

# 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

# 本研究で用いる記号と用語

## 用語

**K-AF** … 本研究で発見したカーネル密度推定を用いた新しい活性化関数の名称

## 記号

## 本研究のモチベーション

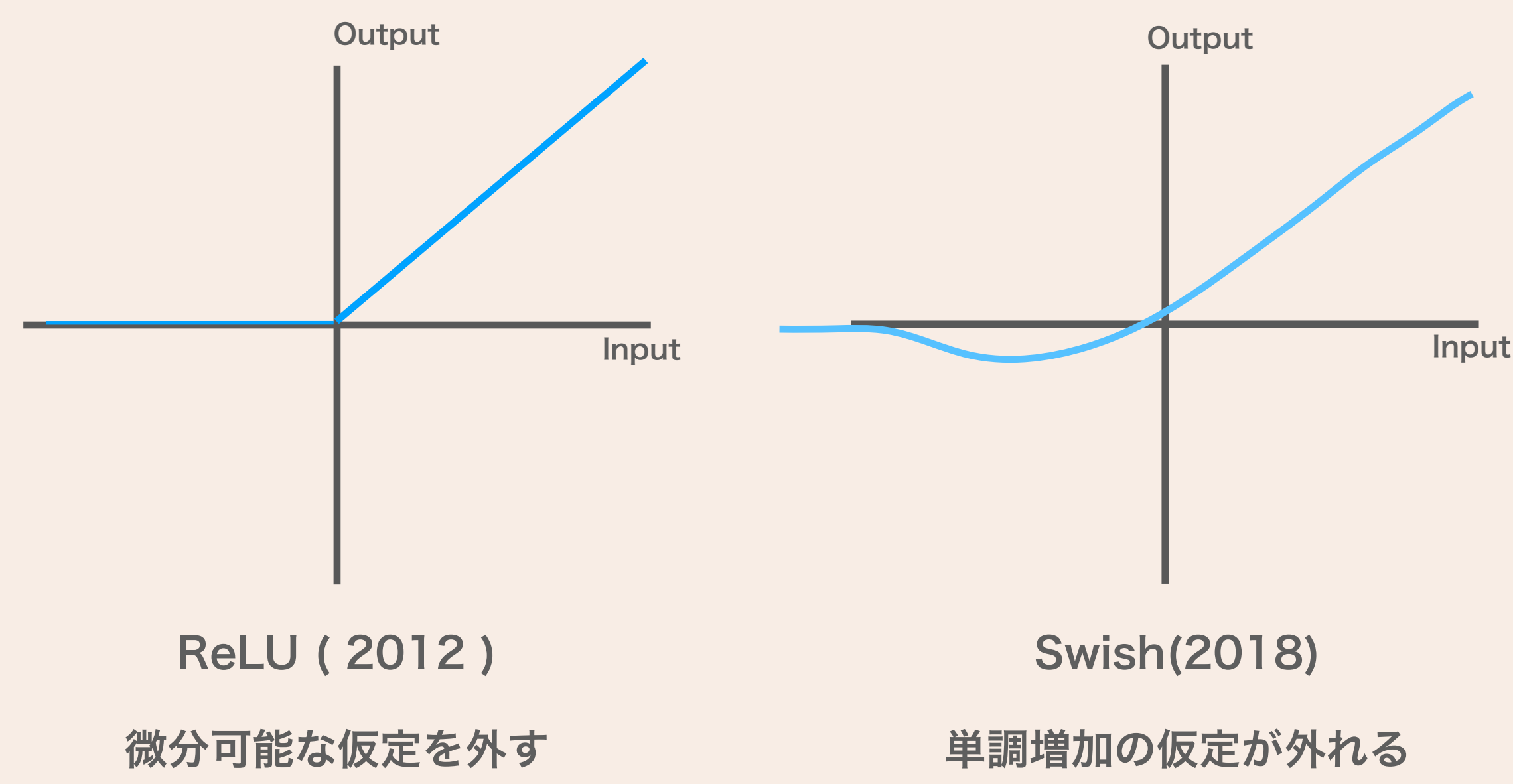
1. ニューラルネットワークにおいてより精度が高い活性化関数を見つ  
きたい。
2. 既存の活性化関数が、さまざまな問題において最適な形か調査した  
い。
3. 最適な活性化関数の選択コストを無くし、機械学習を容易にした  
い。

## 本研究が解決した課題

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. 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。

既存の活性化関数の種類と形状

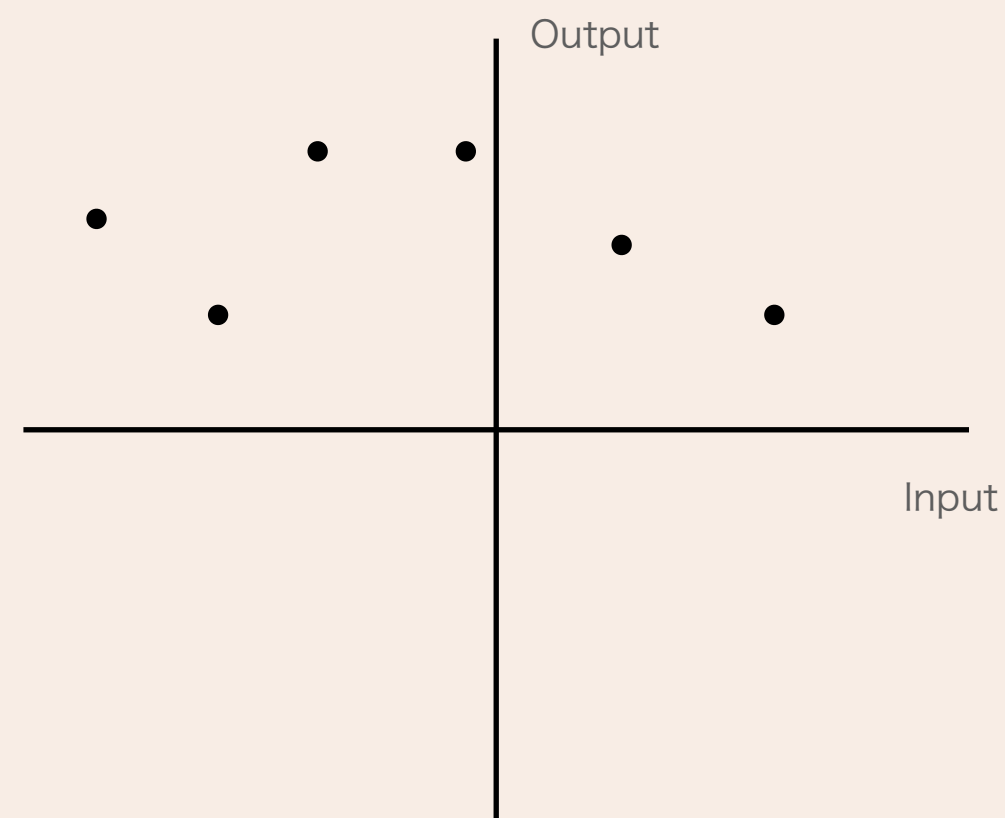
Sigmoid	$\frac{1}{1 + \exp(x)}$
Tanh	$\tanh(x)$
ReLU	$\text{output} = \begin{cases} 0 & \text{when } x < 0 \\ x & \text{when } x \geq 0 \text{ else} \end{cases}$
Swish	$x \cdot \text{sigmoid}(\beta x)$
Mish	$x \cdot \tanh(\log(1 + \exp(x)))$



