カーネル密度推定を用いた汎用的な活性化関数

neco B4 vapor
vapor@sfc.wide.ad.jp
Adviser ks91

研究概要

数学的モデル

機械学習の世界

ニューラルネットワーク

活性化関数・・・ニューロンの出力。
表現の幅を広げる。

- ReLU
- Sigmoid

統計の世界

一般線形化モデルなど

リンク関数・・・表現の幅を広げる

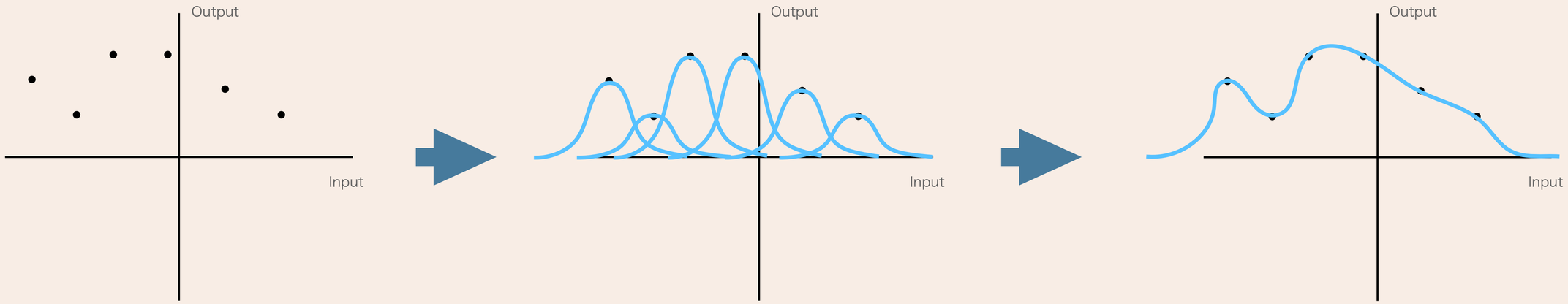
- ロジスティック回帰
- ノンパラメトリックモデル
- カーネル密度推定

新しい活性化関数の発見

背景. 1 (リンク関数のノンパラメトリック推定)

カーネル密度推定

ノンパラメトリックモデルの一つ。データ点をの周囲にカーネル関数を置き、関数の近似を行う。



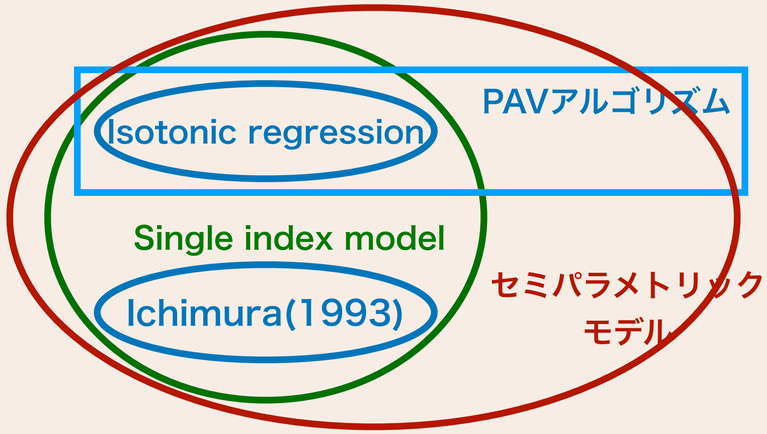
データ点がまばらに存在する。

周囲にカーネル関数を置く

カーネル関数を足し合わせて関数の近似を行う

背景. 1　（セミパラメトリックモデル）

SingleIndexModel	ノンパラメトリックとパラメトリックな手法を組み合わせたモデル（セミパラメトリックモデル）の一つ
・ Ichimura(1993) **・ Isotonic regression**	それぞれ、セミパラメトリックモデルの未知の関数を推論する。関数の推定にカーネル密度推定を用いている。
PAVアルゴリズム	高次元のIsotonic regressionを利用したモデルの推論アルゴリズムの一つ



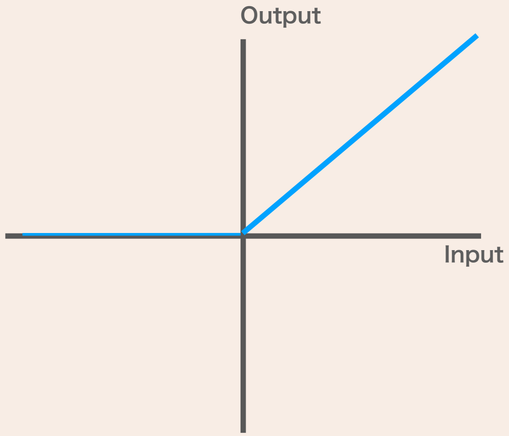
各用語の立ち位置

全て統計学の世界で発見されてきたモデルである。

背景. 2 (機械学習における活性化関数)

