配爆発が発生しなかった場合の MSE を用いた比較を行う。これにより、クリッピングが K-AF の学習に対してどの程度有効か議論する。本実験では、式 2.26 の上限値 M を 0.025 で設定する。また、ハイパーパラメータの設定は表 4.14 を用いる。

第5章　評価

本章では第 4 章ので記述した 3 つの実験の結果を述べ考察を行う。

5.1　実験 1 の結果 既存の活性化関数との比較実験

本節では 4.3 節で示した設定もとに実験を行い、その結果を述べ各データセットにおいてどの程度の学習性能を示したか結論を述べる。

5.1.1　iris での比較実験

第 4 章の 4.3.3 項で設定した実験を行い、結果を以下にまとめた。

設定 1 及び設定 2 の結果

iris を用いた活性化関数の比較結果を表 5.1 に記す。

表 5.1: iris の設定 1 及び設定 2 の Accuracy

活性化関数	設定 1 の Accuracy	設定 2 の Accuracy
K-AF	72.0	26.0
Sigmoid	68.0	77.3
Tanh	20.0	53.0
ReLU	49.6	38.3
Swish	87.3	47.3
Mish	60.0	65.0

設定 1 及び設定 2 の ValidationLoss

ValidationLoss のログデータを図 5.1 と図 5.2 に示す。

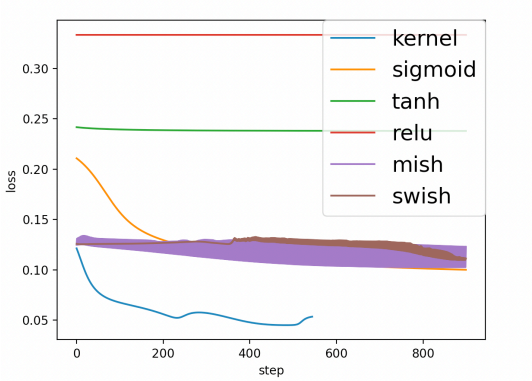

図 5.1: iris の設定 1 の結果の ValidationLoss

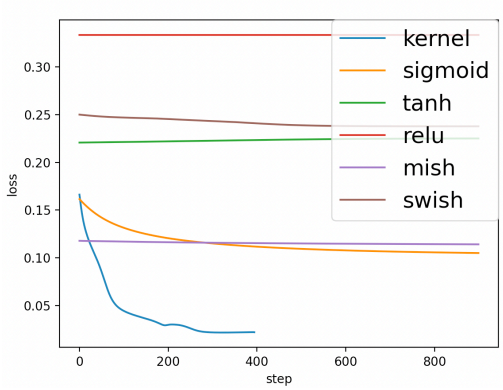

図 5.2: iris の設定 2 の結果の ValidationLoss

iris での実験結果の考察

iris は最も単純な分類問題の一つであるが結果 5.1 を分析すると特に学習を効率的にするテクニックを使わない場合は良い精度が出せている。Swish が高い性能を出していることも確認できる。また、L2 の Regularizer を入れるとかなり精度が下がってしまうことから、K-AF は既存の学習テクニックとの組み合わせでは精度が向上しないことが確認できる。

5.1.2 digits での比較実験

第 4 章の 4.3.3 項で設定した実験を行い、結果を以下にまとめた。

設定 1 及び設定 2 の結果

digits を用いた活性化関数の比較結果を表 5.2 に記す。

表 5.2: digits の設定 1 及び設定 2 の Accuracy

活性化関数	設定 1 の Accuracy	設定 2 の Accuracy
K-AF	91.6	60.0
Sigmoid	92.0	68.6
Tanh	95.6	89.6
ReLU	38.0	54.3
Swish	50.3	56.6
Mish	55.6	53.3

設定 1 及び設定 2 の ValidationLoss

ValidationLoss のログデータを図 5.3 と図 5.4 に示す。

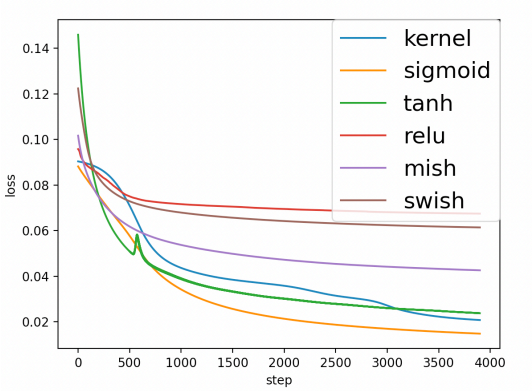

図 5.3: digits の設定 1 の結果の Validation-Loss

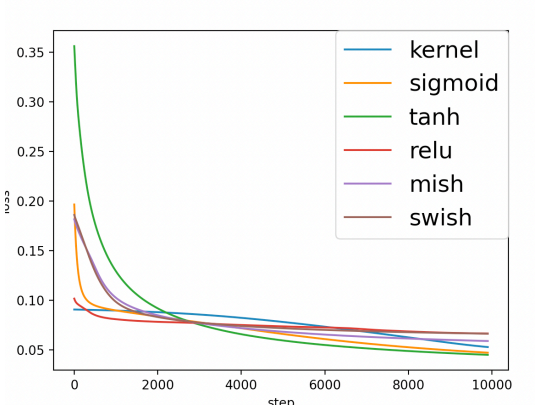

図 5.4: digits の設定 2 の結果の Validation-Loss

digits での実験結果の考察

結果 5.1.2 を確認すると設定 1 の場合では 91.6%と Sigmoid や Tanh といったラベリングによく用いられる活性化関数に匹敵する精度を出せた。ReLU、Mish、Swish 等のの上限値が存在しない関数よりも遥かに良い性能を出すことができた。設定 2 の場合でも Sigmoid や Tanh ほどの成果は出なかったものの、上限値が存在しない関数よりはいい性能を出すことができた。