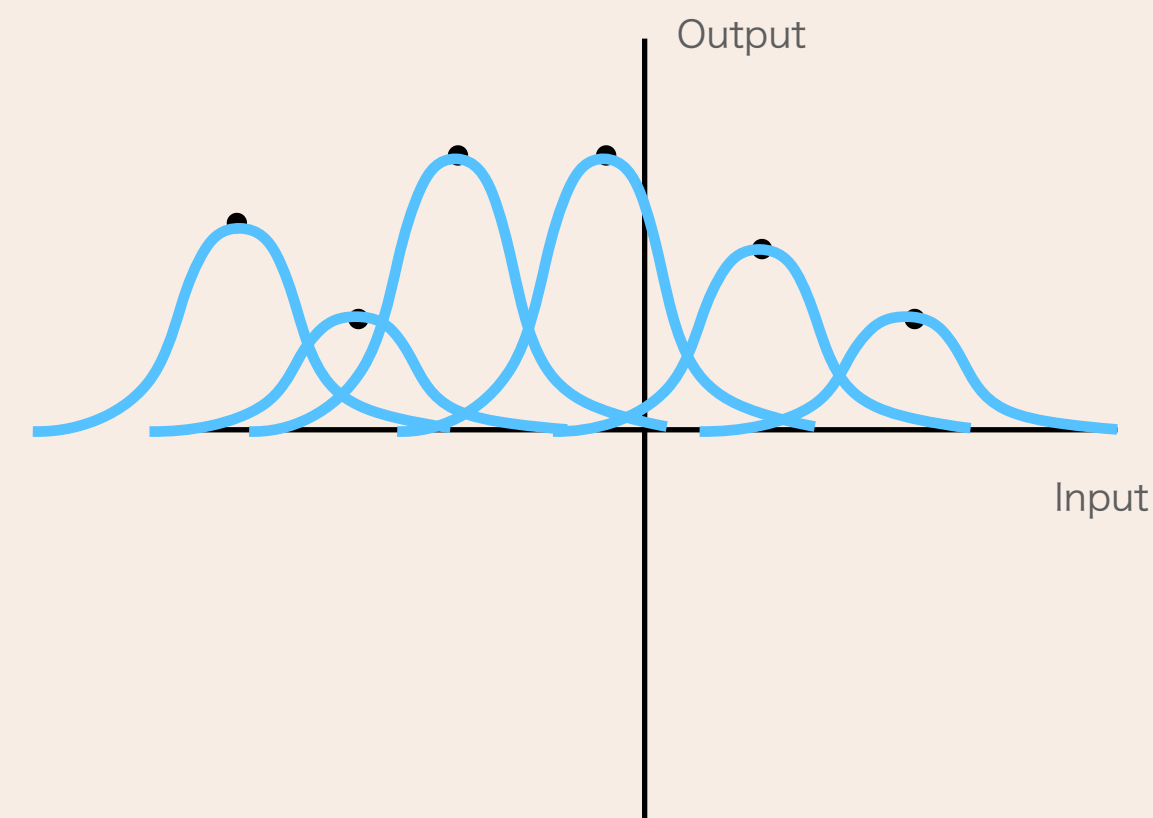
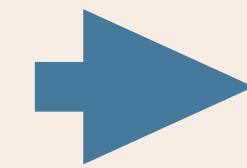
関数の形状が単調増加のシンプルなものから、複雑な形状の方が精度が上がる。

## カーネル密度推定

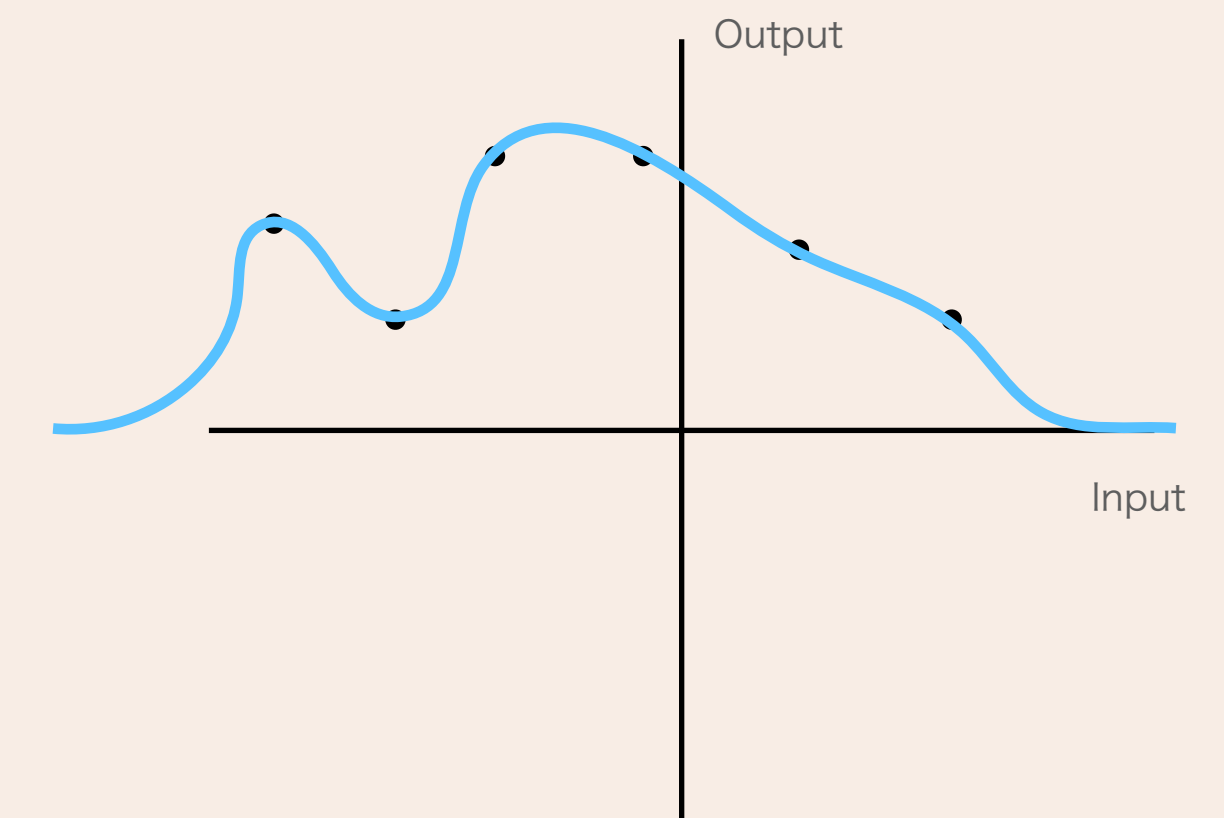
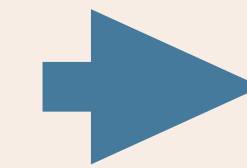
ノンパラメトリックモデルの一つ。データ点をの周囲にカーネル関数を置き、関数の近似を行う。