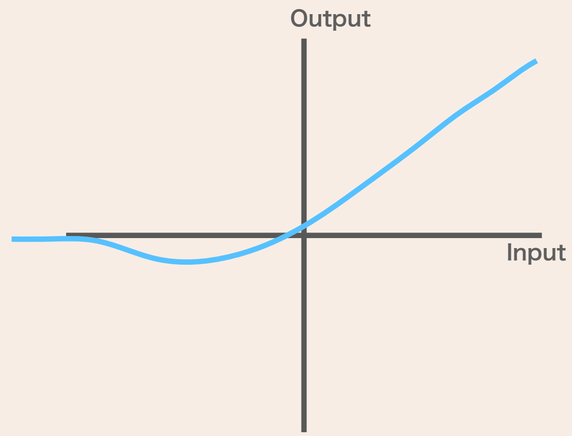
既存の活性化関数の種類と形状

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	$x \cdot \tanh(\log(1+\exp(x)))$



ReLU (2012)

微分可能な仮定を外す



Swish(2018)

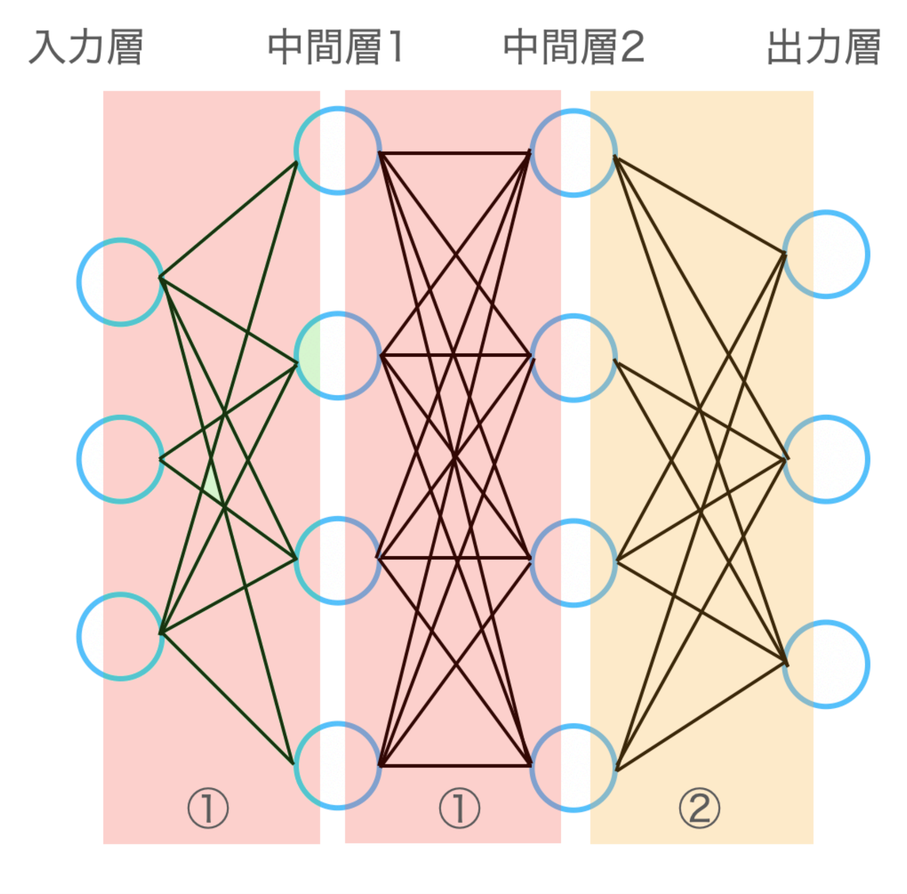
単調増加の仮定が外れる

関数の形状が単調増加のシンプルなものから、複雑な形状の方が精度が上がる。

より精度の高い活性化関数が求め続けられている。

背景. 2（活性化関数の問題点）

ニューラルネットにおける活性化関数の構成



「分類」や「回帰」といったデータの出力形式によって、②の出力層に使われる活性化関数が変わる。

問題の種類	入力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

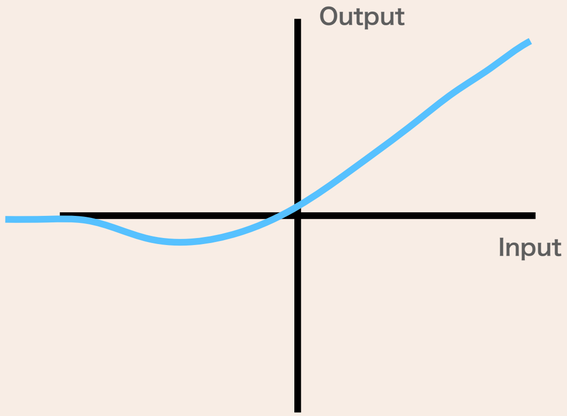
関数の選択は、経験則に基づく必要があり、何がいい選択かわからない

本研究が取り組むべき課題

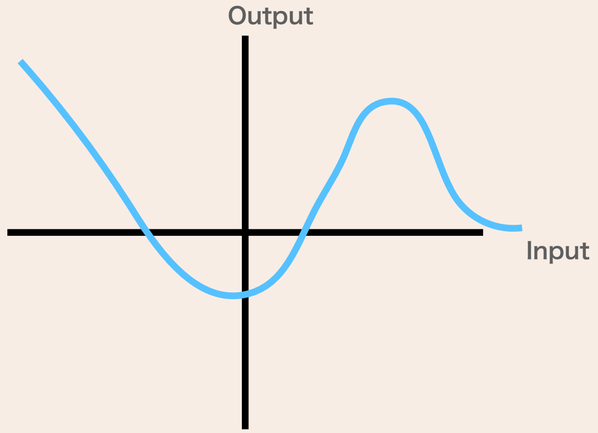
1. 高い汎用度で活性化関数が自動で推論でき、既存のものより安定的で精度がよくな ること。