ValidationLoss のグラフ 5.1.2 を確認すると、設定の 1,2 共に順調に下に推移してることが確認できる。設定 1 の方で途中で Tanh を追い抜いている理由は、活性化関数の形が急激に変わる瞬間があるからだと推測できる。

5.1.3 wine での実験と設定

第 4 章の 4.3.3 項で設定した実験を行い、結果を以下にまとめた。

設定 1 及び設定 2 の結果

wine を用いた活性化関数の比較結果を表 5.3 に記す。

設定 1 及び設定 2 の ValidationLoss

ValidationLoss のログデータを図 5.5 と図 5.6 に示す。

表 5.3: wine の設定 1 及び設定 2 の結果 Accuracy

活性化関数	設定 1 の Accuracy	設定 2 の Accuracy
K-AF	72.0	27.0
Sigmoid	33.3	43.0
Tanh	33.3	38.3
ReLU	33.3	0.0
Swish	33.3	27
Mish	33.3	27.0

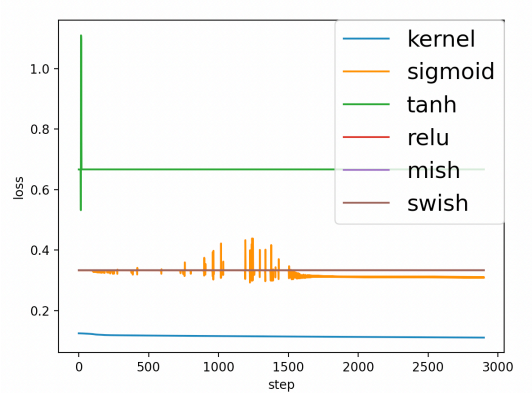

図 5.5: wine の設定 1 の結果の ValidationLoss

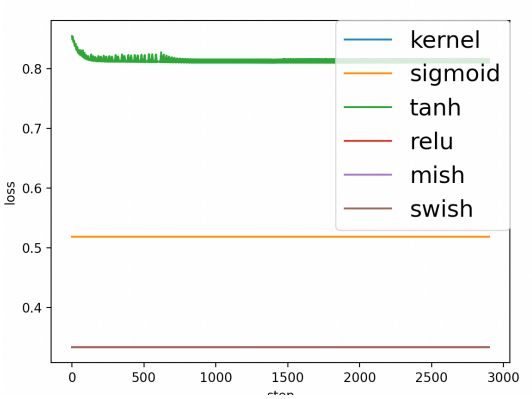

図 5.6: wine の設定 2 の結果の ValidationLoss

wine での実験結果の考察

結果 5.3 を確認すると wine はもともと決定木に使用されるデータセットのためか、設定 1 では K-AF 以外では発散してしまっていることが考えられる。また、設定 2 は Optimizer を Adam に変えただけ、学習が失敗することが明らかになった。全体を通しても SGD の K-AF が最大の Accuracy を出している。

5.1.4 boston での比較実験

第 4 章の 4.3.3 項で設定した実験を行い、結果を以下にまとめた。

設定 1 及び設定 2 の結果

boston を用いた活性化関数の比較結果を表 5.4 に記す。

表 5.4: boston の設定 1 及び設定 2 の MSE

活性化関数	設定 1 の MSE	設定 2 の MSE
K-AF	49.4	69.4
Sigmoid	523.0	541.4
Tanh	540.6	567.8
ReLU	359.2	473.4
Swish	257.0	372.4
Mish	360.0	472.4