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

## SingleIndexModel

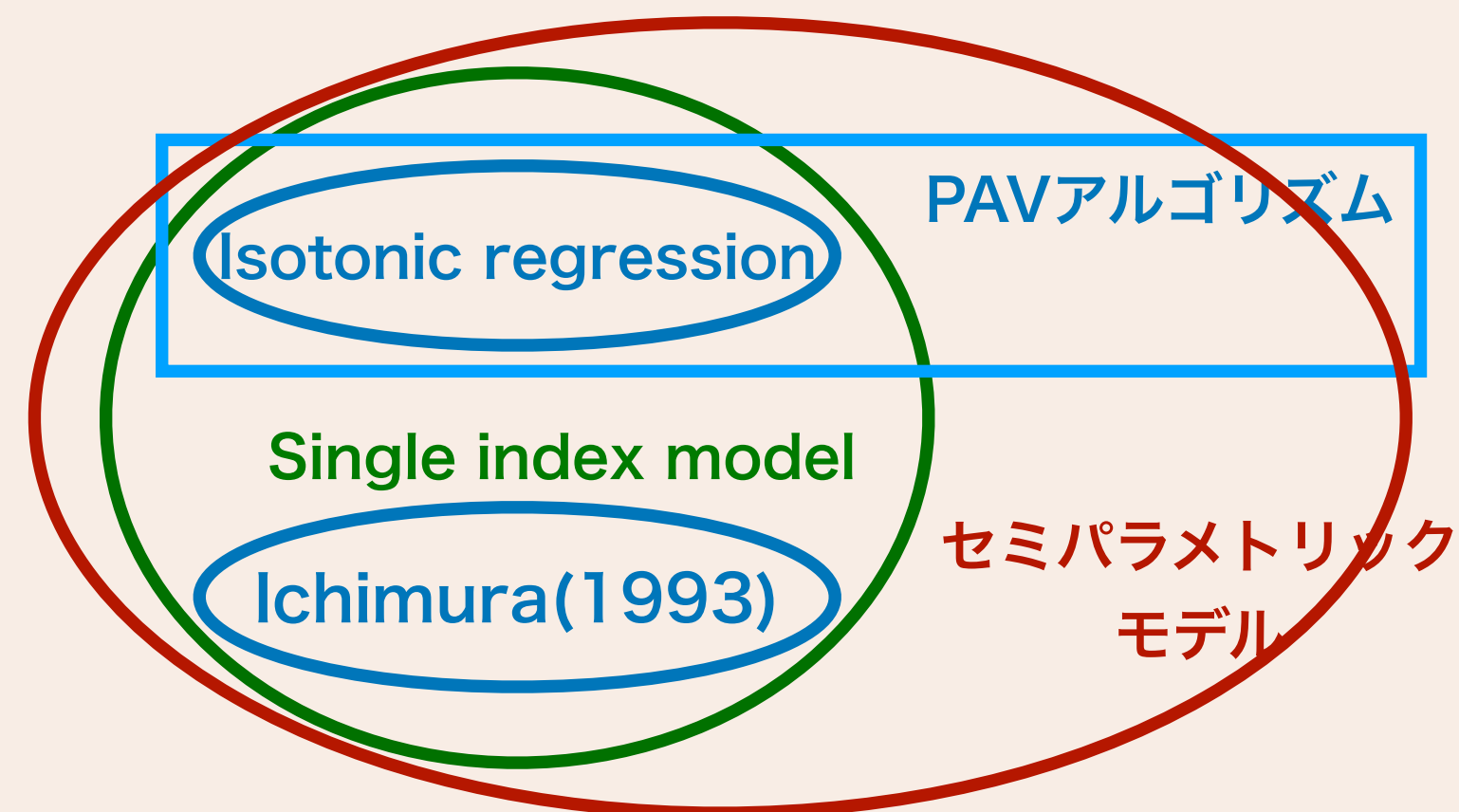
ノンパラメトリックとパラメトリックな手法を組み合わせたモデル（セミパラメトリックモデル）の一つ