2. 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。

仮説



Swish(2018)

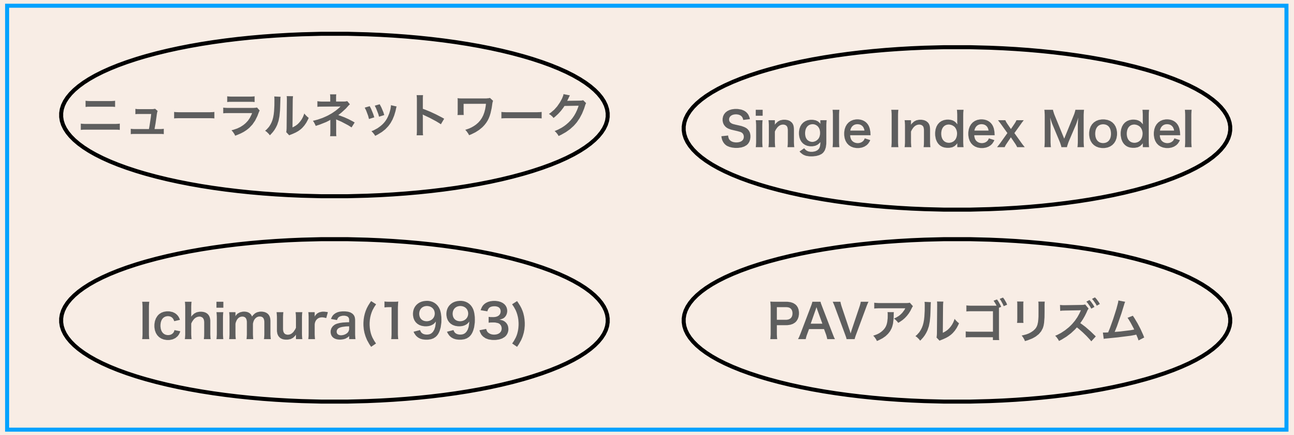


新しい関数

統計の世界の話題を応用することで、形状を固定しない方が精度が上がり、問題に応じて活性化関数を使い分けなくて良い？

提案手法

統計学の知識を組み合わせながら、ニューラルネットワークに使える新しい活性化関数を考える



新しいアルゴリズムの提唱

K-AF（Kernel Activation Function）

提案手法2

K-AFの式

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

$\mathbf{X}^{calc}$ …データセットの一部を表し、 $\mathbf{X}^{calc} \subset \mathscr{D}_x$

$\mathbf{Y}^{calc}$ …データセットの一部を表し、 $\mathbf{Y}^{calc} \subset \mathscr{D}_y$

K …カーネル関数（本実験ではガウス関数）

$\mathbf{W}$ …重み

h_{calc} …バンド幅

提案手法3

K-AFのアルゴリズム

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

1. データセット Dx,Dy から任意個数の Xcalc と Ycalc から取り出す。
2. 現状のパラメータ Wt, Dx, Dy を用いて、リンク関数 gt のカリー化を行う。
3. そのリンク関数を用いて Ytrue の値との最小二乗誤差を取り Wt+1 を求める

実行環境

本研究の３の実験

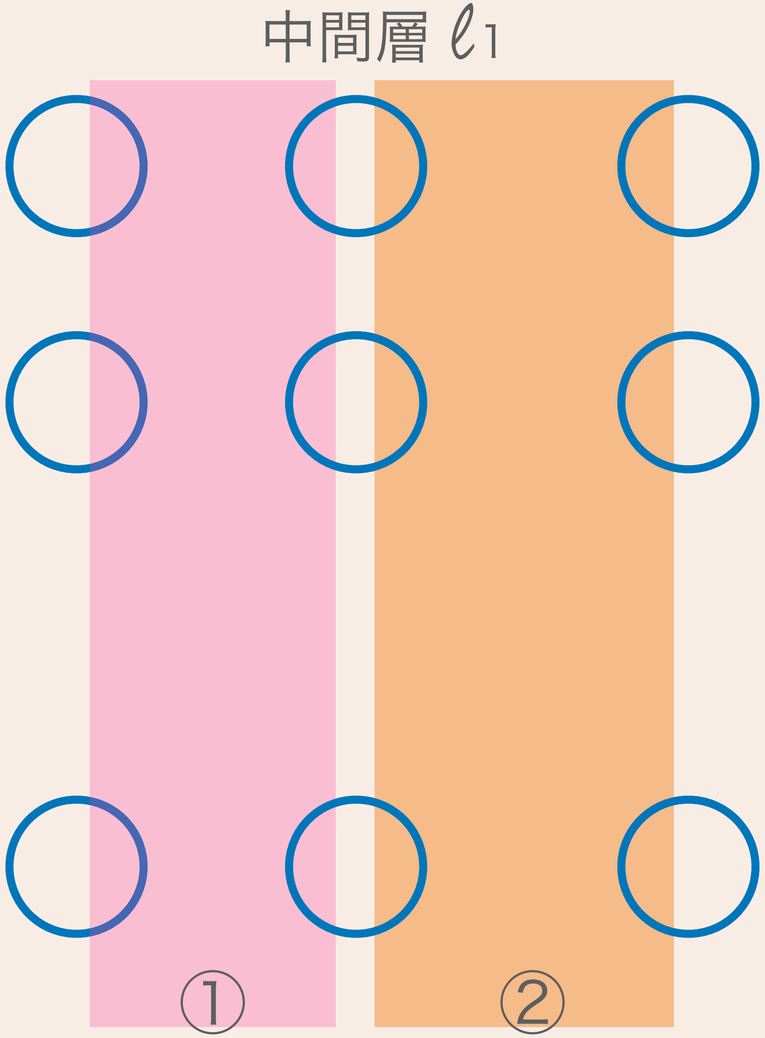
1. K-AFの精度の評価
2. K-AFの可視化による、関数の可視化による、既存の関数の妥当性の評価
3. K-AFが勾配発散しないための条件調査