設定 1 及び設定 2 の ValidationLoss

ValidationLoss のログデータを図 5.7 と図 5.8 に示す。

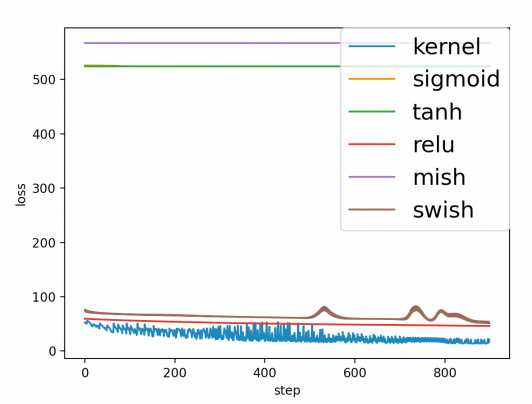

図 5.7: boston の設定 1 の結果の Validation-Loss

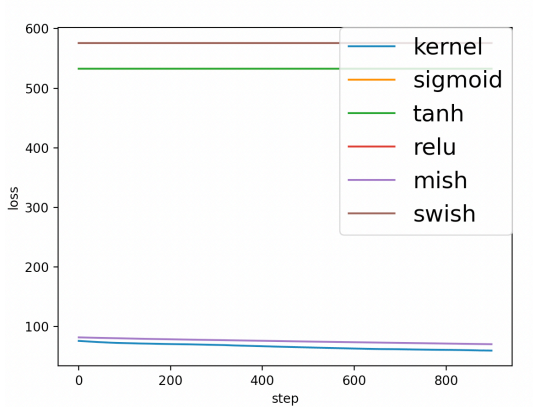

図 5.8: boston の設定 2 の結果の Validation-Loss

boston での実験結果の考察

boston は本実験で唯一の回帰のデータセットである。
結果 5.4 を考察すると、ReLU 等の関数よりも高い精度が出ることが判明した。

5.1.5 breast_cancer での比較実験

設定 1 及び設定 2 の結果

breast_cancer を用いた活性化関数の比較結果を表 5.4 に記す。

表 5.5: breast_cancer の設定 1 及び設定 2 の Accuracy

活性化関数	設定 1 の Accuracy	設定 2 の Accuracy
K-AF	43.0	90.3
Sigmoid	47.3	78.3
Tanh	60.0	29.0
ReLU	43.0	29.0
Swish	55.3	42.6
Mish	47.3	42.6

設定 1 及び設定 2 の ValidationLoss

ValidationLoss のログデータを図 5.9 と図 5.10 に示す。

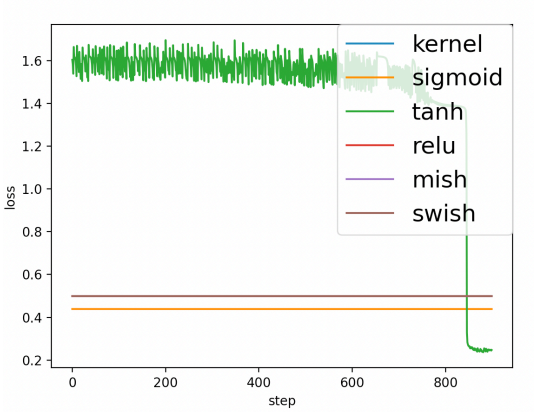

図 5.9: breastcancer の設定 1 の結果の ValidationLoss

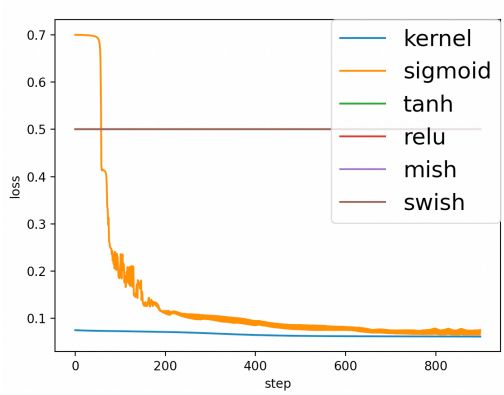

図 5.10: breastcancer の設定 2 の結果の ValidationLoss

breastcancer での実験結果の考察

結果 5.5 を考察すると、勾配爆発が起きる条件に関わるものとして、カーネル密度関数を推定するためのデータセットの数が関与してることが明らかになった。こちらも決定木の性能評価でよく使用されるデータセットであるため、ニューラルネットワークで評価をするのには向いていないことが精度が出づらいことの原因であると考えられる。設定 1 での性能が極端に低い原因は計算用のデータセットの数が少ないことによる勾配爆発が原因であると考えられる。

5.1.6 実験 1 全体のまとめ

結果全体を考察すると、K-AF は本実験で使うようなデータセットの場合でも、いい条件で学習することができれば既存の活性化関数と同等もしくはより高い精度でどのデータセットも学習できることが判明した。特に決定木の問題に対しても calc_num を多めに取ることができれば予想以上に高い精度を出すことが明らかになった。複雑な問題に対しては、LearnigRate が大きい場合や calc_num が少ない場合はかなりの確率で勾配爆発を起こすことが判明した。

5.2 実験 2 の結果 K-AF の関数形状の調査

本節では、4.4 節で示した設定もとに実験を行い、4.12 の表を軸にどのような活性化関数になったか、損失を見ながら考察を行い結論を述べる。

iris の活性化関数

実験の設定は表 4.5 の LearningRate と表 4.7 の LearningRate 以外の設定を用いる。推論した活性化関数の形は図 5.11 のようになる。

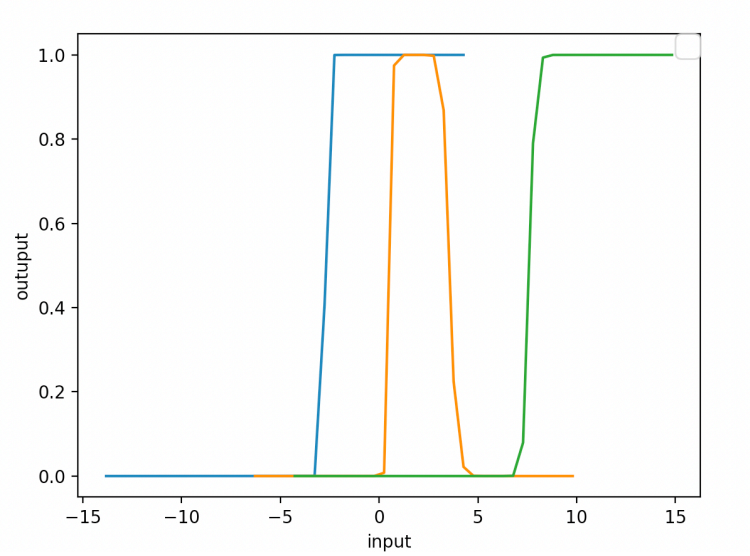

図 5.11: iris で推論した活性化関数の形

またこれを 4.12 の表に当てはめると表 5.6 のようになる。

単純な分類問題であるが、5.11 を観察するに Sigmoid 等の活性化関数と違い単調増加もしくは単調減少しない活性化関数が導かれることが明らかになった。

表 5.6: iris で推論した活性化関数の分析表

単調増加関数か	勾配が 0 の点があるか	上限値があるか
×	○	○

digits の活性化関数

実験の設定は表 4.5 の LearningRate と表 4.8 の LearningRate 以外の設定を用いる。推論した活性化関数の形は図 5.12 のようになる。