- 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

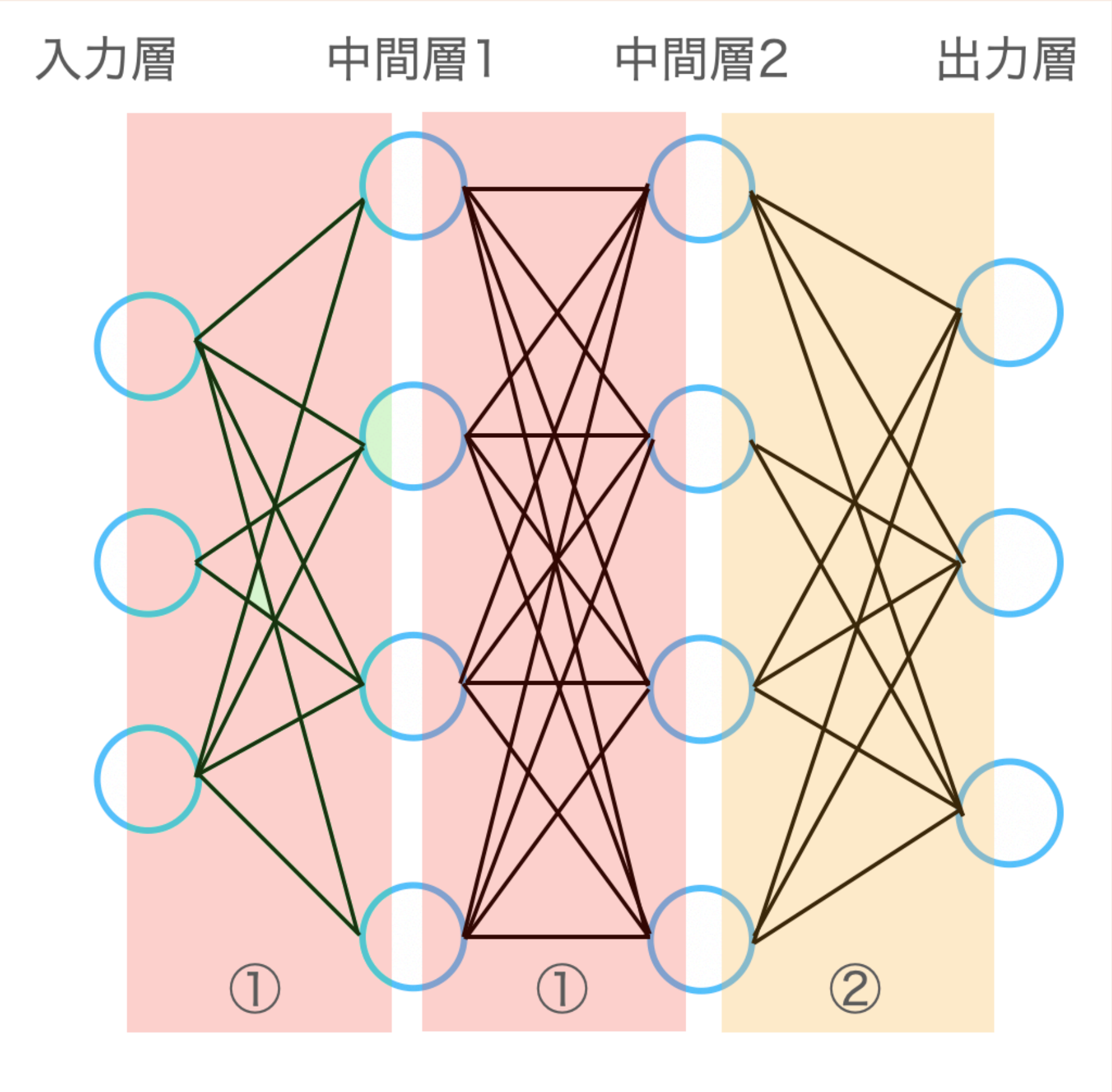


各用語の立ち位置

全て統計学の世界で発見されてきたモデルである。



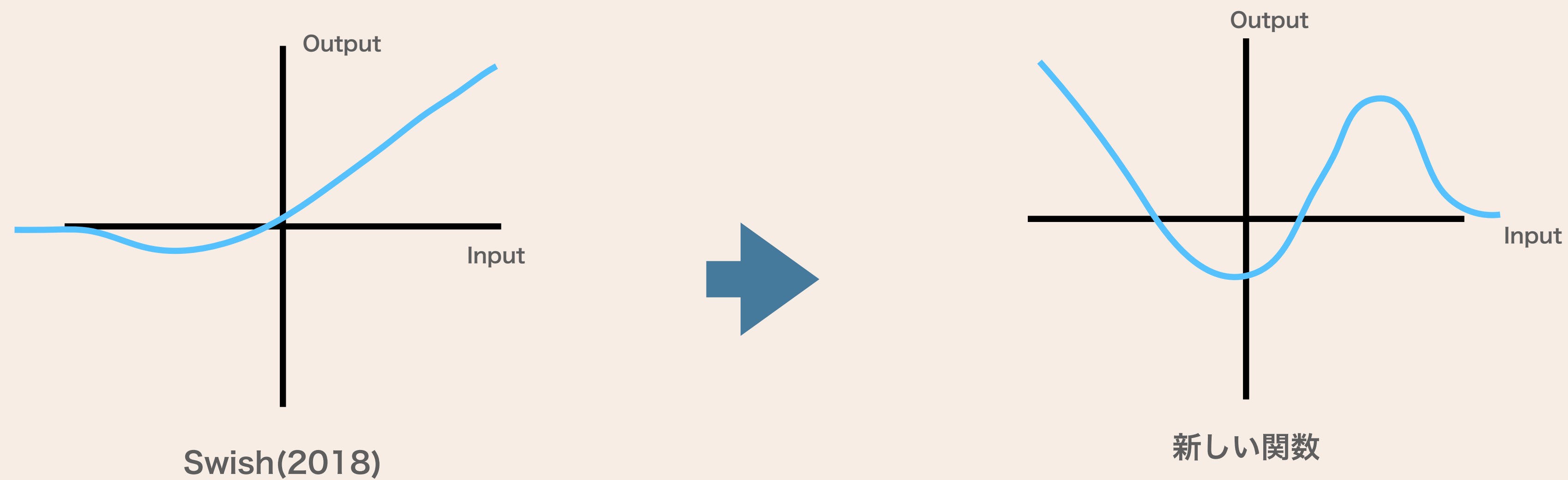
ニューラルネットにおける活性化関数の構成



「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

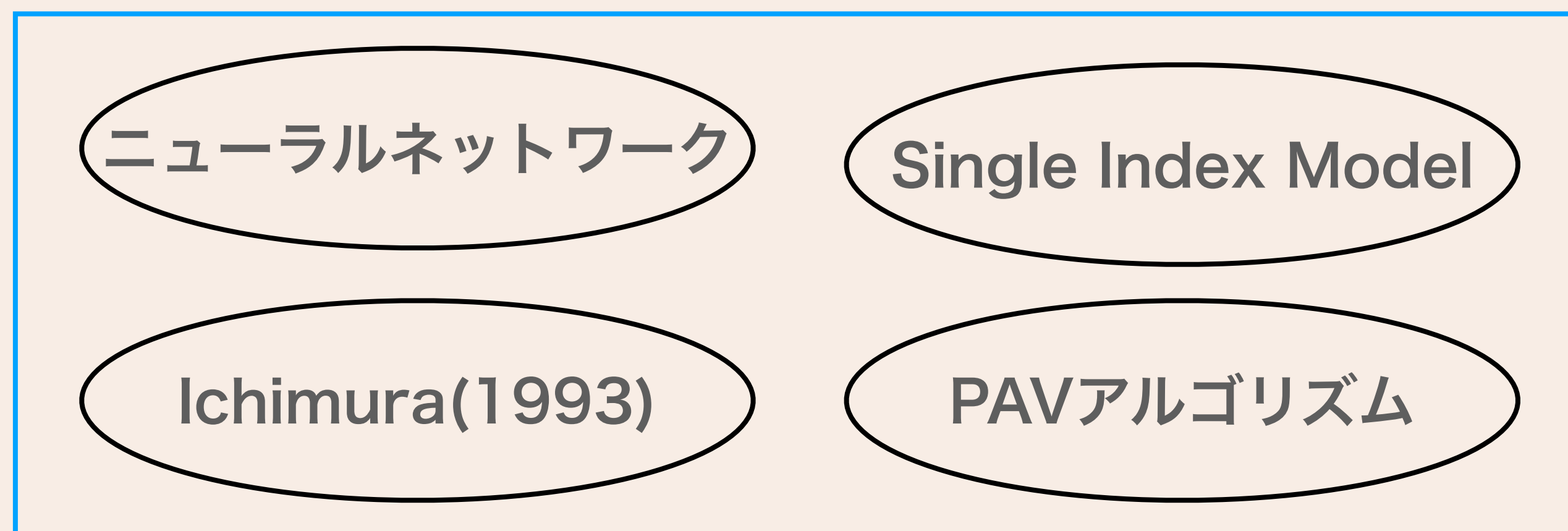




統計の世界の話題を応用し、形状を固定  
しない方が精度が上がる？

# 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)**

## 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、 $\mathbf{X}^{calc} \subset \mathcal{D}_x$   
 $\mathbf{Y}^{calc}$  ...データセットの一部を表し、 $\mathbf{Y}^{calc} \subset \mathcal{D}_y$

$K$  ...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

## K-AFのアルゴリズム

### Algorithm 1 K-AF

**Input:** data  $\langle (\mathbf{X}_i, \mathbf{Y}_i) \rangle_{i=1}^m \in \mathbb{R}^d \times \mathbb{R}^e$ ,  $G : (\mathbb{R}^d, \mathbb{R}^{d \times n}, \mathbb{R}^{e \times n}, ) \rightarrow \mathbb{R}^e$ .

$\mathbf{X}^{calc} \sim_n \mathcal{D}_x$ ;

$\mathbf{Y}^{calc} \sim_n \mathcal{D}_y$ ;

**for**  $t = 1, 2, \dots$  **do**

$g^t := \tilde{G}(\mathbf{W}^t, \mathbf{X}^{calc}, \mathbf{Y}^{calc})$ ;

$\min_w E(w) = \frac{1}{2} \sum_m (\mathbf{Y}^{true} - g^t(\mathbf{X}_i))^2$

**end for**

1. データセット  $\mathcal{D}_x, \mathcal{D}_y$  から任意個数の  $\mathbf{X}^{calc}$  と  $\mathbf{Y}^{calc}$  から取り出す。
2. 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  $g^t$  のカーリー化を行う。
3. 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

