

**本チュートリアルのHP→**

**<https://bit.ly/2sdNEJ4>**

**最新版のスライドもHPにおいています。**

**スライドのパスワードは**

**deep-com**

**です。**

# 無線物理層における データ駆動アプローチ

—深層学習への期待と展望—

和田山 正

名古屋工業大学

# 深層学習の「深化と広がり」





## PROMISES AND CHALLENGES OF MACHINE LEARNING IN COMMUNICATION NETWORKS (ML4COM)

### PROMISES AND CHALLENGES OF MACHINE LEARNING IN COMMUNICATION NETWORKS (ML4COM)

# Workshop on Promises and Challenges of Machine Learning in Communication Networks (ML4COM)

#### Workshop Co-Chairs

- Paul de Kerret, EURECOM, [paul.dekerret@eurecom.fr](mailto:paul.dekerret@eurecom.fr)
- Deniz Gündüz, Imperial College London, [d.gunduz@imperial.ac.uk](mailto:d.gunduz@imperial.ac.uk)
- David Gesbert, EURECOM, [david.gesbert@eurecom.fr](mailto:david.gesbert@eurecom.fr)

#### Scope of the Workshop

Machine learning is among the most active research fields today, and much can be expected from its successful application to communication networks. Yet, transforming this expectation into reality requires significant research efforts and interactions across the communications and

# IEEE COMSOCにおけるDL+Comの動き

- ICC2018, Globecom2018 における機械学習関連のセッションとワークショップの盛り上がり
- COMSOC Emerging Technology Initiative ``Machine Learning for Communications'' 発足



**6<sup>th</sup> IEEE Global Conference on  
Signal and Information Processing**

**<http://2018.ieeeglobalsip.org/>**

**Anaheim, CA USA  
November 26-28, 2018**

## **Symposium on Design, Implementation and Optimization of Deep Learning for Wireless Communications**

**General Chairs:**

Chuan Zhang, *Southeast University*

Yeong-Luh Ueng, *National Tsing Huang University*

**Technical Program Chairs:**

Yair Be'ery, *Tel Aviv University*

Christoph Studer, *Cornell University*

Warren J. Gross, *McGill University*

Witnessing its success in fields including computer vision, speech recognition, bioinformatics, and so on, researchers are considering deep learning for wireless communications. Preliminary results in channel estimation and baseband processing have shown that deep learning can help to understand the wireless contents and produce results comparable and in some cases superior to classic approaches. Though such initiatives have been named, their design, implementation, and optimization are not complete and ir

# 2019 IEEE International Symposium on Information Theory



The IEEE International Symposium on Information Theory (ISIT) will take place in the center of Paris at an atypical venue and historic site, Maison de la Mutualité, France, from July 7th to 12, 2019. We seek original, unpublished contributions in all areas of information theory, including but not limited to the topics listed below. In addition, papers that broaden the reach of information theory, including emerging fields and novel applications of information theory, are encouraged.

## Topics

Big Data Analytics	Deep Learning for Communication Networks	Network Information Theory Pattern Recognition and Machine Learning
Coding for Communication and Storage	Distributed Storage	Quantum Information and Coding Theory
Coding Theory	Emerging Applications of Information Theory	Shannon Theory
Combinatorics and Information Theory	Information Theory and Statistics	

# 本チュートリアルの構成

- 第1部：通信工学研究者のための深層学習クラッシュコース（前提知識を仮定せず）(約80分)
  - 深層学習の基本
  - 深層学習を学ぶためのリソース
  - Google Colaboratory & PyTorch 紹介
- 休憩（10分～15分）
- 第2部：無線物理層における深層学習
  - 自己符号化器に基づくEnd-to-Endアプローチ
  - データ駆動アプローチに基づく反復アルゴリズムの改善
  - 研究事例の紹介

# 深層学習技術のアウトライン

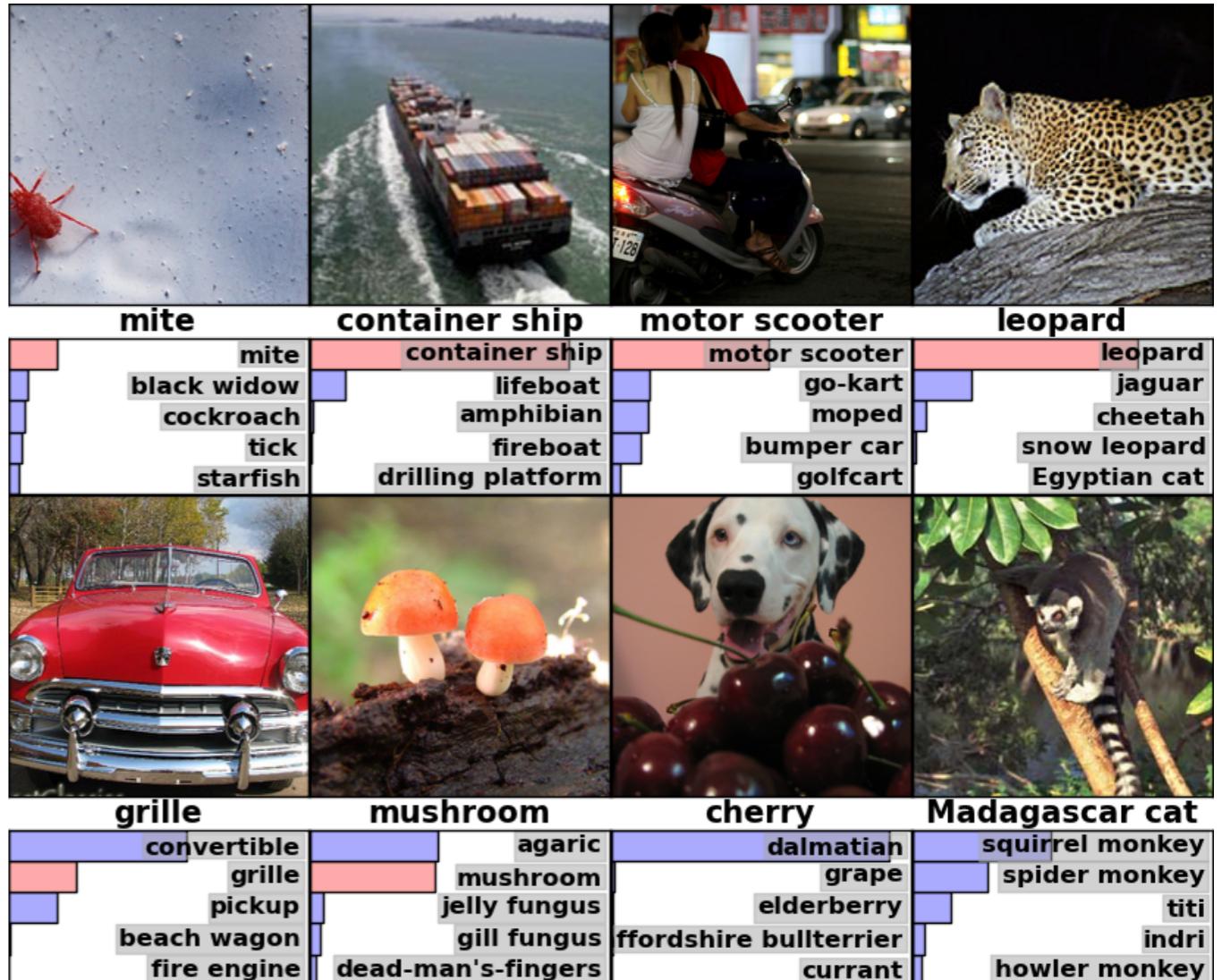
- 深層学習技術の強力さが初期に認識された分野のひとつが画像認識分野
- 個別の各論に入る前に深層学習技術の概要を見ていく

# 深層學習技術の進展

- ・画像認識
- ・音声認識
- ・自然言語処理
- ・機械翻訳

深層學習技術は、これらの分野において特に圧倒的な強みを見せている

ImageNet Classification



cited from: ``ImageNet Classification with Deep Convolutional Neural Networks'', Alex Krizhevsky et al.

# ImageNet

スタンフォード大学が ImageNet を立ち上げる

- 現在 2 万カテゴリのラベル付けがされた、1400 万枚の画像が集められている
- すべて人力によるラベル付け



ILSVRC(ImageNet Large Scale Visual Recognition Challenge)