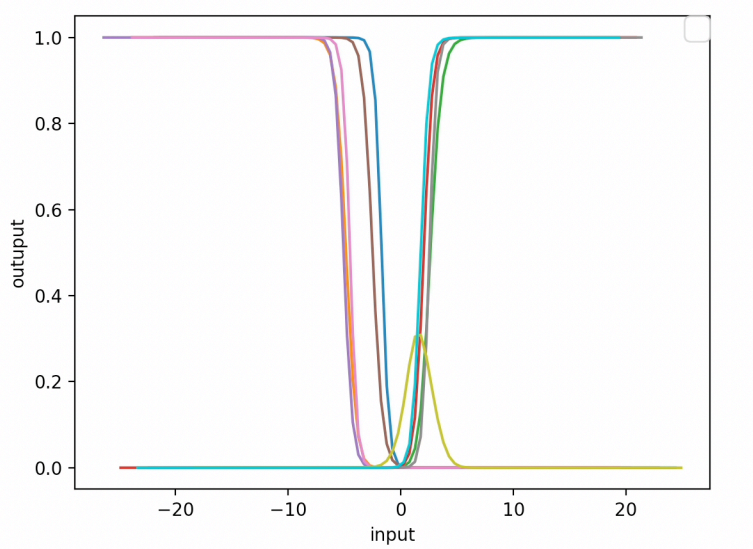

図 5.12: digits で推論した活性化関数の形。出力層は 10 次元なので 10 個の関数が存在する。

またこれを 4.12 の表に当てはめると表 5.7 のようになる。

表 5.7: digits で推論した活性化関数の分析表

単調増加関数か	勾配が 0 の点があるか	上限値があるか
▲	○	○

▲になる理由は単調増加とは対称的な単調減少の関数も推論した関数の中に存在するからである。推論するデータセットがラベリング問題であるため上限値が 1 であることから、このような形になったと考察できる。例外的に一つ形が崩れているが、Sigmoid と関数の形がにているため、使用する関数としても Sigmoid のような形のもので十分な精度が出ることが予想できる。実際に 5.1.3 では Sigmoid や Tanh でも十分な性能を出している。

wine の活性化関数

実験の設定は表 4.5 の LearningRate と表 4.9 の LearningRate 以外の設定を用いる。推論した活性化関数の形は図 5.13 のようになる。

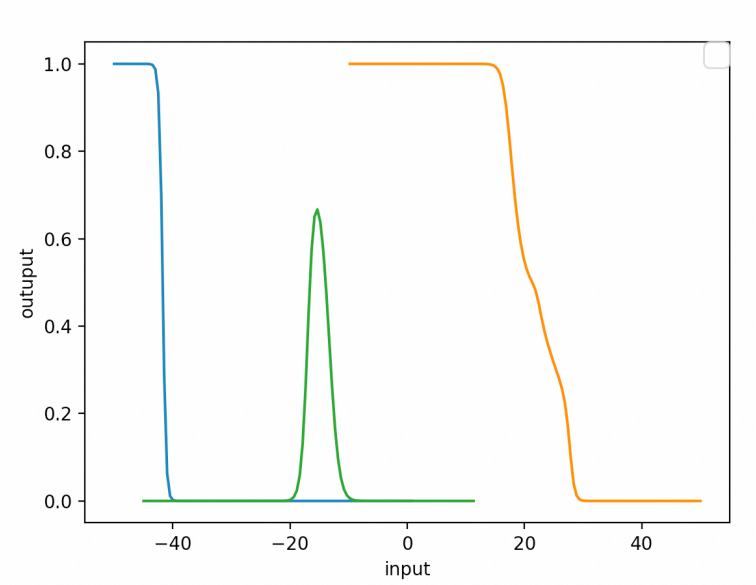

図 5.13: wine で推論した活性化関数の形。出力層が 3 次元なので 3 つの関数が存在する。

またこれを 4.12 の表に当てはめると表 5.8 のようになる。

表 5.8: wine で推論した活性化関数の分析表。出力層が 1 次元の回帰問題なので、関数は一つだけである。

単調増加関数か	勾配が 0 の点があるか	上限値があるか
×	○	○

K-AF では高い性能を出すことができた。分類問題であるため digits と同様に、上限が存在するが歪な形になっていることが確認できる。これは決定木の性能評価のために使われるデータセットのため、ニューラルネットで表現できる関数空間との相性が悪いことが原因であると考えられる。

boston の活性化関数

実験の設定は表 4.5 の LearningRate と表 4.10 の LearningRate 以外の設定を用いる。推論した活性化関数の形は図 5.14 のようになる。