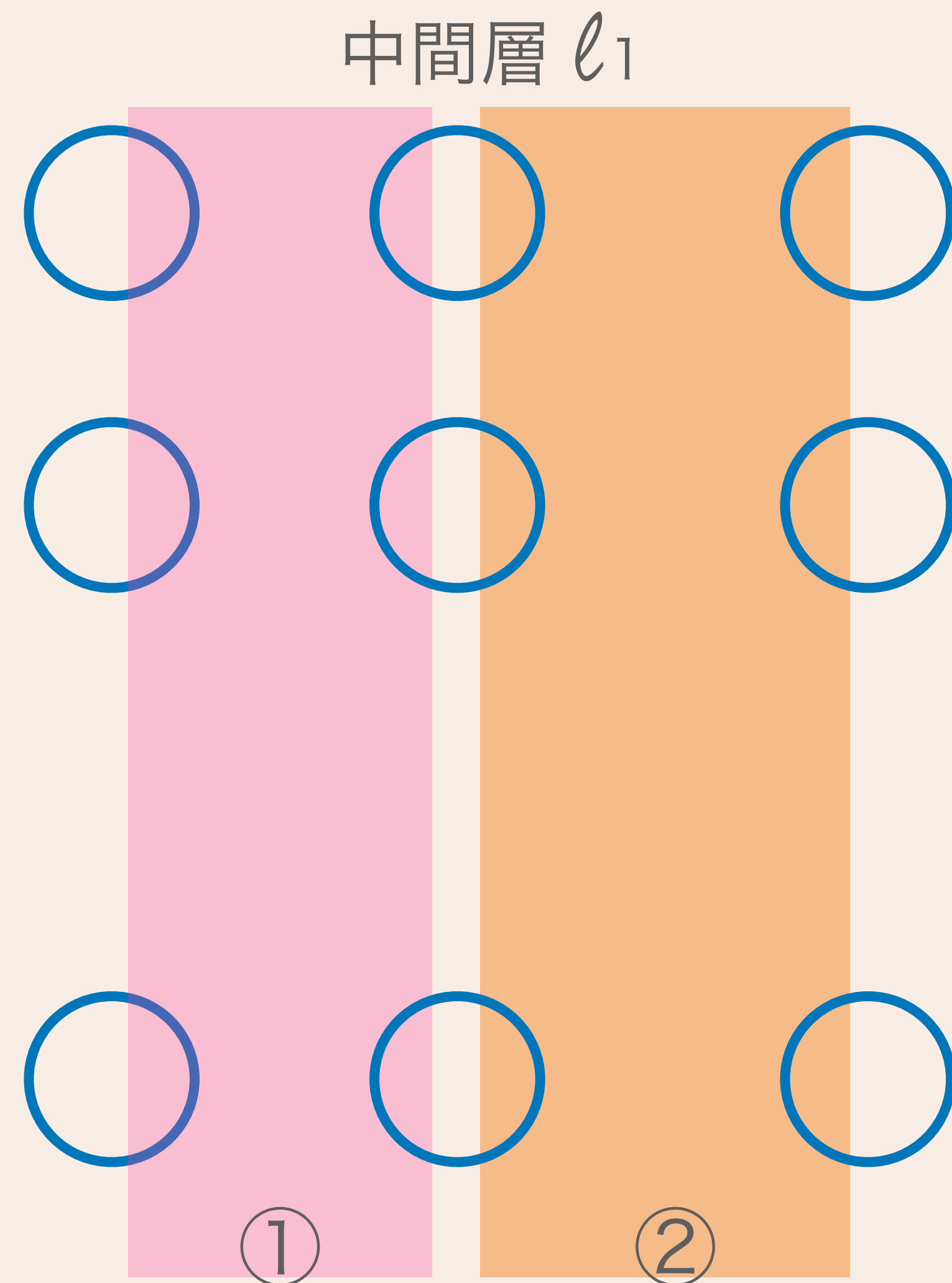
## 本研究の 3 の実験

- 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



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

- 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

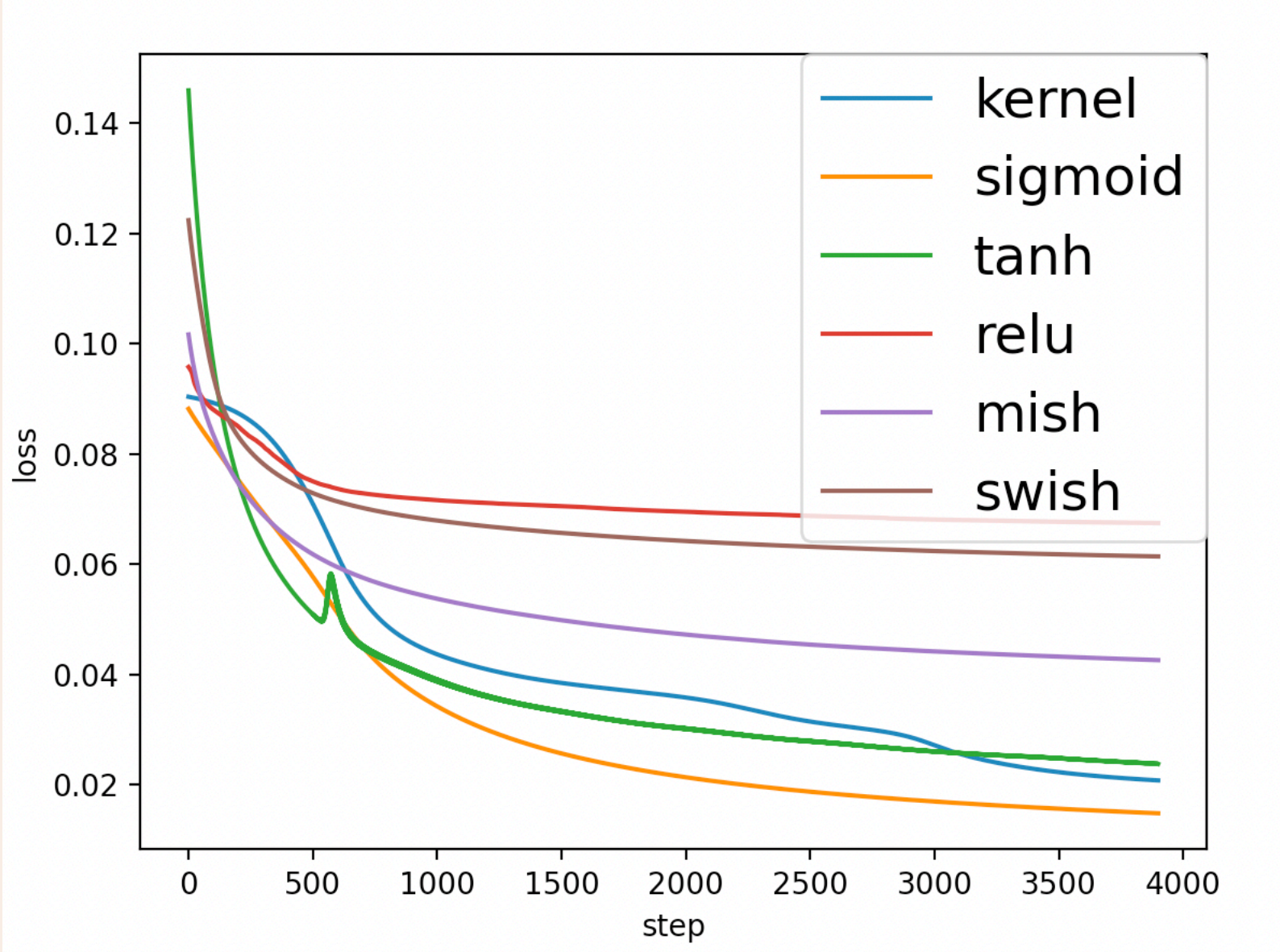