- 画像認識の精度を競うコンペティション

# ILSVRC

全部で 1000 カテゴリの画像が用意され、各テスト画像に含まれている対象のカテゴリをどれだけ正しく認識できるかを競う

## 使用データ

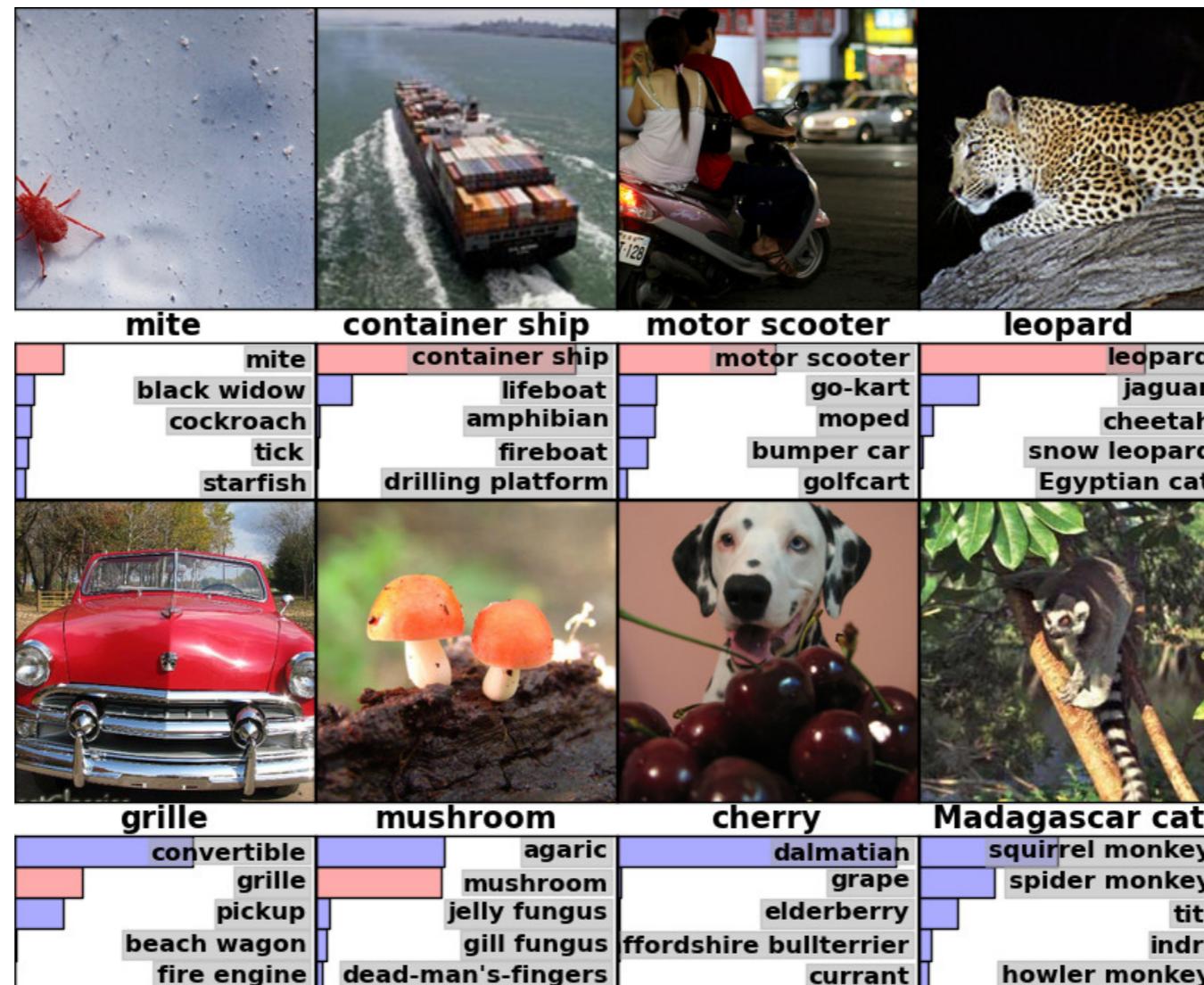
- ▶ 学習用画像：120 万枚
- ▶ 確認用画像：5 万枚
- ▶ テスト用画像：15 万枚