本研究を動かすための実行環境及びライブラリ

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

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

実験環境



図yy

今回のニューラルネットワークは
中間層を一つで実験すし、出力層にK-AFを配置し
実験1~3を評価する。

①にはReLUを置く
②にK-AF等の様々な活性化関数を配置る。

実験. 1の概要

K-AFの精度の評価

1. 図yyの②に対して
   ReLU, Sigmoid, Tanh, Swish, Mish, K-AF
   の6つの活性化関数を入れ替え、K-AFの評価をする。

2. 比較用のデータの性質及び設定を表zzに示す。
   実験は全てバッチ学習で性能評価をする。

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

表zz　スライドではdigitsとbostonの性能を確認する。

実験. 1.1　digitsの比較設定

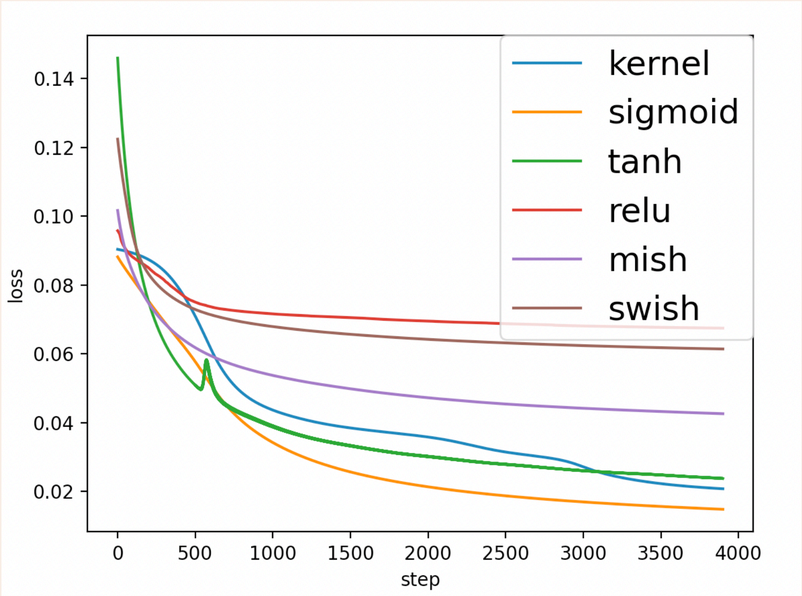
以下の表の二つの設定でdigitsを学習した際のK-AFのAccuracy及びValidationLossを取得する。

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

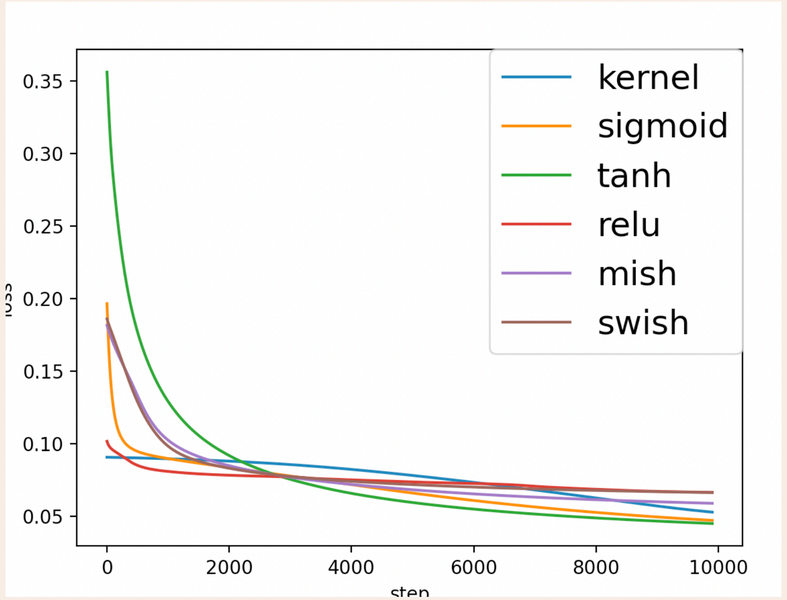
（epoch_numはepoch数, exp_numは実験回数, calc_numは. $\mathbf{X}^{calc}$ の数）