以下の表の二つの設定でdigitsを学習した際のK-AFのAccuracy及びValidationLossを取得する。

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

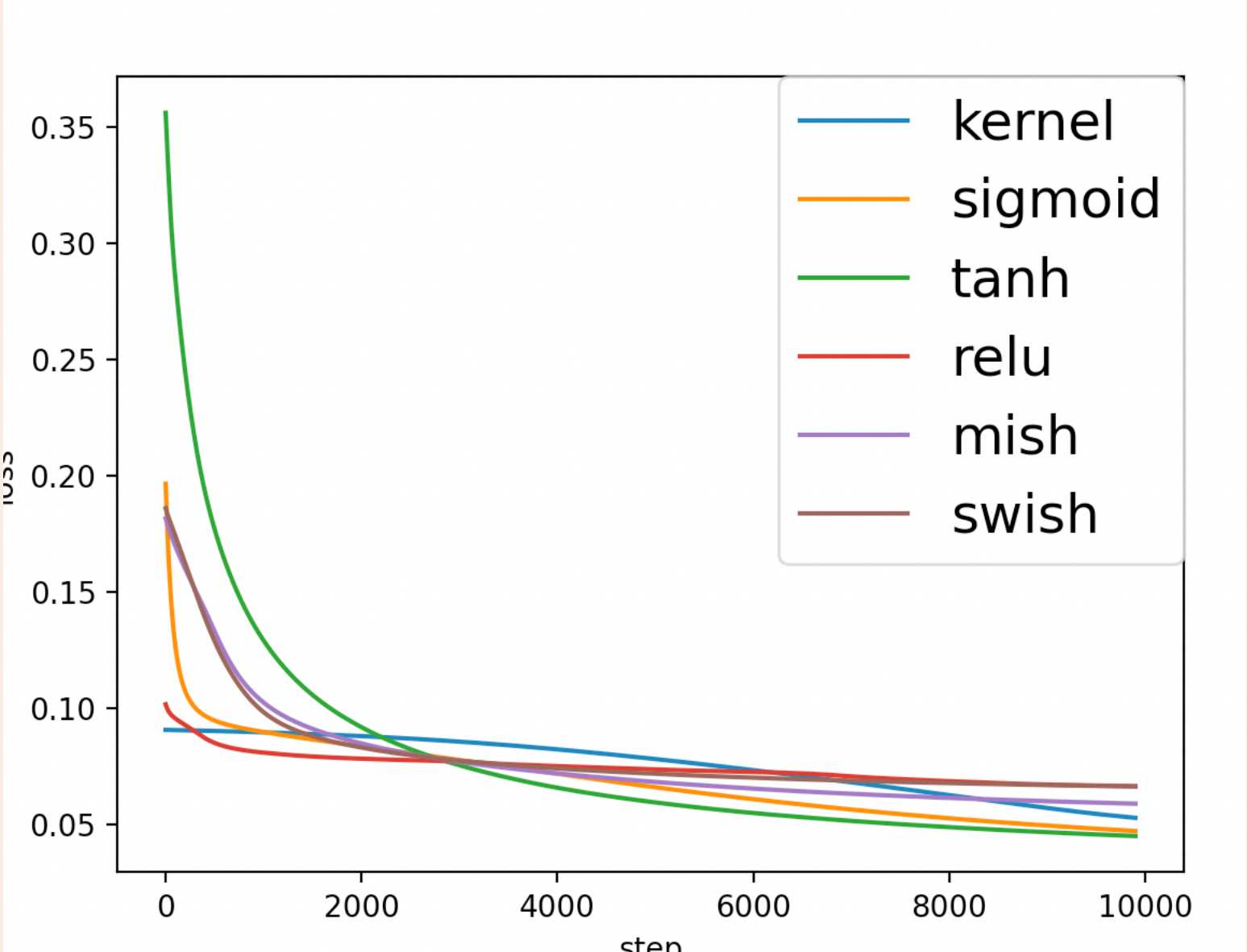
(epoch\_numはepoch数, exp\_numは実験回数, calc\_numは.  $\mathbf{X}^{calc}$  の数)