# top-5 エラー率

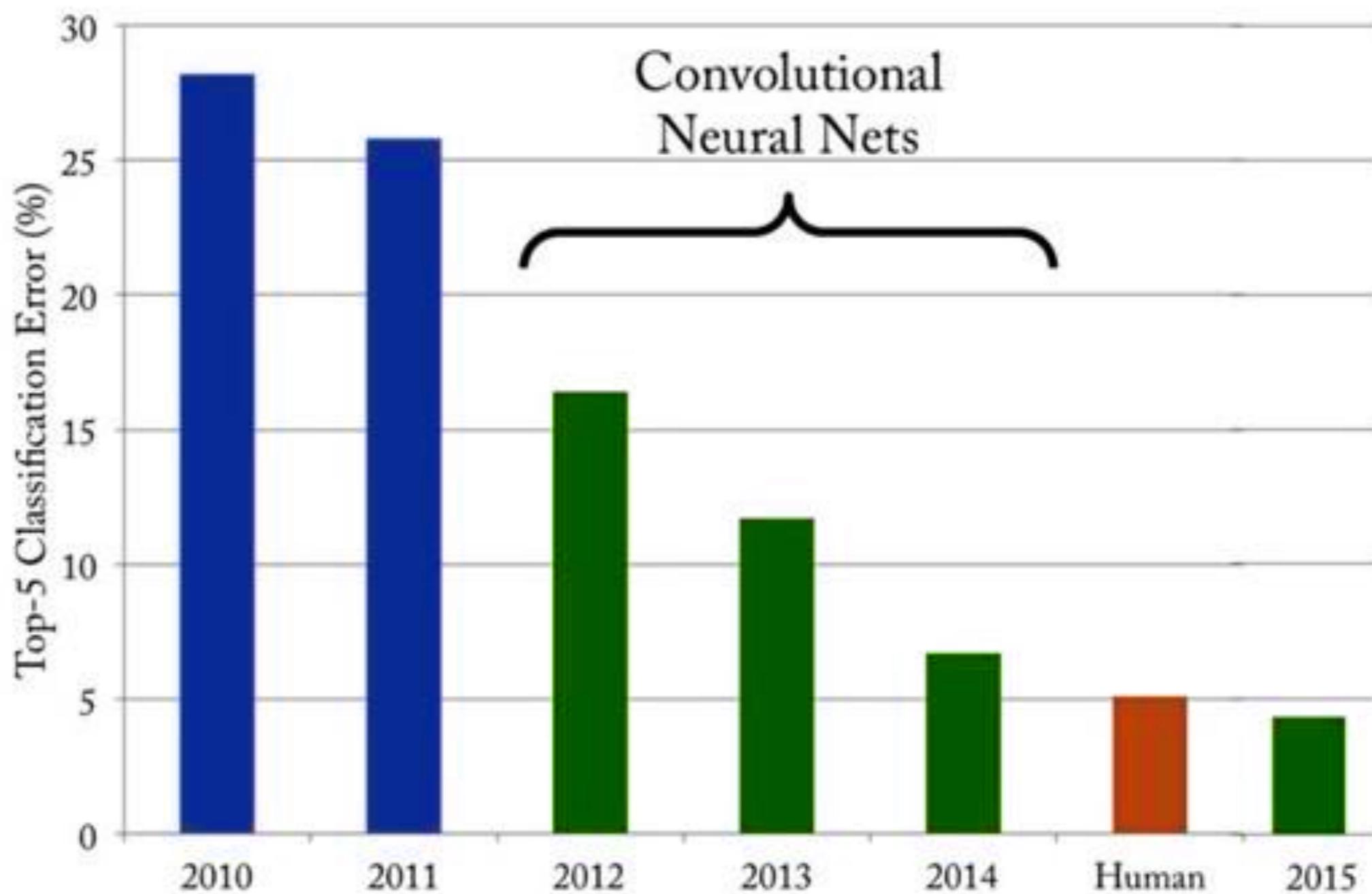
## top-5 エラー率

認識システムが候補としてあげた上位 5 つの答えの中に正解が含まれていれば、認識成功  
ILSVRC では、このエラー率を競う



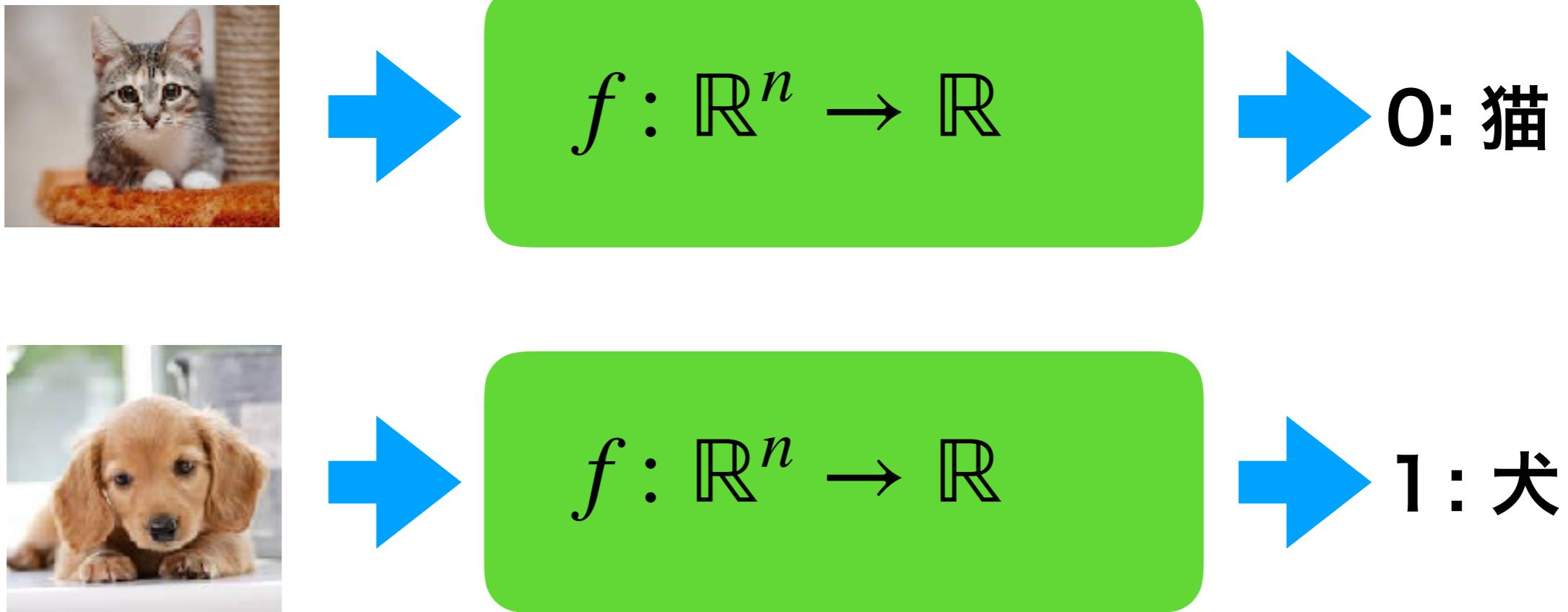
# 画像認識率の進歩

## ImageNet Classification (2010 – 2015)



# イヌ・ネコ分類問題

こんな関数を作りたい！



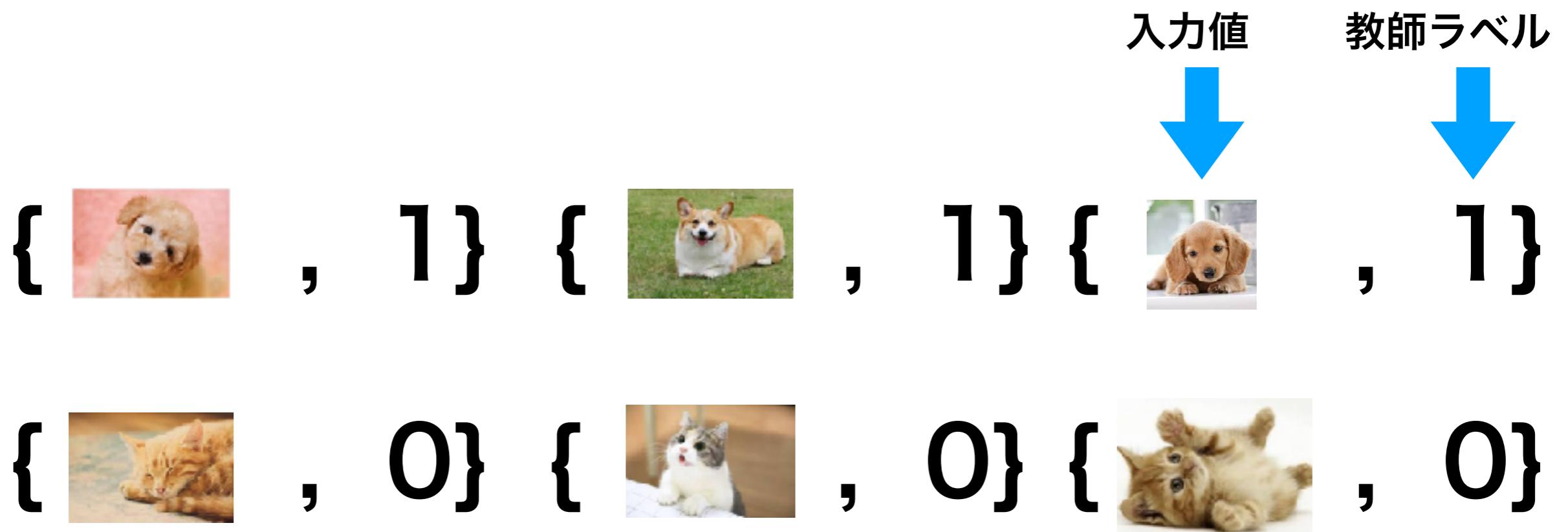
# 関数の形状を制御するパラメータを導入する

パラメトリックモデル

$$f_{\Theta} : \mathbb{R}^n \rightarrow \mathbb{R}$$

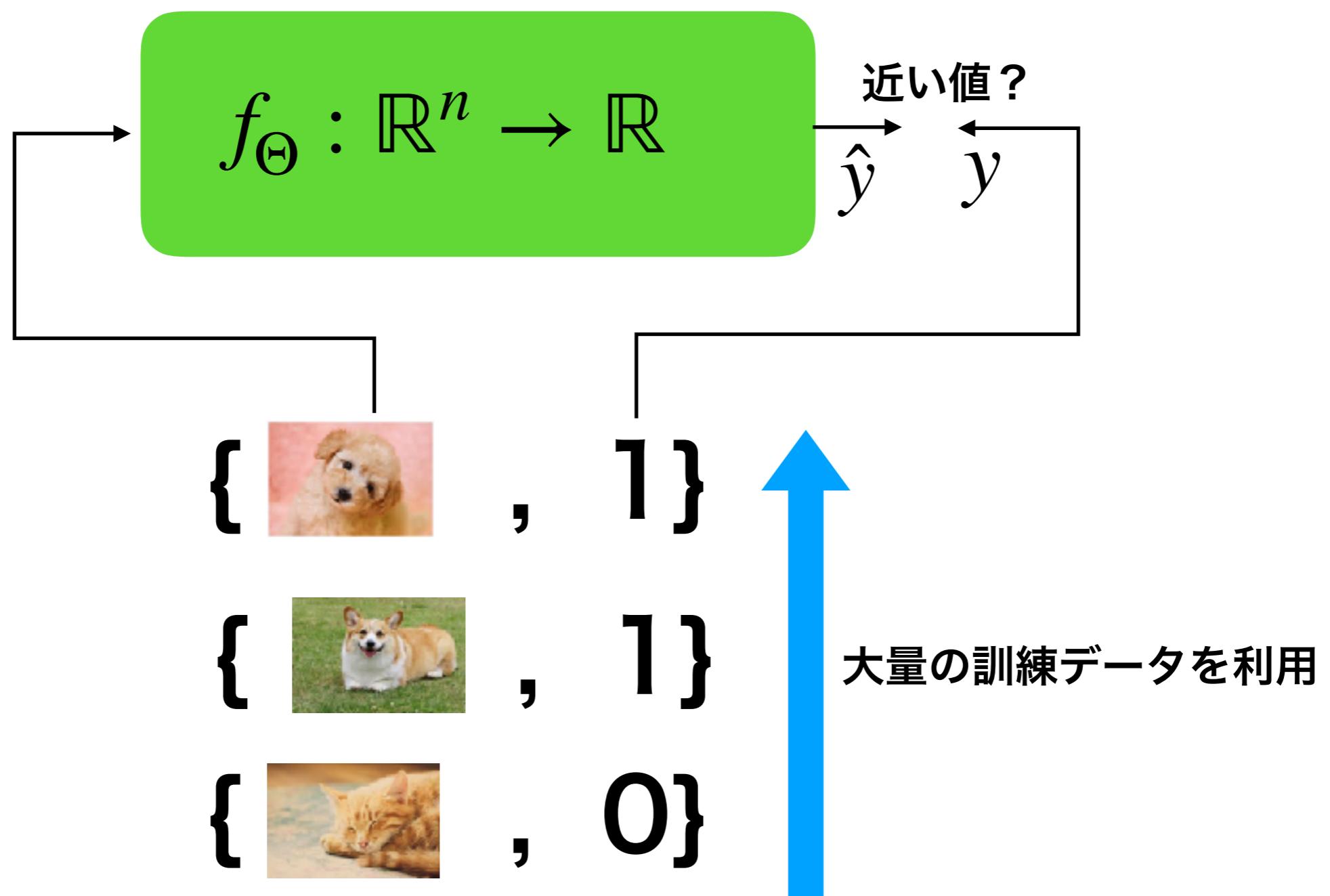
Θ 関数の形状をコントロールするパラメータ群

# 訓練データを準備する



# パラメータを更新する(訓練・学習)

出力と教師ラベルとの間の**食い違い**がなるべく小さくなるように  
パラメータを更新する

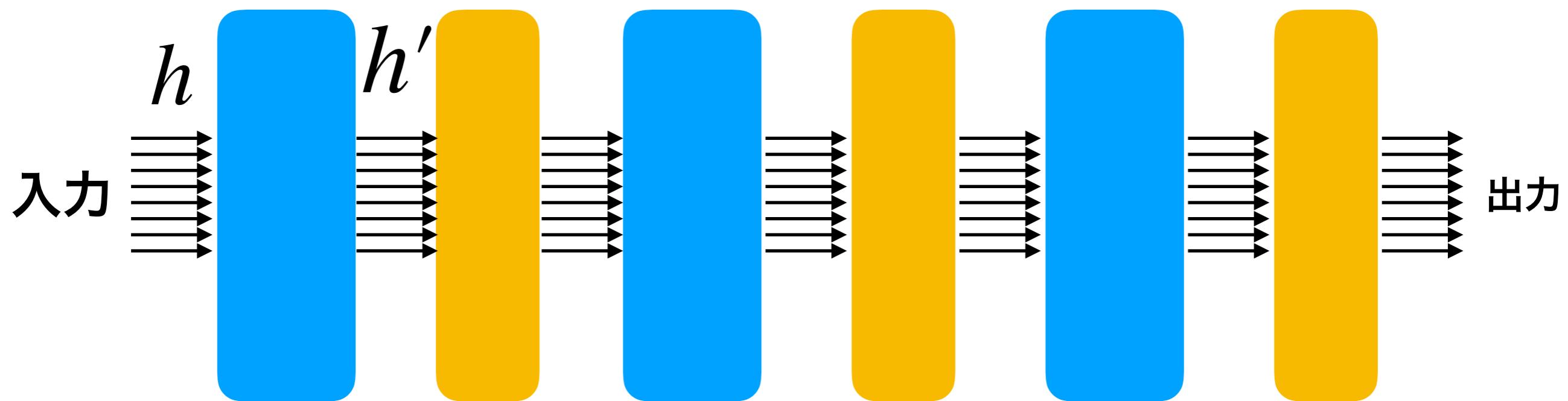


# 深層ネットワークモデル

$$f_{\Theta} : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

線形層 活性化関数

$$\Theta = \{W_1, b_1, W_2, b_2, \dots\}$$



$$h' = \begin{matrix} W \\ \vdots \\ h + b \end{matrix}$$

係数行列 バイアス