※Accuracyはexp_numの数分の平均値を取得する。

実験. 1.1　digitsの比較結果



設定1のValidationLoss



設定2のValidationLoss

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

AccuracyもLossも既存のSigmoidやTanhなどぐらい性能よく学習できた。

特に学習率が高い場合はいい性能を出している。

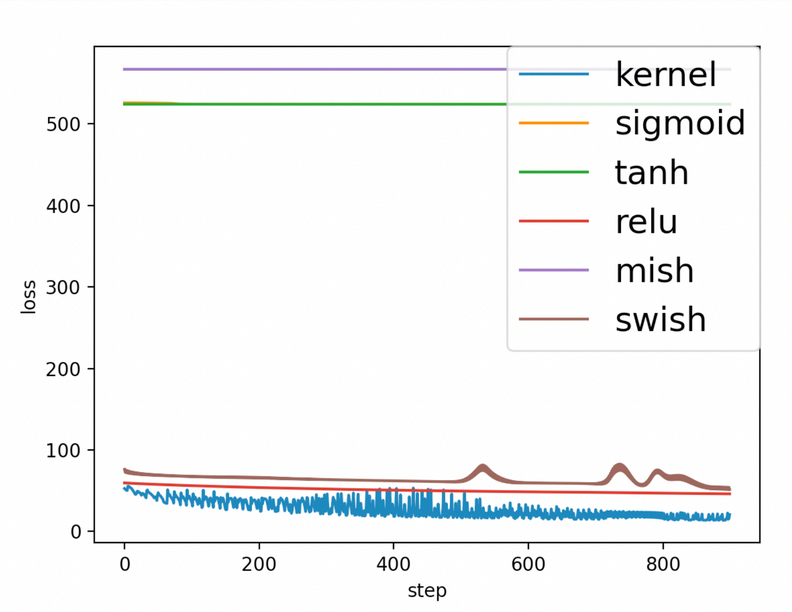
実験. 1.2　bostonの比較設定

以下の表の二つの設定でBostonを学習した際のK-AFのMSE及びValidationLossを取得する。

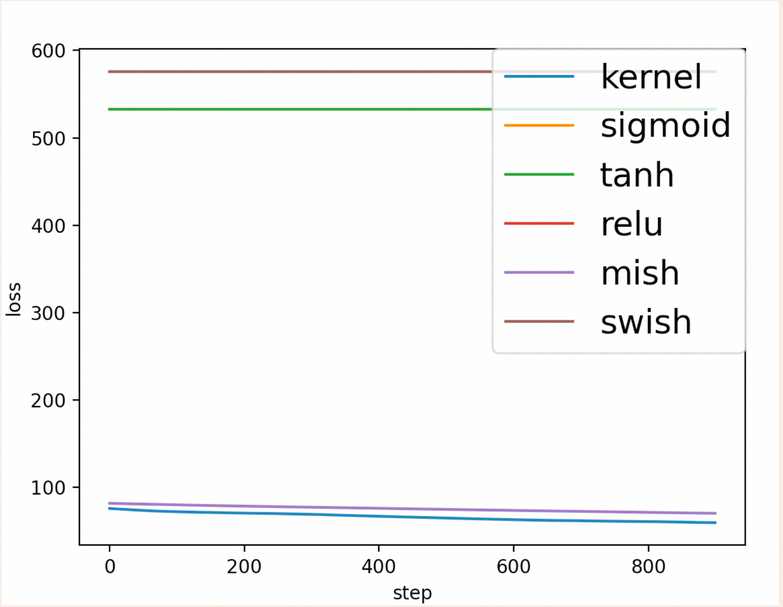
設定名	設定 1	設定 2
LearningRate	10^{-5}	10^{-5}
Initializer	KaimingUniform	Xavier
Optimizer	SGD	Adam
Regularizer	なし	なし
epoch_num	1000	1000
exp_num	5	5
calc_num	26	26