※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。



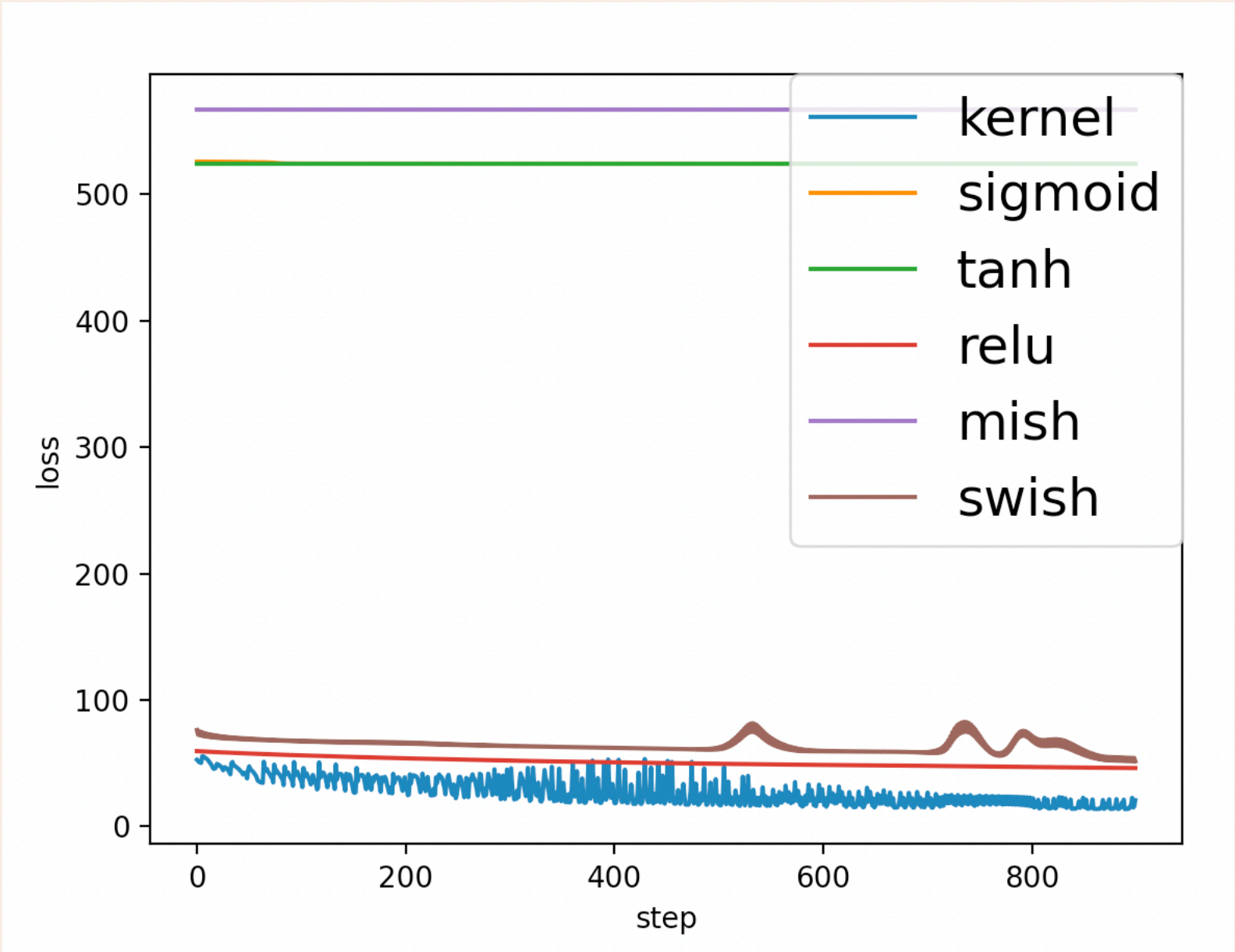
# 実験. 1.2    bostonの比較設定

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。

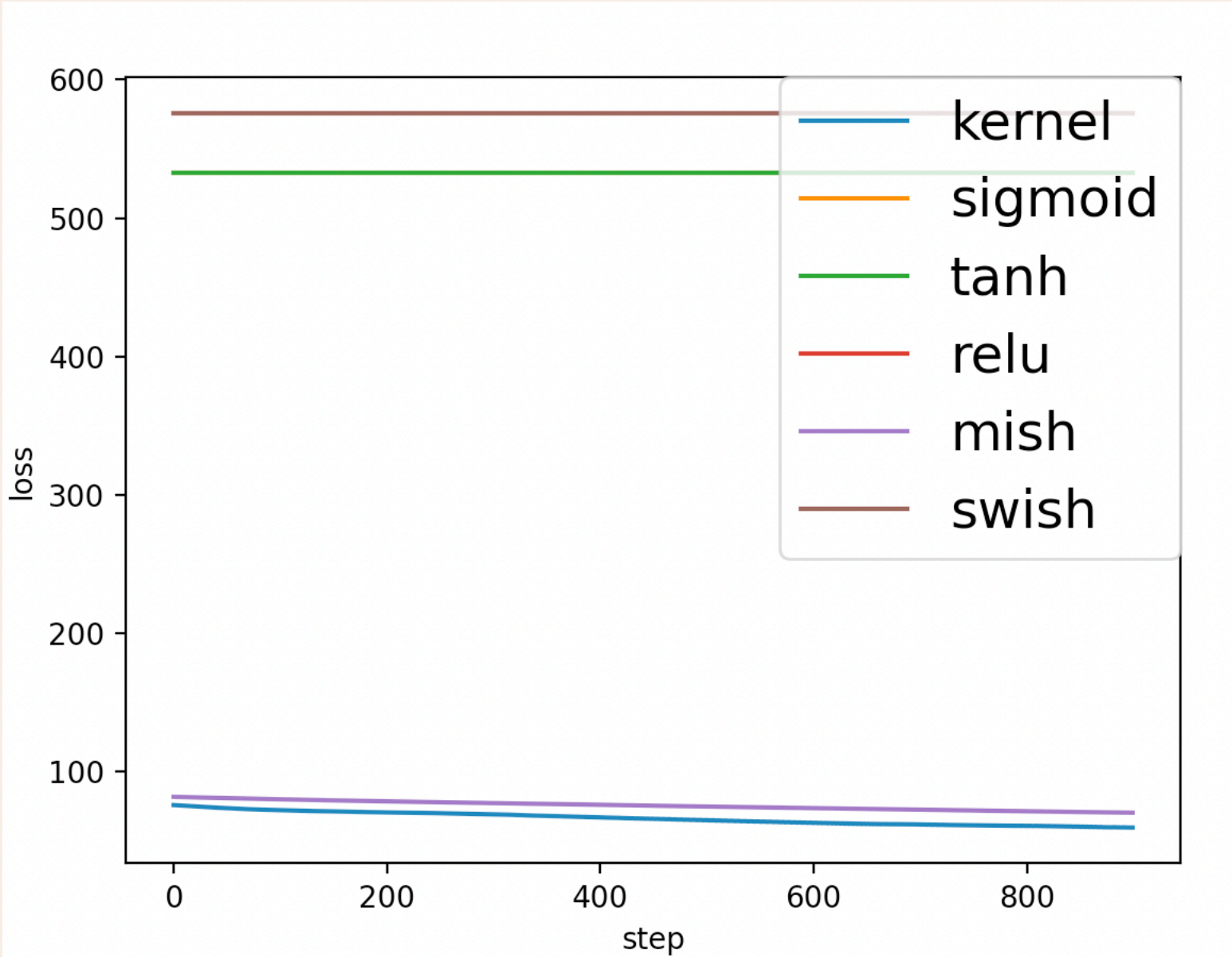
設定名	設定 1	設定 2
LearningRate	$10^{-5}$	$10^{-5}$
Initializer	KaimingUniform	Xavier
Optimizer	SGD	Adam
Regularizer	なし	なし
epoch_num	1000	1000
exp_num	5	5
calc_num	26	26