# アフィン変換

$$h' = W h + b$$

シグモイド関数

$$\sigma(u) = \frac{1}{1 + e^{-u}}$$

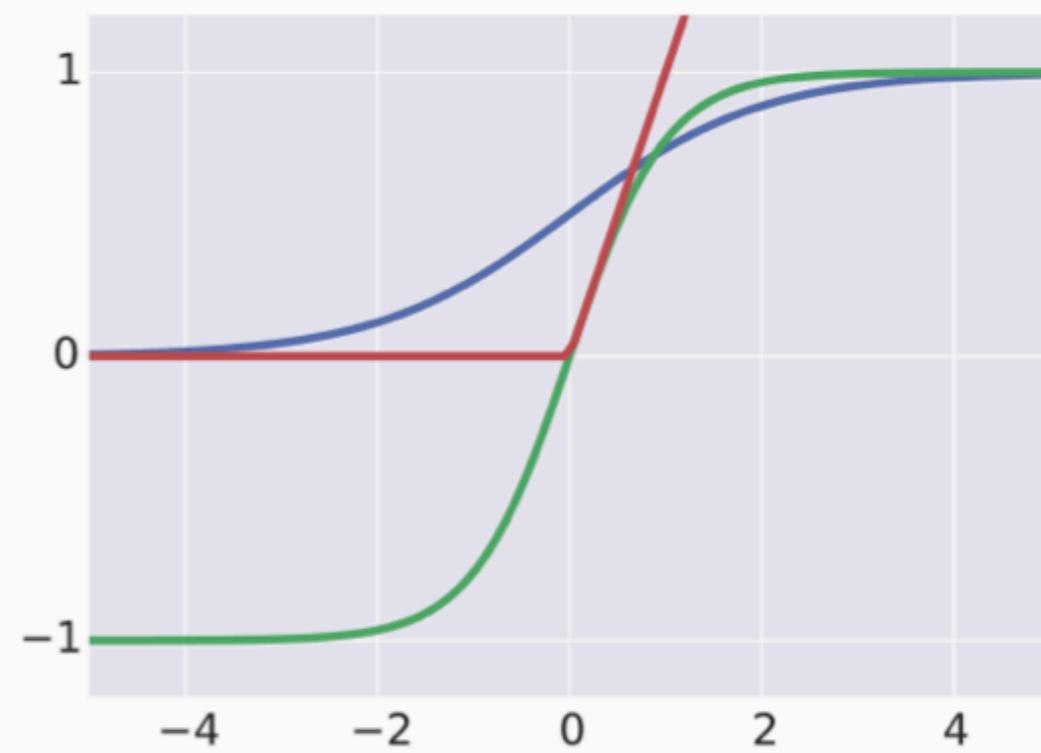
双曲線正接関数

$$\tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

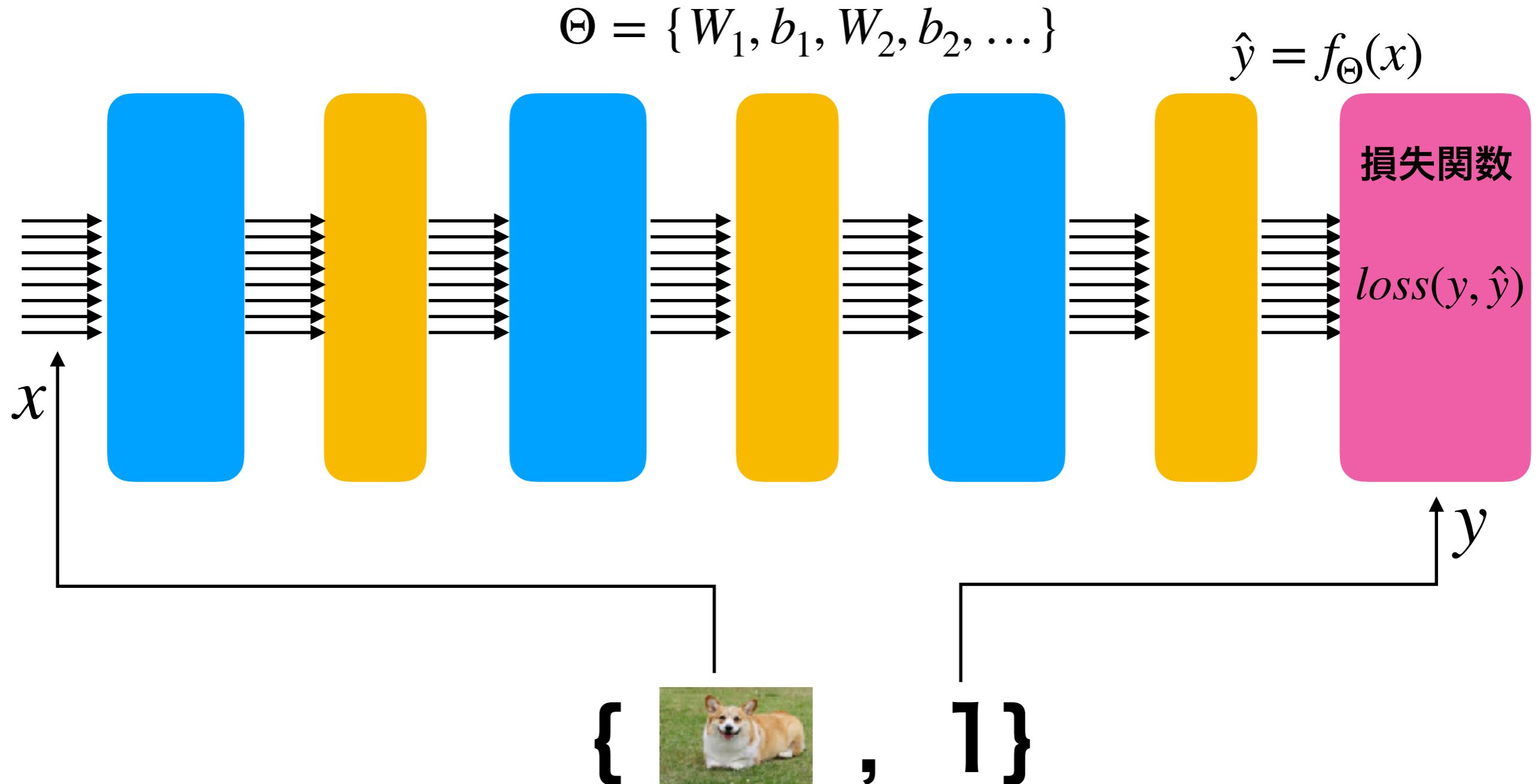
ランプ関数

$$\text{ReLU}(u) = \max(0, u)$$

## 活性化関数

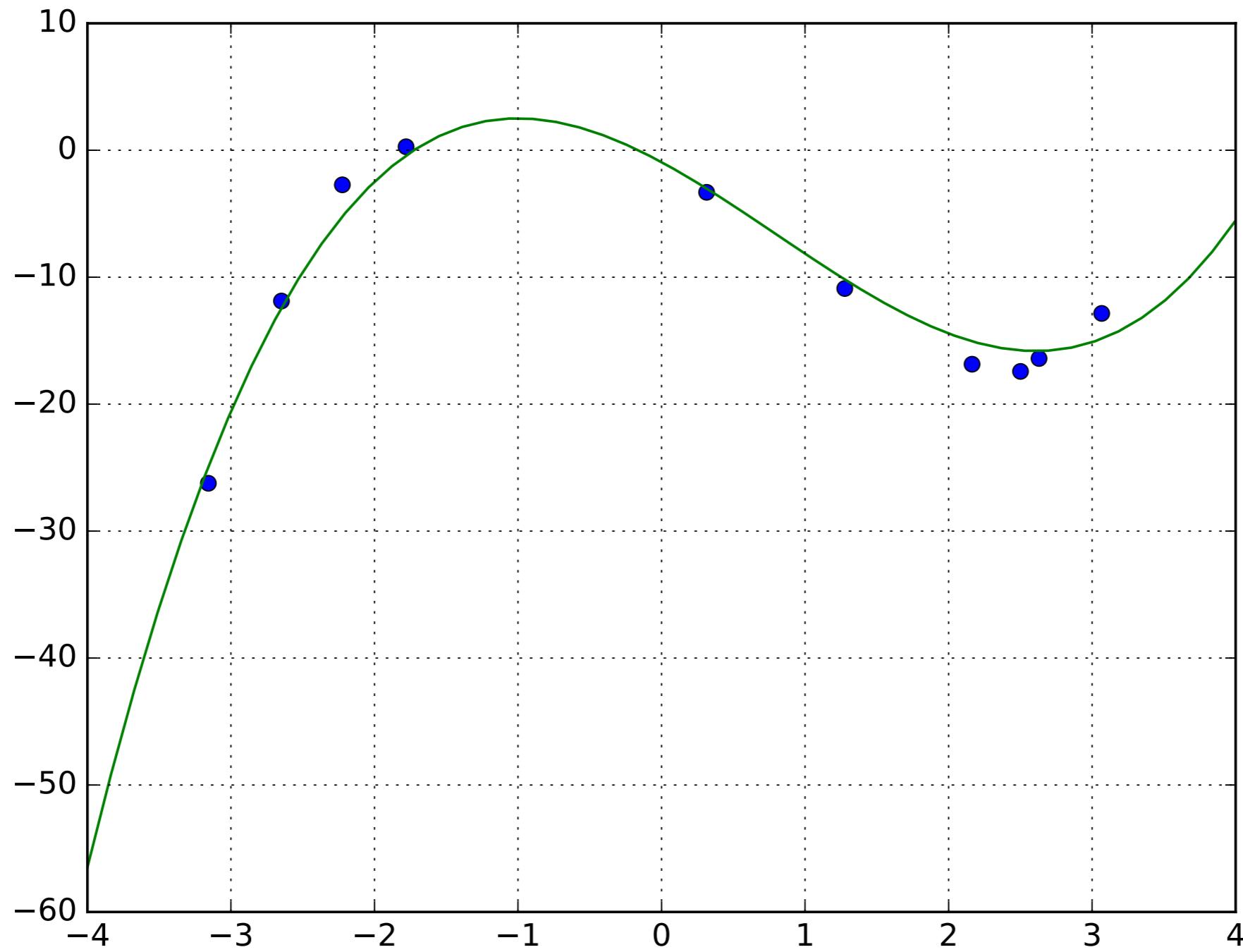


# 深層ネットワークの訓練



訓練・学習過程では、損失関数值を最小化するように  
パラメータを変更する

# 深層学習 = 最小二乗法の親玉？



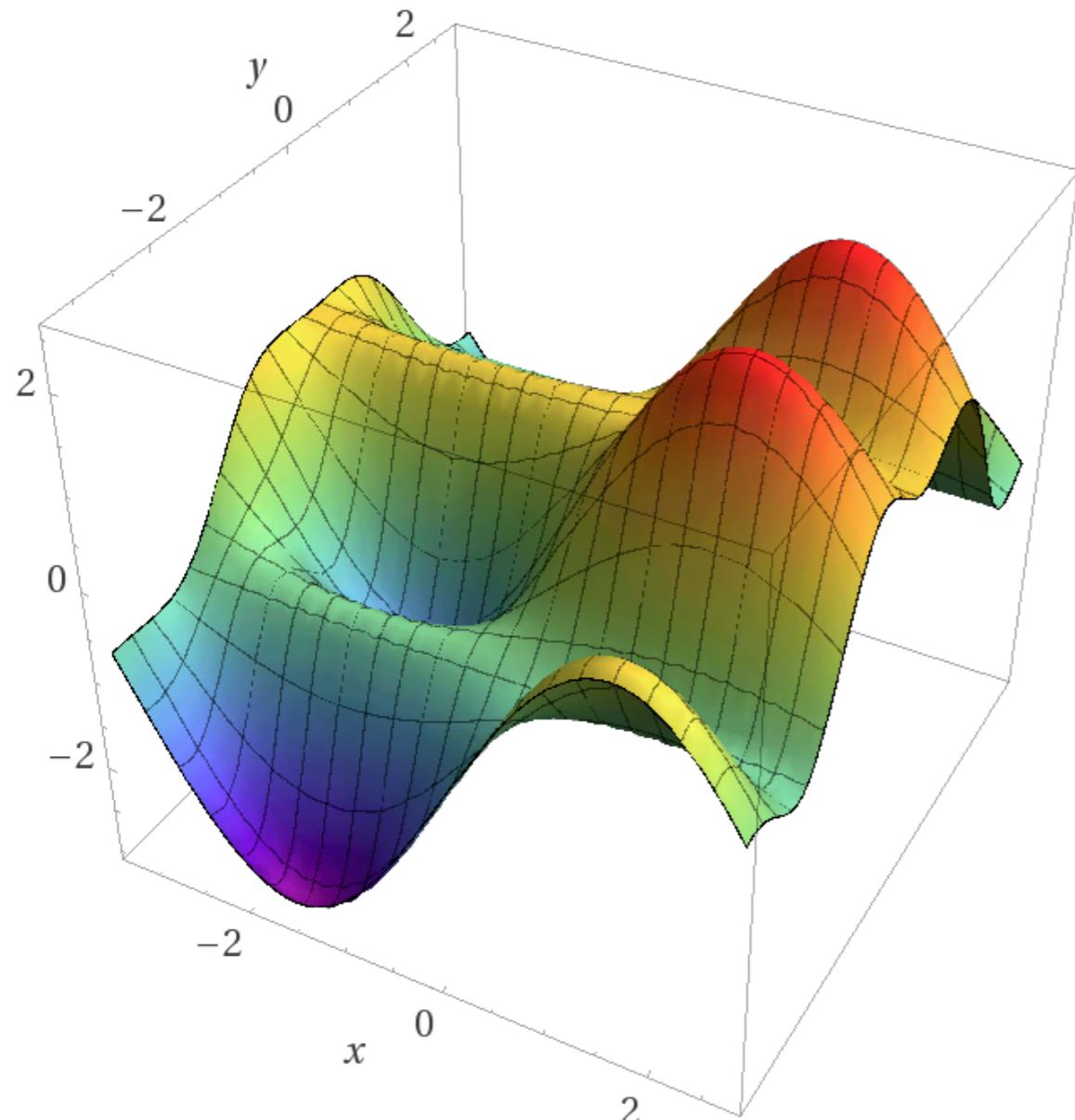
# 学習の手順

- パラメトリック予測モデルを構築（人力）
- 訓練データセット・確認データセット・テストデータセットを作成（人力）
- 訓練プロセスによりパラメータ最適化・フィッティング（自動）
- 確認(validation)データにより、予測精度の測定（自動）
- 精度に満足できなければ最初に戻る
- テスト(test)データにより、予測精度を測定（自動）

# 深層学習に関する補足

-  最適化技法として確率的勾配法を利用
-  パラメータの勾配ベクトルの効率的な計算には誤差逆伝播法(back prop)を利用する
-  よい汎化性能(十分に小さい汎化誤差)を得るために、大量の訓練データが必要
-  一般には、非凸最適化となる

# 非凸目的関数



- 複数の停留点（極値）
- 勾配法では局所解にしか到達できない