またこれを 4.12 の表に当てはめると表 5.9 のようになる。

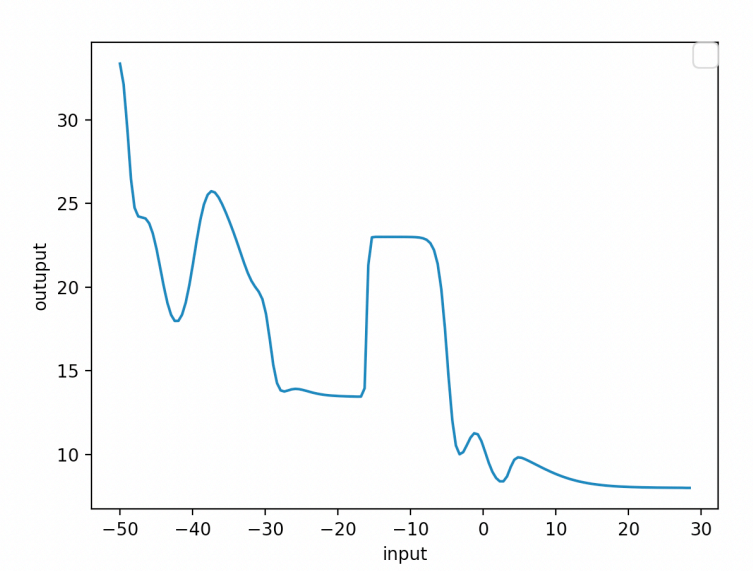

図 5.14: boston で推論した活性化関数の形

表 5.9: boston で推論した活性化関数の分析表

単調増加関数か	勾配が 0 の点があるか	上限値があるか
×	○	▲

boston は回帰問題であるため、分類問題とは違い上限値が 30 になり、その中間の値も出力するようになった。複雑な関数の形も推論できていることが確認できると同時に、このような問題に適した活性化関数がないことが原因で boston での実験 5.1.4 は非常に高い性能を出したのではないかと考察することもできる。図 5.14 のような形をした活性化関数は現状発表されてないので、既存の活性化関数では表現の幅が狭いことが確認できる良い例である。

breastcancer の活性化関数

実験の設定は表 4.5 の LearningRate と表 4.11 の LearningRate 以外の設定を用いる。推論した活性化関数の形は図 5.15 のようになる。

またこれを 4.12 の表に当てはめると表 5.10 のようになる。

表 5.10: breastcancer で推論した活性化関数の分析表

単調増加関数か	勾配が 0 の点があるか	上限値があるか
▲	○	○

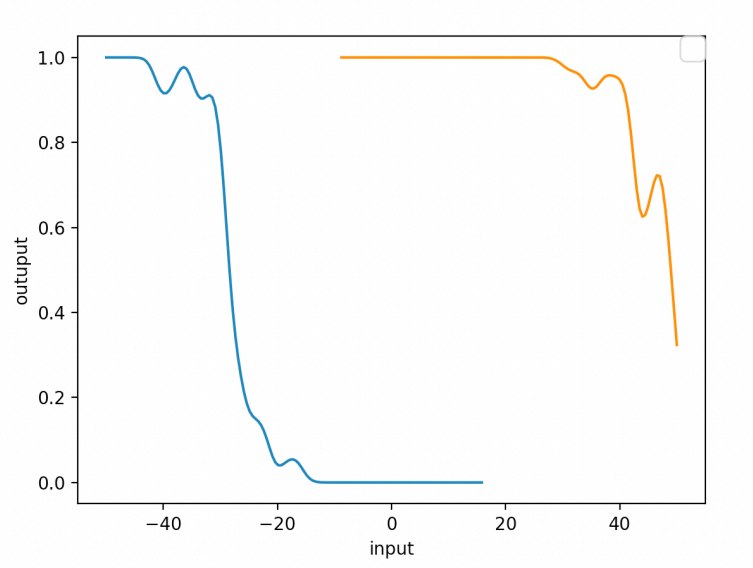

図 5.15: breast_cancer で推論した活性化関数の形

breast_cancer は分類問題であるが決定木の評価に使われる関数である。そのため、学習の不安定さ原因となり、LearnigRate が大きい場合や活性化関数に使うデータセットが少ない場合に勾配が発散することが多かった。

5.2.1 実験 2 全体のまとめ

実験 25.2 により、K-AF が活性化関数を最適に推論してることが視覚的に理解できた。また表 4.12 を用いた分析により、決定木で使うようなデータセットの場合や回帰の場合などは既存の活性化関数では表現の幅が不足していることが判明した。

また、iris、digits、wine、boston の K-AF の学習過程の動きを AppendixC で紹介している。

5.3 実験 3 の結果 K-AF の性能が上がる条件探査

本節では 4.5 節で示した設定もとに実験を行い、一般的にどのような条件で K-AF がより良い性能を出すか、また、欠点が存在するかを 5.3.1 項と 5.3.2 項にて調査する。

5.3.1 実験 3.1 ニューラルネットワークの設定変更により精度向上の条件調査の結果

4.5.1 項で示した比較実験の結果を記述する。

boston において勾配爆発の確率の少ないニューラルネットワークの構成

boston のデータセットで最も性能が良かったニューラルネットワークの設定の上位 5 つを表 5.11 に記述する。

表 5.11: boston を推論するときの最も発散確率が低いニューラルネットワークの設定の順位

順位	Initializer	Optimizer	Reguralizer	発散回数	発散確率
1	Xavier	Adam	non	275	27.5%
2	Xavier	Adam	l1	290	29.0%
3	Xavier	Adam	l2	294	29.4%
4	Xavier	SGD	l1	294	29.4%
5	Xavier	RMSprop	l1	298	29.8%