※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

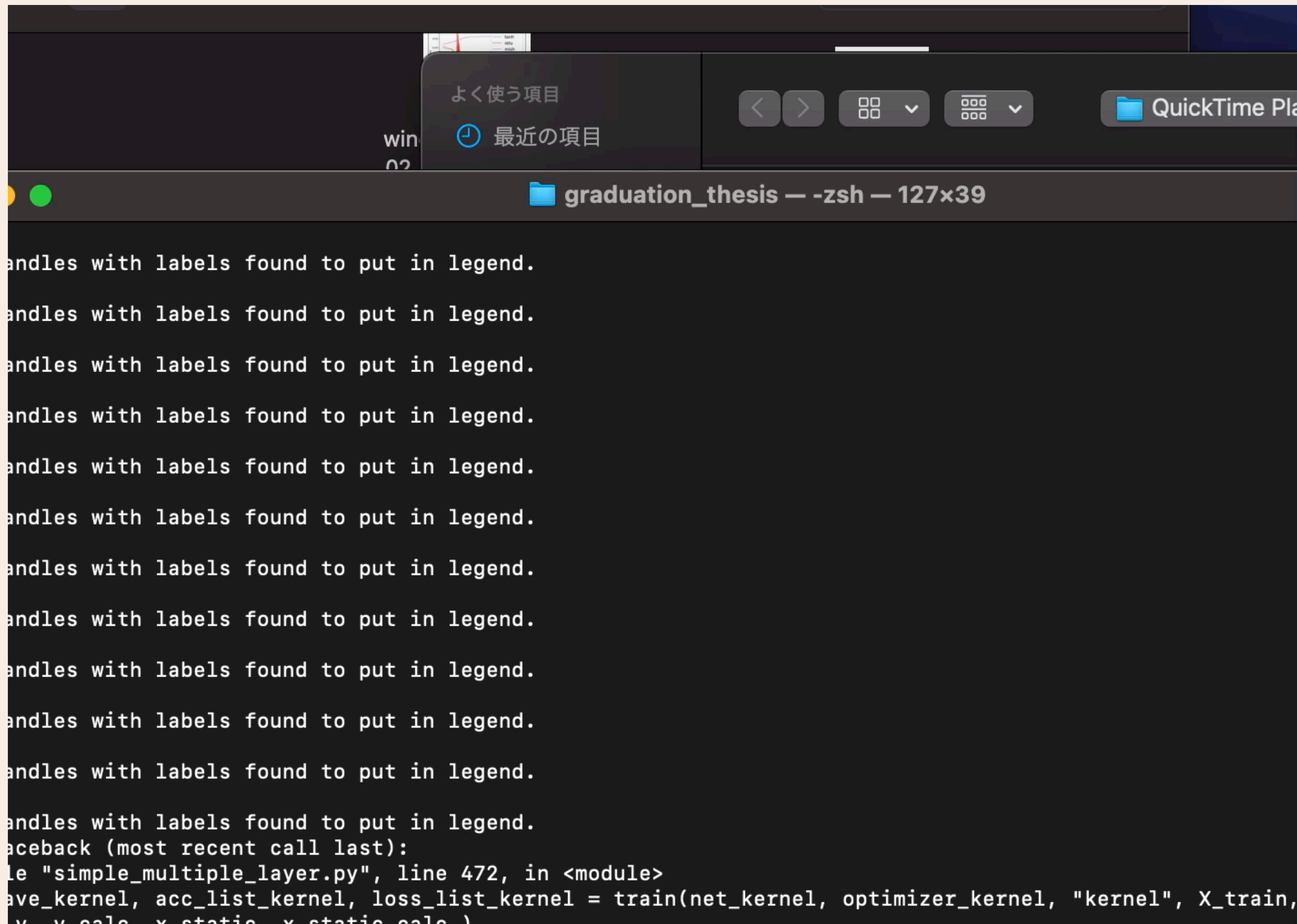
1. 推論した活性化関数の形状を調査し、既存の活性化関数との違いを比べる
2. 定量評価を行うために、以下の表項目を用い調査をする。

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

※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 bostonの場合



```
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
handles with labels found to put in legend.
Traceback (most recent call last):
  File "simple_multiple_layer.py", line 472, in <module>
    ave_kernel, acc_list_kernel, loss_list_kernel = train(net_kernel, optimizer_kernel, "kernel", X_train,
    y_train, scale, static_scale)
```

# 実験. 3の概要



# 実験. 1

**K-AF**

# 実験. 1

## K-AF

## K-AF

# 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性能を出すことができた。
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. 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。

# 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。

# 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。

- [1] Hidehiko Ichimura, Wolfgang Hardle, and Peter Hall. Optimal smoothing in single index model. [https://projecteuclid.org/download/pdf\\_1/euclid.aos/1176349020](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/>