- 鞍点付近などでは、勾配法の収束速度が大幅にスローダウン)

# 確率的勾配法(SGD)

訓練データ  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_T, y_T)\}$

ミニバッチ  $B = \{(x_{b1}, y_{b1}), (x_{b2}, y_{b2}), \dots, (x_{bK}, y_{bK})\}$

目的関数  $G_B(\Theta) = \frac{1}{K} \sum_{k=1}^K loss(y_{bk} - f_\Theta(x_{bk}))$

## 確率的勾配法に基づく最小化

Step 1 (初期点設定)  $\Theta := \Theta_0$

Step 2 (ミニバッチ取得)  $B$  をランダムに生成

Step 3 (勾配ベクトルの計算)  $g := \nabla G_B(\Theta)$

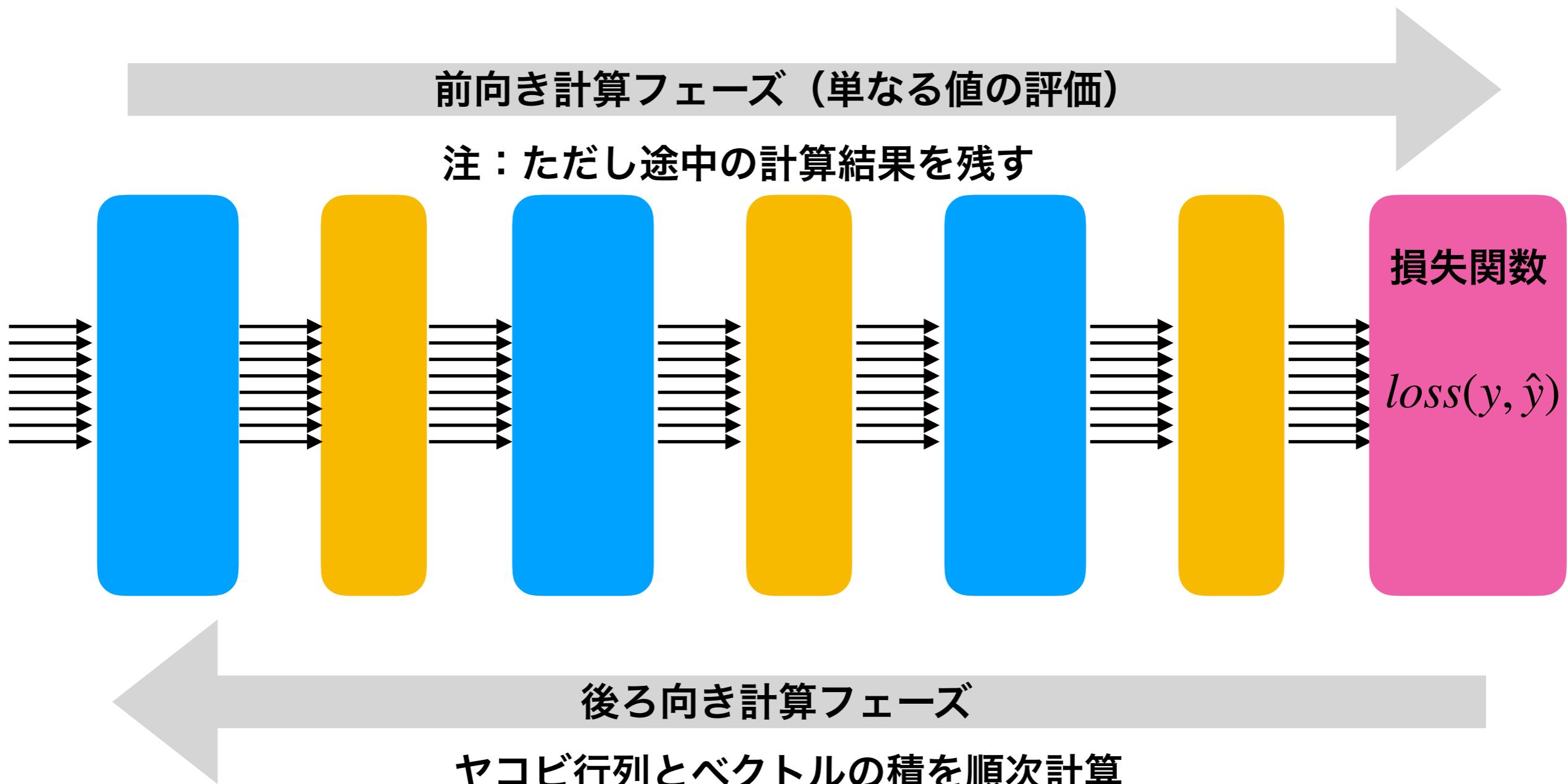
Step 4 (探索点更新)  $\Theta := \Theta - \alpha g$

Step 5 (反復) Step 2 に戻る

バリエーション  
**Momentum**  
**AdaDelta**  
**RMSprop**  
**Adam**

# 誤差逆伝播法(backprop)

- ・パラメータの勾配ベクトルを効率良く求めることが目的
- ・微分の連鎖律の利用(BCJRアルゴリズムにとても良く似ている)



# 機械学習技術の分類

- 教師付き学習 (supervised learning)
- 教師なし学習 (unsupervised learning)
- 強化学習 (reinforcement learning)
- 生成モデル (generative model)