性能評価まとめ

表 5.11 の順位を確認すると性能に直結しそうなものは初期値であり、上位の順位のものが全て Xavier であることが明らかになった。また、Appendix の表 D より Xavier は KaimingUniform によりもどの条件において勾配爆発の発生回数が半分程度だったことが判明した。この議論において大切なことは、どの Initializer 性能が向上したかということではなく、Initializer そのものが K-AF の性能に大きく影響しているということである。Initializer の他には統計的な誤差の範囲内ではあるが、Optimizer として Adam を使用した場合にエラー頻度が低いことが明らかになった。

5.3.2 実験 3.2 クリッピングを用いた性能評価

4.5.2 節で示した比較実験の結果を記述する。

表 5.12: boston を用いた K-AF にクリッピングを用いた際の勾配爆発の回数と確率

構成	勾配爆発の回数	勾配爆発の確率
クリッピングなし	648	64.8%
クリッピングあり	590	59.0%

クリッピングの性能まとめ

5.12 節の結果により、クリッピングが K-AF の勾配爆発の確率を減らすことには直接影響しないことが明らかになった。

5.3.3 実験 3 まとめ

実験 3.1 により、K-AF の性能を引き出す最大の要素は Initializer に関わっていることが明らかになった。これは初期値の段階で勾配爆発に陥ってる可能性が高いことが原因であると考えられる。

第6章　結論

本章では第 5 章の 3 つの実験の結果に対する考察を行い、第 1 章で述べた貢献 1.1 にどのように繋がったか考察する。そして、今後の問題点についてまとめ、将来的な展望について記す。

6.1　本研究のまとめと解決した課題

実験 1(4.3 節) により Sigmoid では精度が低いデータセットや ReLU では精度の低いデータセットでも K-AF はある程度高い精度が出せることが明らかになった。これによりデータセットの形を意識せずとも同様の活性化関数を用いても問題なく学習を行えるので、初心者でも気軽に使える活性化関数になった。実験 2(4.4 節) により状況に応じた活性化関数の形が実際可視化されることで、既存の活性化関数の有用性を示すだけでなく、新たな活性化関数の模索の必要性を示すことができた。また、分類系の問題では Sigmoid のような活性化関数でも表現の幅として程度十分であることも可視化され理解することができた。これらの結果により、ブラックボックス化された活性化関数選択問題の解決に近づいた。実験 3(4.5 節) ではこのような K-AF がどのような場合に勾配爆発の少ない有効な活性化関数になるか定量的に示すことができた。

以上により 3.2 で述べた二つの課題について部分的に貢献し、1.1 節で述べた課題を解決した。

6.2　本研究の問題点

本研究をよりディープラーニング等のより実用的な問題に応用するためには以下の二つの課題を解決する必要がある。

6.2.1　勾配の爆発問題

K-AF は一部のデータセットにおいては適切に Initializer を設定しなければ、勾配が爆発し学習が収束しない問題が本論文でも挙げられた。今後はそのような Initializer がどのようなものであるが研究を積み重ね、より良い確率で収束させることができるアルゴリズムを発見していく必要がある。

6.2.2 ディープラーニングへの応用

本論文では K-AF は出力層の活性化関数のみを置き換え精度を上げることに成功した。しかしながら、K-AF では、次の層の正解のラベルデータを必要とするため、中間層に生かすことができない。より良い精度を出すためには中間層でも使える新しいアルゴリズムが求められる。

6.3 将来的な展望

活性化関数を汎用的に推論するという論文は未だ少なく研究分野として今後非常に注目すべきである。将来的には自動運転などの産業分野においても有用なモデルへの応用されることを望む。また、K-AF が汎用的な活性化関数の代表として初学者や非エンジニアが扱いやすい道具として応用されることを望む。

付 録A　ガウス分布とカーネル密度推定

A.1　K-AFの導出

本研究で実際に使用したアルゴリズムに用いた式実際に導出する。Ichimura(1993) [6]の手法を用いてまずは以下の式に変換する。

$$G(\mathbf{X}_i, \mathbf{W}) = \frac{\sum_{i \neq j} K\left(\frac{\mathbf{X}_j\mathbf{W} - \mathbf{X}_i\mathbf{W}}{h}\right)\mathbf{Y}_j}{\sum_{i \neq j} K\left(\frac{\mathbf{X}_j\mathbf{W} - \mathbf{X}_i\mathbf{W}}{h}\right)} \tag{A.1}$$

ここで、$\sum_{i \neq j} K\left(\frac{\mathbf{X}_j\mathbf{W} - \mathbf{X}_i\mathbf{W}}{h}\right) \approx \sum_{i \neq j} K\left(\frac{\mathbf{X}_j^{calc}\mathbf{W} - \mathbf{X}_i\mathbf{W}}{h_{calc}}\right)$ となるようなバンド幅 h_{calc} を見つけることで、全てのデータ点を使わなくとも $\mathbf{X}_{calc}$ を用いてA.1を近似することができる。