※Bostonは回帰問題なので、MSEで性能評価を行う事に注意

実験. 1.2　bostonの比較結果



設定1のValidationLoss



設定2のValidationLoss

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

Bostonは正確に推測するためには決定機など十分な複雑性を持ったモデルが必要なため、K-AFでは非常に高い性能を出した。

MishやSwishでも、回帰問題のためいい性能を出している。

実験. 2の概要

K-AFの可視化による、関数の可視化による、既存の関数の妥当性の評価

1. 推論した活性化関数の形状を調査し、既存の活性化関数との違いを比べる

2. 定量評価を行うために、以下の表項目を用い調査をする。

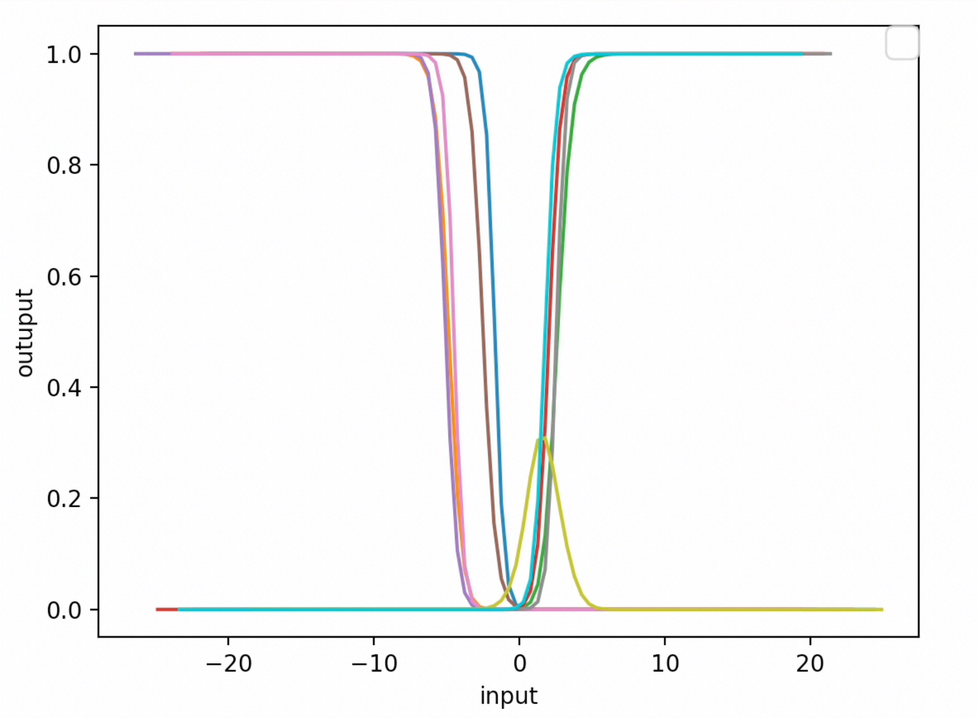
単調増加関数か	上限値があるか
○ or ×	○ or ×

3. 各データセットにおいて以下の表項目の設定で学習を行う。

データセット名	LearningRate	Optimizer	Initializer	Reguralizer	中間層の数	calc_num	epoch_num
iris	10^{-1}	SGD	KaimingUniform	なし	4	30	1000
digits	10^{-1}	SGD	KaimingUniform	なし	100	36	1000
wine	10^{-2}	SGD	KaimingUniform	なし	40	36	1000
boston	10^{-5}	SGD	KaimingUniform	なし	20	26	1000
breast_cancer	10^{-2}	SGD	KaimingUniform	なし	30	285	1000

※実験1同様にスライドではdigitsとbostonの場合のみ観測する。

実験. 2.1 digitsの場合(画像)