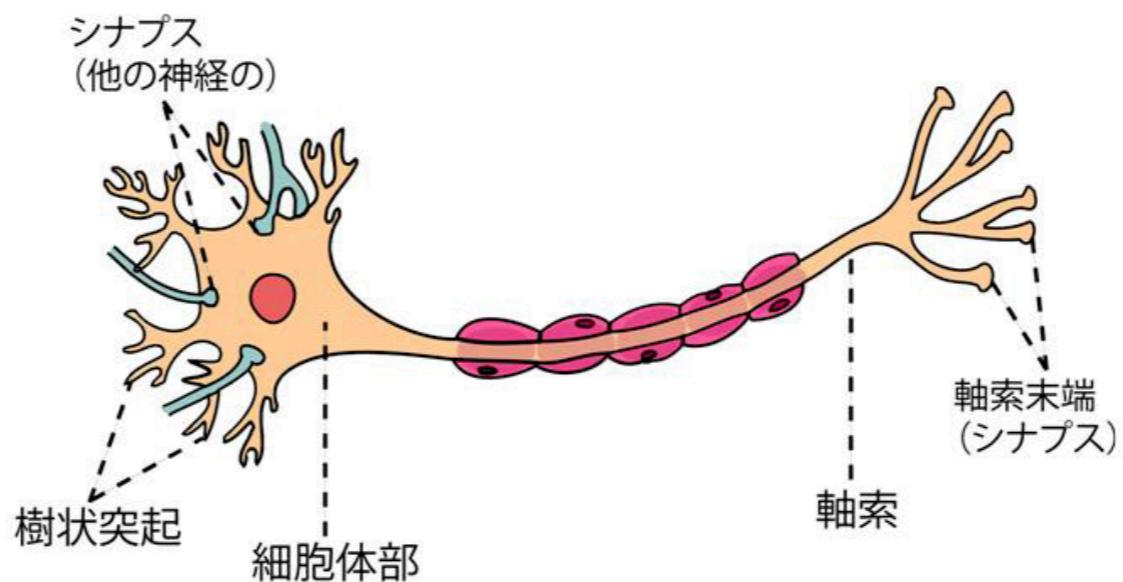
# ニューラルネットワーク研究の歴史を辿る

- 深層学習技術は複数の技法からなるいわばツールセット
- 歴史的コンテキストもある程度知っておくと各ツールの位置付けが明確になる

# 形式ニューロンの登場

- ▶ 脳神経学者のマカロックと數学者のピツは、神経細胞を模した「形式ニューロン」を導入した。(McCulloch, W. and Pitts, W. (1943).)
- ▶ 現実のニューロンの振る舞いを簡略化して得られた
- ▶ 形式ニューロンは、現実のニューロンが「発火するか (1)」または「発火しないか (0)」という 2 状態を持つとしてモデル化を行っている。

ニューロン（脳の神経細胞）



出典

<https://qiita.com/nishiy-k/items/1e795f92a99422d4ba7b>

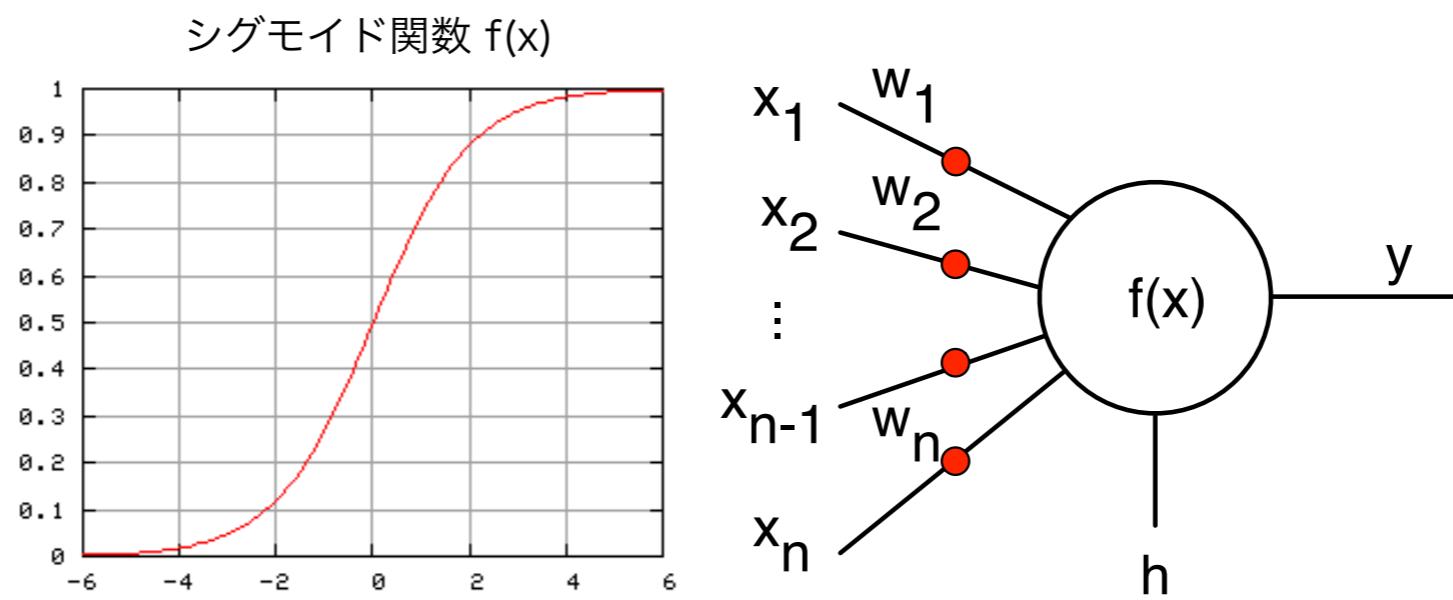
# ニューラルネットワークの登場

人工ニューロン素子は次の構造を持つ:

$$y = f \left( \sum_{i=1}^n w_i x_i + b \right) \quad (2)$$

関数  $f$  は活性化関数 (activation function) と呼ばれる関数で、1980 年代には、シグモイド関数 (sigmoid function) が利用されることが多かった。(出力が実数値になったことに注意! )

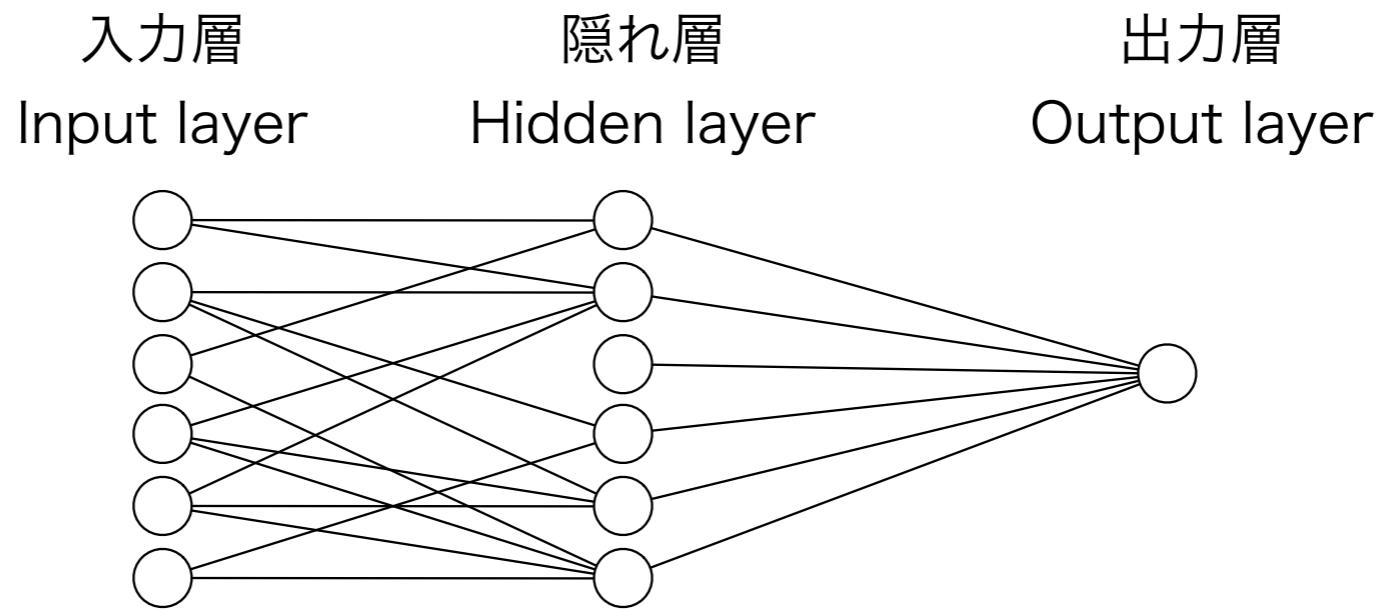
$$f(x) = \frac{1}{1 + e^{-x}} \quad (3)$$



出典 <https://ja.wikipedia.org/wiki/シグモイド関数>

# 3層ネットワークの隆盛 (1980年代)

- ▶ 人工ニューロンからなる3層順伝播型ネットワークの優れたパターン認識性能が知られてきた。



- ▶ 活性化関数が微分可能な関数となったことで、 $w_i$  や  $b$  の微分を考えることができるようになった → 勾配法による学習パラメータの更新
- ▶ ルメルハートらによる逆誤差伝播法 (1986) により、学習パラメータの勾配ベクトル (gradient) の高速計算が可能に。
- ▶ 3層以上のネットワークの学習 → 勾配消失による学習の困難性の認識が広がった → 研究活動の沈滯 (2回目の冬)

# 人工ニューラル素子(ふたたび)

人工ニューロン素子は次の構造を持つ:

$$y = f \left( \sum_{i=1}^n w_i x_i + b \right) \quad (4)$$

関数  $f$  は活性化関数であり、さまざまな種類の関数が利用されている:

シグモイド関数

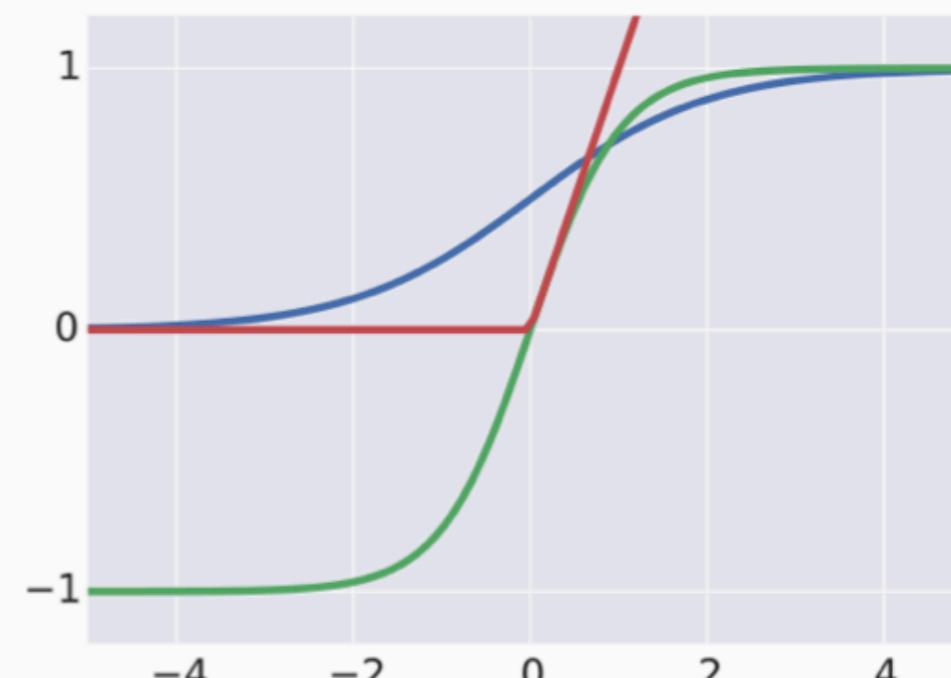
$$\sigma(u) = \frac{1}{1 + e^{-u}}$$

双曲線正接関数

$$\tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

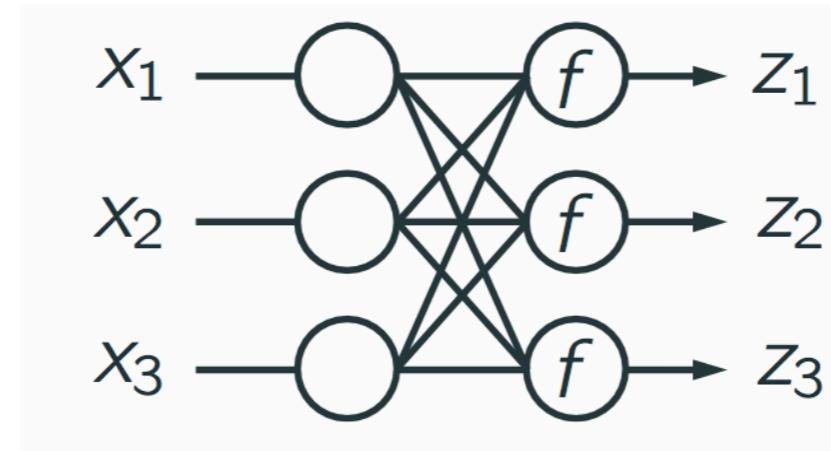
ランプ関数

$$\text{ReLU}(u) = \max(0, u)$$



## レイヤー(層)

順伝播型ネットワークでは、複数のレイヤーが直列に接続される。ひとつのレイヤーは次のような形状をしている：



ここで、レイヤーへの入力ベクトルを  $x = (x_1, x_2, \dots, x_n)^T$ 、出力ベクトルを  $z = (z_1, z_2, \dots, z_m)^T$  とするとき、

$$z = f(Wx + b) \quad (5)$$

と書くことができる。ここで、

$$W = \begin{pmatrix} W_{1,1} & \dots & W_{1,n} \\ W_{2,1} & \dots & W_{2,n} \\ \vdots & \vdots & \vdots \\ W_{m,1} & \dots & W_{m,n} \end{pmatrix}, \quad b = (b_1, b_2, \dots, b_m)^T \quad (6)$$

# ネットワークの含む学習パラメータ

順伝播型ネットワーク  $y = F(x)$  は、調節可能なパラメータ 1

$$\Theta = \left\{ W^{(1)}, \dots, W^{(L)}, b^{(1)}, \dots, b^{(L)} \right\}$$

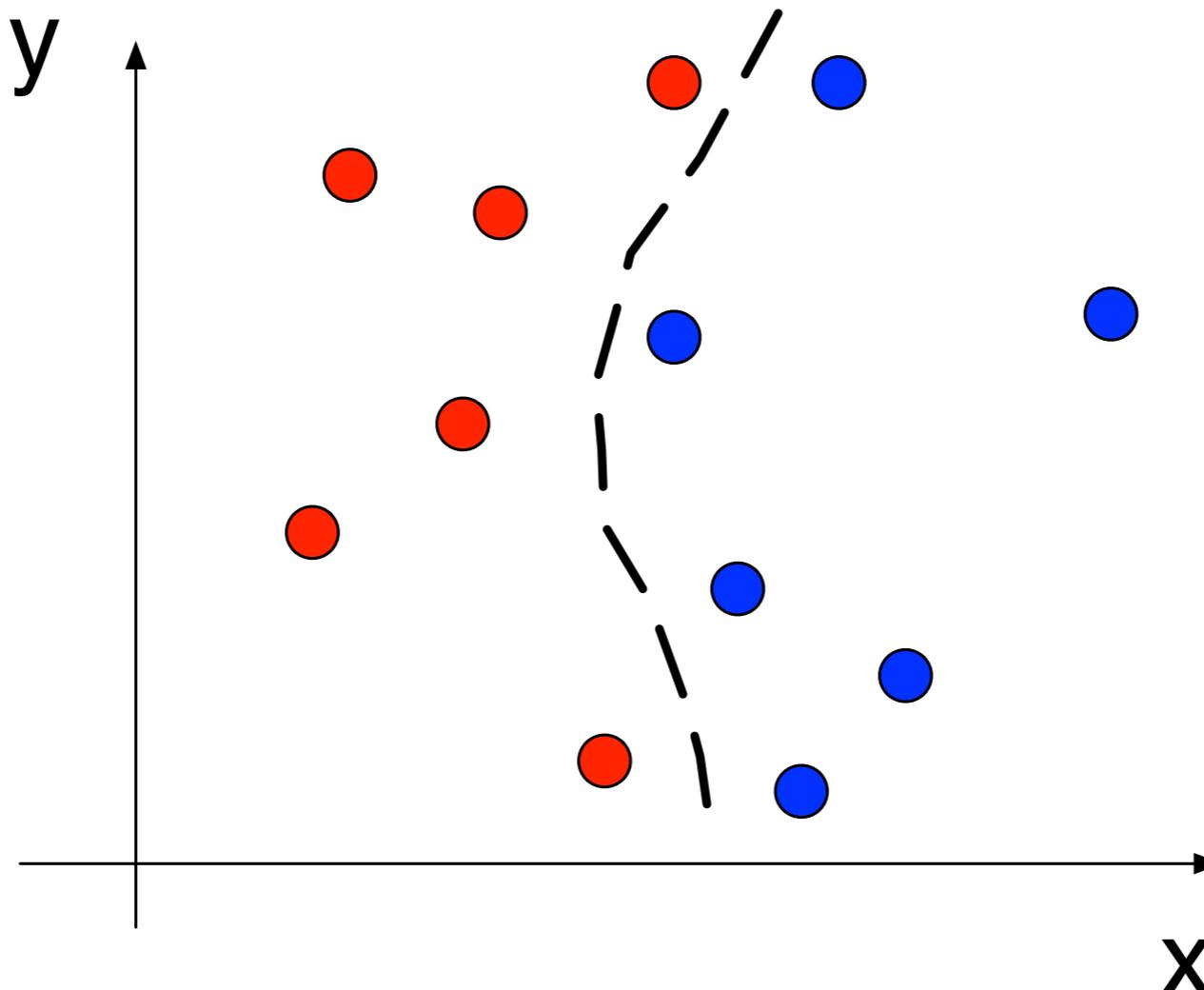
を持つ。これらの学習パラメータを訓練データに従い、調節することを「学習」または「訓練(トレーニング)」という。パラメータを明示したい場合には、

$$y = F(x; \Theta) \tag{9}$$

と表記する場合もある。学習パラメータの調節(学習)においての目標は、**損失関数值(クラス判別問題)**・**誤差関数值(回帰問題)**を小さくすることにある。

## クラス判別問題 (2値判別を例として)

各データ  $(x, y)$  の組は、ラベル（赤、青）を持つ。訓練データは  $(x_i, y_i, t_i) (i = 1, 2, \dots, T)$  の形式である。 $t_i$  はラベルを意味する。



順伝播型ネットワークは、未知のデータ  $(x, y)$  に対して、そのデータが属するクラス (= ラベル) を可能な限り正しく推定するよう学習したい → クラス判別問題

# 多クラス分類問題



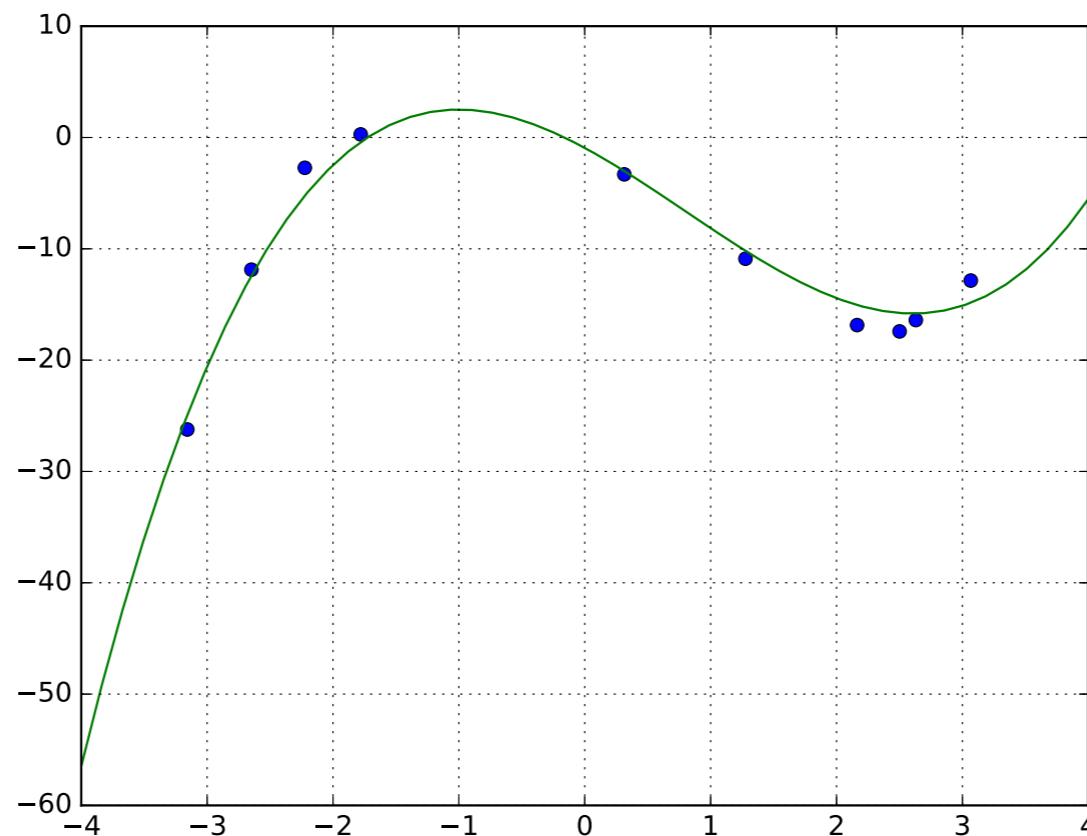
MNISTデータセット



CIFAR10データセット

# 回帰問題

- ▶ いま、未知の関数  $G(x)$  からひとつの観測データ  $(x^*, y^*)$  が得られたものとしよう。ここで、 $y^* = G(x^*)$  である。
- ▶ 順伝播ネットワーク  $F(x; \Theta)$  を利用して、関数  $G(x)$  を近似したい。
- ▶ 未知関数の近似問題 → **回帰問題**