これによりA.1は以下の式に直すことができる。

$$G(\mathbf{X}_i, \mathbf{W}) \approx \tilde{G}(\mathbf{X}_i, \mathbf{W}, \mathbf{X}^{calc}, \mathbf{Y}^{calc})) = \frac{\sum_{i \neq j} K\left(\frac{\mathbf{X}_j^{calc}\mathbf{W} - \mathbf{X}_i\mathbf{W}}{h_{calc}}\right)\mathbf{Y}_j^{calc}}{\sum_{i \neq j} K\left(\frac{\mathbf{X}_j^{calc}\mathbf{W} - \mathbf{X}_i\mathbf{W}}{h_{calc}}\right)} \tag{A.2}$$

以上により、Ichimura(1993) の手法に対してデータ点を減らしても近似できることを示した。

付 録B　カーネル活性化関数の実装

B.1　クラス

中間層が一つの K-AF の計算を考慮した実装クラスを B.1 に示す。
実装の全容は https://github.com/latte0/graduation_thesis に公開している。

プログラム B.1: Pytorch を用いた K-AF の計算用のクラス

```
1  class Net(nn.Module):
2
3      def __init__(self, Y, calc_Y, X, calc_X, settings):
4          super(Net, self).__init__()
5
6          self.fc1 = nn.Linear(DATA_INPUT_LENGTH, DATA_MID_LENGTH
               , bias=False)
7          self.fc2 = nn.Linear(DATA_MID_LENGTH,
               DATA_OUTPUT_LENGTH, bias=False)
8          # leave_ont_outのために事前に入力と出力をセットしておく
9          self.Y = Y
10         self.calc_Y = calc_Y
11         self.calc = False
12         # バンド幅も推定する
13         self.h = nn.Parameter(torch.tensor(1.5, requires_grad=
               True))
14
15         self.last_layer_result = []
16         self.sigmoid = nn.Sigmoid()
17
18     # kernel推定量の計算
19     def kernel(self, Zw):
20         numerator = 0
21         denominator = 0
22         result = []
23         for j, X_j in enumerate(self.train_X):
24
25             Xw = self.fc2(F.relu(self.fc1(X_j)))
26             tmp = gauss((Xw - Zw) / self.h)
27
28             tmp[j] = 0
29             denominator += tmp
30             numerator += tmp * self.Y[j]
31
32         g = numerator/denominator
33         return g
34
35
```

```
36      def forward(self, x):
37
38          xw = F.relu(self.fc1(x))
39          xw = self.fc2(xw)
40
41          y = self.kernel(xw)
42
43          return y
```

付 録C　カーネル活性化関数の学習過程の動画

活性化関数の学習が進んでいく様子を録画し、github に公開している。

- 実験 24.4 の iris の K-AF の学習過程
  https://github.com/latte0/RG-Thesis-Template/blob/main/asset/iris.mov?raw=true
- 実験 24.4 の digits の K-AF の学習過程
  https://github.com/latte0/RG-Thesis-Template/blob/main/asset/digits.mov?raw=true
- 実験 24.4 の wine の K-AF の学習過程
  https://github.com/latte0/RG-Thesis-Template/blob/main/asset/wine.mov?raw=true
- 実験 24.4 の boston の K-AF の学習過程
  https://github.com/latte0/RG-Thesis-Template/blob/main/asset/boston.mov?raw=true

付 録D　bostonの勾配爆発回数の記録

実験 3.14.5.1 の表 5.11 を導くために必要な情報をまとめた。各活性化関数の設定においての勾配爆発の頻度を記した結果を以下の表 D.1 に記述する。

表 D.1: boston を推論した時の勾配爆発の頻度と性能。LearningRate は 10^{-5} で固定する。

Initializer	Optimizer	Reguralizer	発散回数	発散確率
KaimingUniform	Adagrad	l1	622	62.2%
KaimingUniform	Adagrad	l2	613	61.3%
KaimingUniform	Adagrad	なし	635	63.5%
KaimingUniform	Adam	l1	632	63.2%
KaimingUniform	Adam	l2	615	61.5%
KaimingUniform	Adam	なし	628	62.8%
KaimingUniform	RMSprop	l1	639	63.9%
KaimingUniform	RMSprop	l2	622	62.2%
KaimingUniform	RMSprop	なし	629	62.9%
KaimingUniform	SGD	l2	585	58.5%
KaimingUniform	SGD	l1	619	61.9%
KaimingUniform	SGD	なし	601	60.1%
Xavier	Adagrad	l1	312	31.2%
Xavier	Adagrad	l2	321	32.1%
Xavier	Adagrad	なし	314	31.4%
Xavier	Adam	l1	290	29.0%
Xavier	Adam	l2	294	29.4%
Xavier	Adam	なし	275	27.5%
Xavier	RMSprop	l1	298	29.8%
Xavier	RMSprop	l2	313	31.3%
Xavier	RMSprop	なし	328	32.8%
Xavier	SGD	l1	294	29.4%
Xavier	SGD	l2	309	30.9%
Xavier	SGD	なし	321	32.1%

謝辞

本論文の執筆にあたり、ご指導頂いた慶應義塾大学環境情報学部教授村井純博士、同学部教授中村修博士、同学部教授楠本博之博士、同学部准教授高汐一紀博士、同学部教授三次仁博士、同学部准教授植原啓介博士、同学部准教授中澤仁博士、同学部準教授 Rodney D。Van Meter III 博士、同学部教授武田圭史博士、同大学政策・メディア研究科特任准教授鈴木茂哉博士、同大学政策・メディア研究科特任准教授佐藤 雅明博士、同大学 SFC 研究所上席所員斉藤賢爾博士に感謝致します.