digitsの学習で生成されたK-AF

単調増加関数か	上限値があるか
▲	○

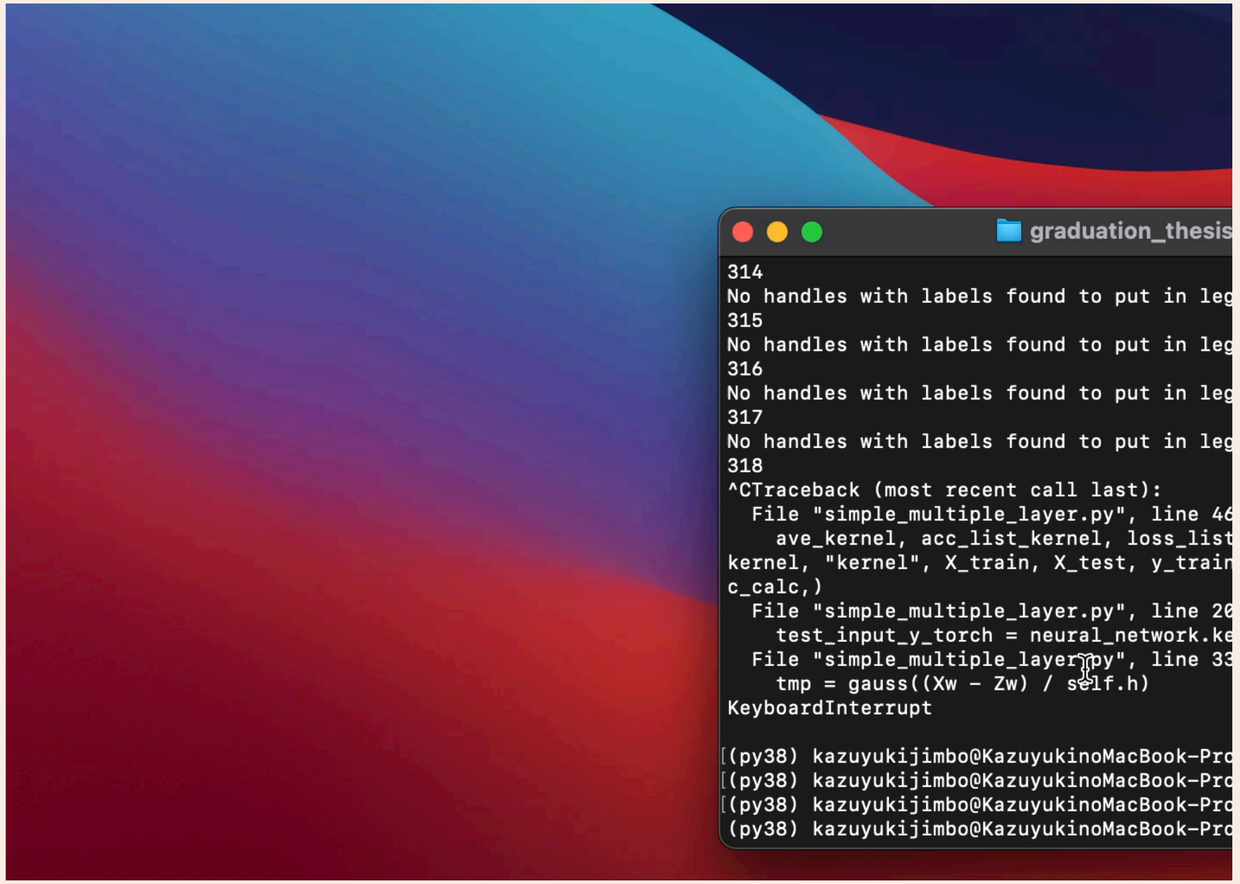
ほぼ全てのK-AF単調増加関数もしくは単調減少関数となった。

ラベリング問題なので上限値が1.0になるのも想定通りである。

Sigmoidに等に性質や形が似ているので、このデータセットではSigmoidでも十分な性能持っているとも言える。

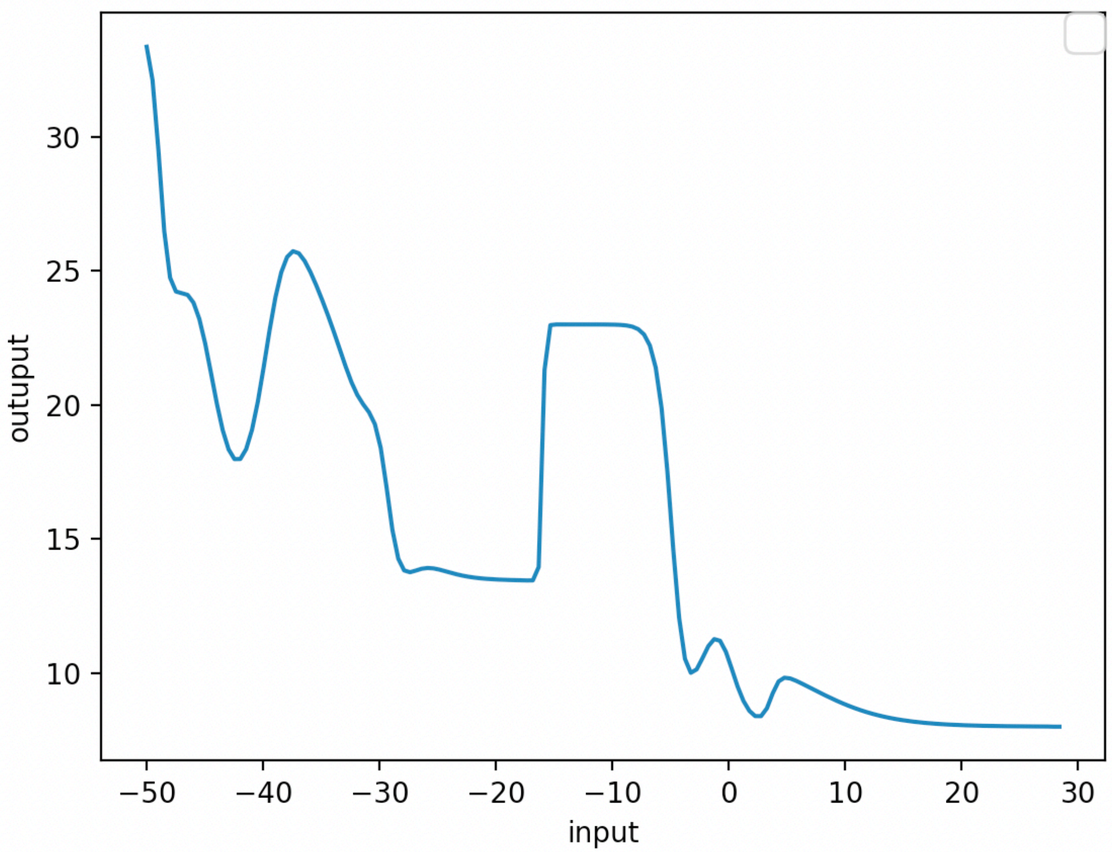
実験. 2.1 digitsの場合(動画)

digitsのK-AFの学習過程の可視化



https://github.com/latte0/RG-Thesis-Template/blob/main/asset/digits.mov?raw=true

実験. 2.2 bostonの場合(画像)