特に斉藤氏には重ねて感謝致します。研究活動を通して技術的視点、社会的視点等の様々な視点から私の研究に対して助言を頂き、深い思考と学びを経験させて頂くことができました。これらの経験は私の人生において人・学ぶ者として、素敵な財産として残りました。博士の指導なしには、卒業論文を執筆することは出来ませんでした。

徳田・村井・楠本・中村・高汐・バンミーター・植原・三次・中澤・武田合同研究プロジェクトに所属している学部生、大学院生、卒業生の皆様に感謝致します。研究会に所属する多くの方々が各々の分野・研究で奮闘している姿を見て学んだことが私の研究生活をより充実したものとさせました。

異なる分野同士が触れ合い、学び合う環境に出会えたことを嬉しく感じます。また、NECO 研究グループとして多くの意見・発想・知見を与えてくださった、慶應義塾大学政策メディア・研究科 阿部涼介氏、卒業生 菅藤佑太氏、在校生 島津翔太氏、宮本眺氏、松本三月氏、梶原留衣氏、渡辺聡紀氏、木内啓介氏、後藤悠太氏、倉重健氏、九鬼嘉隆氏、内田渓太氏、山本哲平氏、吉開拓人氏、金城奈菜海氏、長田琉羽里氏、前田大輔氏に感謝致します。

皆様には、私の研究に対する多くの助言や発想を頂いただけでなく、研究活動における学びを経験させて頂きました。多くの出会いと学びの環境である SFC に感謝致します。多様な学問領域に触れ、学生同士で議論し思考することが出来ました。幸せで素敵な時間でした。

最後に、これまで私を育て、見守り、学びの機会を与えて頂いた、父 良昭氏、母 ちや子氏、兄 良行氏 に感謝致します。

参考文献

[1] Pytorch. https://pytorch.org/, 2021.

[2] Tensorflow. https://www.tensorflow.org/, 2021.

[3] Neural network console. https://dl.sony.com/.

[4] Xavier. Glorot and Antoine. Bordes. Deep sparse rectifier neural networks. http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf, 2011.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. https://arxiv.org/abs/1512.03385/, 2015.

[6] Hidehiko Ichimura, Wolfgang Hardle, and Peter Hall. Optimal smoothing in single index model. https://projecteuclid.org/download/pdf_1/euclid.aos/1176349020, 1993.

[7] M. N. Favorskaya and V. V. Andreev. The study of activation functions in deep learning for pedestrian detection and tracking. https://www.researchgate.net/publication/332975597_THE_STUDY_OF_ACTIVATION_FUNCTIONS_IN_DEEP_LEARNING_FOR_PEDESTRIAN_DETECTION_AND_TRACKING, 2019.

[8] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. https://arxiv.org/abs/1505.00853, 2015.

[9] Djork-Arne Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning byexponential linear units (elus). https://arxiv.org/pdf/1511.07289.pdf, 2016.

[10] Günter Klambauer, Thomas Unterthiner, and Andreas Mayr. Self-normalizing neural networks. https://arxiv.org/pdf/1706.02515.pdf, 2017.

[11] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. https://arxiv.org/pdf/1710.05941.pdf, 2017.

[12] Diganta. Misra. Deep sparse rectifier neural networks. https://arxiv.org/pdf/1908.08681.pdf, 2019.

[13] Richard O. Duda and Peter E. Hart. Pattern classification and scene analysis, 1973.

[14] Adam Tauman Kalai and Ravi Sastry. The isotron algorithm: High-dimensional isotonic regression. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.414.453&rep=rep1&type=pdf, 2008.

[15] Roger W. Klein and Richard H. Spady. An efficient semiparametric estimator for binary response models. https://www.jstor.org/stable/2951556?seq=1, 1993.

[16] Sham Kakade, Adam Tauman Kalai, Varun Kanade, and Ohad Shamir. Efficient learning of generalized linear and single index models with isotonic regression. https://arxiv.org/pdf/1104.2018.pdf, 2011.

[17] Ravi Ganti, Nikhil Rao, Rebecca M. Willett, and Robert Nowak. Learning single index models in high dimensions. https://arxiv.org/pdf/1506.08910.pdf, 2015.

[18] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. http://proceedings.mlr.press/v9/glorot10a.html, 2010.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. https://arxiv.org/abs/1502.01852, 2015.

[20] Ning Qian. On the momentum term in gradient descent learning algorithms. neural networks :the official journal of the international neural network society, 12(1):145–151, 1999. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.5612&rep=rep1&type=pdf, 1999.

[21] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf, 2011.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, 2014.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf.

[24] an J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. https://arxiv.org/abs/1406.2661.

[25] プログラミング不要！ 約 50 の ai 構築 gui ツールをまとめたサービスマップを公開！ https://ainow.ai/2019/07/09/173221/, 2019.

[26] Alberto Marchisio, Muhammad Abdullah Hanif, Semeen Rehman, Maurizio Martina, and Muhammad Shafique. A methodology for automatic selection of activation functions to design hybrid deep neural networks. https://arxiv.org/pdf/1811.03980.pdf, 2018.

[27] Garrett Bingham, William Macke, and Risto Miikkulainen. Evolutionary optimization of deep learning activation functions. https://arxiv.org/pdf/2002.07224.pdf, 2020.

[28] Chainer: A flexible framework for neural networks. https://chainer.org/, 2021.

[29] scikit-learn machine learning in python. https://scikit-learn.org/stable/, 2021.