bostonの学習で生成されたK-AF

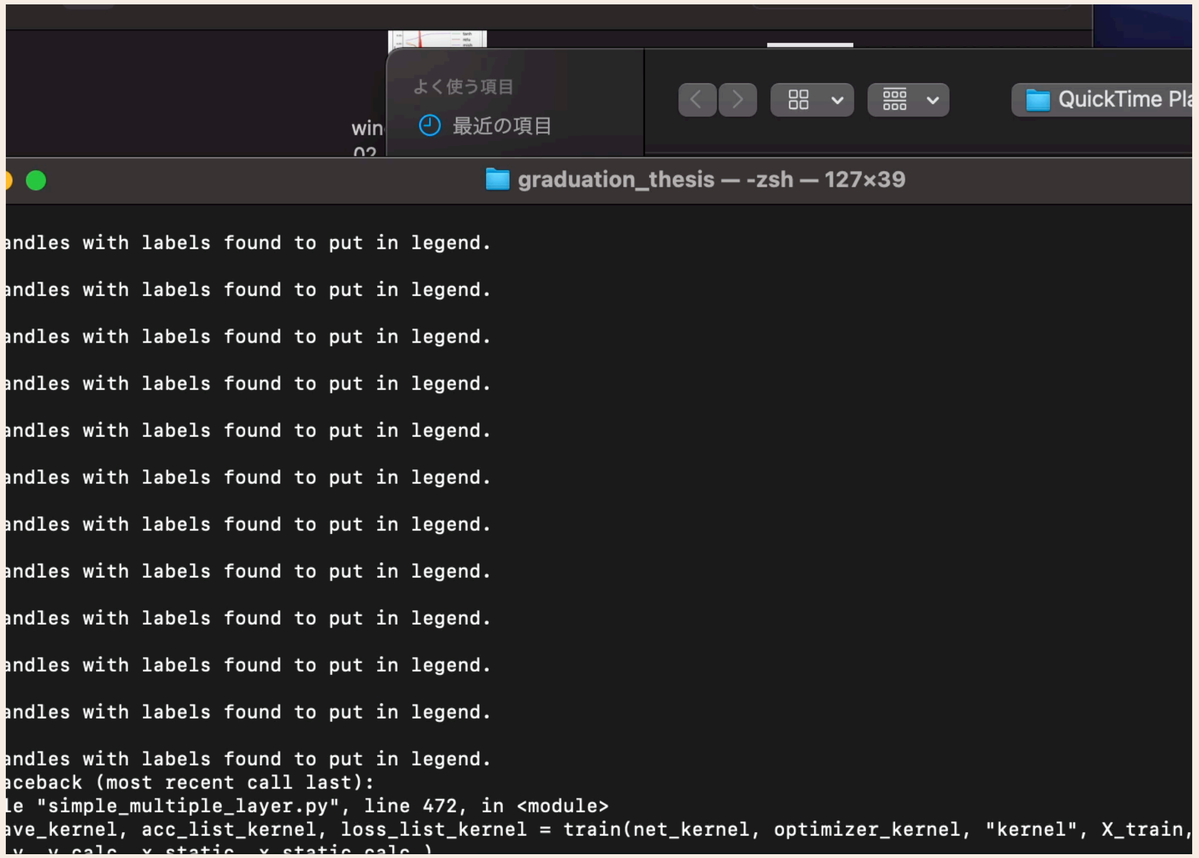
単調増加関数か	上限値があるか
×	▲

歪な形をしていて、単調増加なReLUやSigmoidでは表現のとして不十分であることが言える。

回帰問題であるので、上限値がないような関数が適切であったと言える。

実験. 2.2 bostonの場合(動画)

BostonのK-AFの学習過程の可視化



https://github.com/latte0/RG-Thesis-Template/blob/main/asset/boston.mov?raw=true

実験. 3の概要

K-AFが勾配発散しないための条件調査

1. 以下の表の変更リストの全ての組み合わせで勾配爆発がどの程度起こるかテストをする。

ニューラルネットワークの構成	変更のリスト
Initializer	[Xavier, KaimingUniform]
Regularizer	[何もなし, L1 ノルム, L2 ノルム]
Optimizer	[SGD, RMSProp, AdaGrad, Adam]

2. データセットは**boston**を用い、その他のパラメータを以下に設定する。

ハイパーパラメータ	値
LearningRate	10^{-5}
中間層の次元数 l_1	20
epoch_num	5
exp_num	1000
calc_num	50

※論文中のクリッピングの実験は特に効果がなかったのでスライドでは省く。

実験. 3の結果

全てのパターンの中でも最も勾配爆発確率が低かったトップ5を以下に示す。

順位	Initializer	Optimizer	Reguralizer	勾配爆発回数	勾配爆発確率
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%

1. XavierはKaimingUniformの2倍程度、勾配爆発の確率が低かった。

2. OptimzerとしてAdamが上位に来た。

実験結果まとめ

1. 一部の条件で、回帰と分類どちらでもいい精度の活性化関数をK-AFで導くことができた。

2. K-AFの形状の調査により、既存の活性化関数は問題に対して適切とは言えない場合があることが判明した。

3. K-AFは初期値に依存して勾配爆発が防げることがわかった。

解決した課題

1. 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き、 高い精度を出せるようにした。

2. 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。

3. 分類や回帰といった問題を考えずとも、高い精度でニューラルネットワークが構築できるようになった。

本研究の問題点と展望

勾配の発散問題

データセットが複雑な場合、勾配が発散するのが一定の確率で防げてない

ディープラーニングへの応用

出力層のみでなく、中間層の活性化関数も推論できる形を
見つける必要がある。

参考文献

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

[2] 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.

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

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

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

[6] scikit-learn Machine Learning in Python, https://scikit-learn.org/stable/