# 順伝播型ネットワークの学習の流れ（回帰）の基本形

- ▶ 訓練データを用意する。訓練データ：（入力値のベクトル、出力値のベクトル）
- ▶ 誤差関数を用意する。
- ▶ 順伝播型ネットワークを用意する。

$$z^{(i)} = f^{(i)} \left( W^{(i)} z^{(i-1)} + b^{(i)} \right)$$

- ▶ 勾配法を利用して誤差関数值を最小とする学習パラメータ  $\Theta^*$  を求める（以上が「訓練（training）フェーズ」）。
- ▶  $F(x; \Theta)$  を利用して、予測計算などを行う（これは「推論（inference）フェーズ」と呼ばれることが多い）

1980年代までの3層ネットワークの学習には、おおむね上記の手順が利用されていた。現在の深層学習においても「学習の大枠」は以上の通りである。

# 多層化への困難を乗り越える

- 単純な学習戦略は、多層化が困難であることが80年代にわかつてきた
- この困難を乗り越えるための複数の工夫が深層学習を支える重要な要素技術となっている

# 単純な学習手法が抱えるいくかの困難

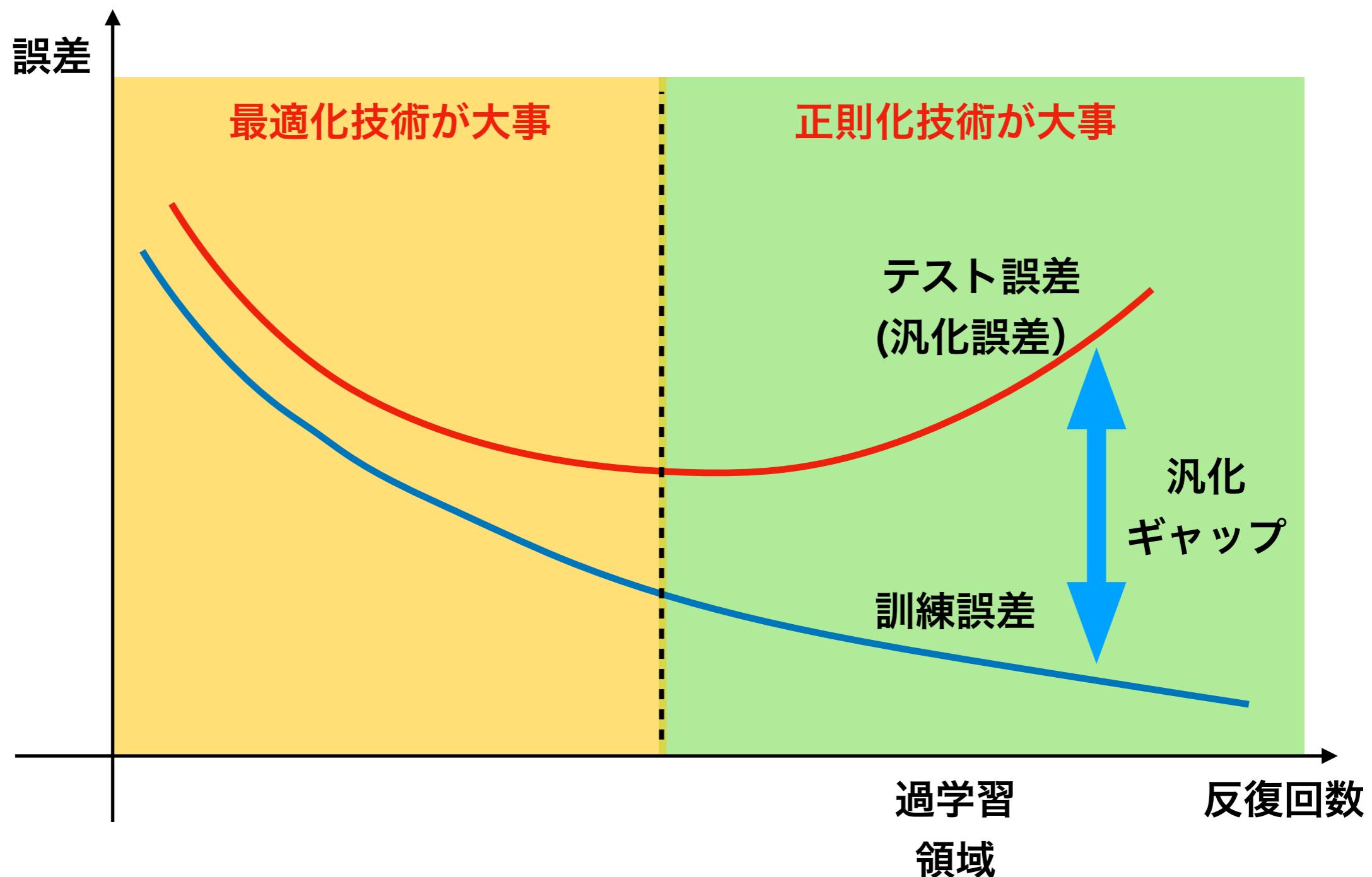
層の数を増していくことで、順伝播ネットワークの性能は向上していきそうな雰囲気は感じていた(80年代初頭)ところが。。。

- ▶ 過学習
- ▶ 勾配ベクトルの計算がかなり大変(計算量的な意味で)
- ▶ 勾配消失問題(入力層に近いところの学習パラメータの微分値がほとんどゼロになる)
- ▶ 学習プロセスの停滞(プラトーに陥る。。。)

3層より多層のネットワークの学習は、ほとんど望みがなさそうに思われた→第2次ブームの終焉

# 汎化誤差

訓練データに含まれない初見のデータに対して適切な出力を与えることが望ましい

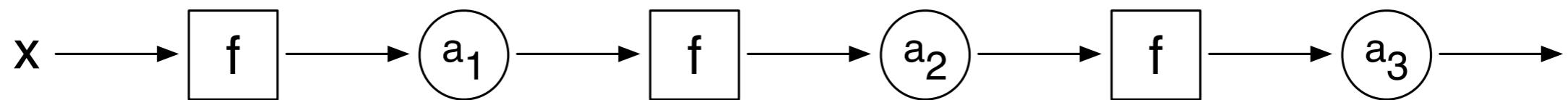


# 勾配ベクトルの計算における困難

極端に簡単化した順方向ネットワークを考える。

$$F(x; A) = a_3 f(a_2 f(a_1 f(x)))$$

という関数が与えられている。ここで、 $A = \{a_1, a_2, a_3\}$  である。



$$\frac{\partial F(x; A)}{\partial a_1}$$

を求めよ。

# 合成関数の微分 (微分の連鎖律)

$$y = f(g(x))$$

について、 $\frac{\partial y}{\partial x}$  を求めたい。このとき、まず

$$u = g(x) \tag{10}$$

$$y = f(u) \tag{11}$$

と分ける。このとき、合成関数の微分公式は (微分の連鎖律) は次のとおり:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \times \frac{\partial u}{\partial x} \tag{12}$$

## 連鎖律にもとづき計算してみる

$F(x; A) = a_3 f(a_2 f(a_1 f(x)))$  を

$$u_1 = a_1 f(x) \quad (13)$$

$$u_2 = a_2 f(u_1) \quad (14)$$

$$u_3 = a_3 f(u_2) \quad (15)$$

と書き換える。ここで、連鎖律を使うと

$$\frac{\partial u_3}{\partial a_1} = \frac{\partial u_3}{\partial u_2} \times \frac{\partial u_2}{\partial u_1} \times \frac{\partial u_1}{\partial a_1} \quad (16)$$

である。ここで、

$$\frac{\partial u_1}{\partial a_1} = f(x), \quad \frac{\partial u_2}{\partial u_1} = a_2 f'(u_1), \quad \frac{\partial u_3}{\partial u_2} = a_3 f'(u_2)$$

より、

$$\frac{\partial u_3}{\partial a_1} = a_3 f'(u_2) a_2 f'(u_1) f(x) \quad (17)$$

を得る。

# 前向き・後ろ向き計算 (バックプロパゲーションの雛形)

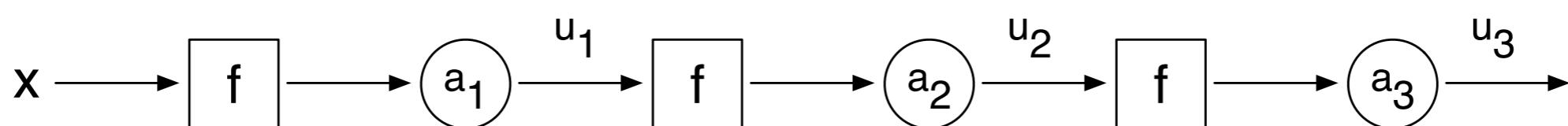
$$u_1 = a_1 f(x) \quad (18)$$

$$u_2 = a_2 f(u_1) \quad (19)$$

$$u_3 = a_3 f(u_2) \quad (20)$$

$$\frac{\partial u_3}{\partial a_1} = a_3 f'(u_2) a_2 f'(u_1) f(x) \quad (21)$$

前向き計算



$$f(x)$$

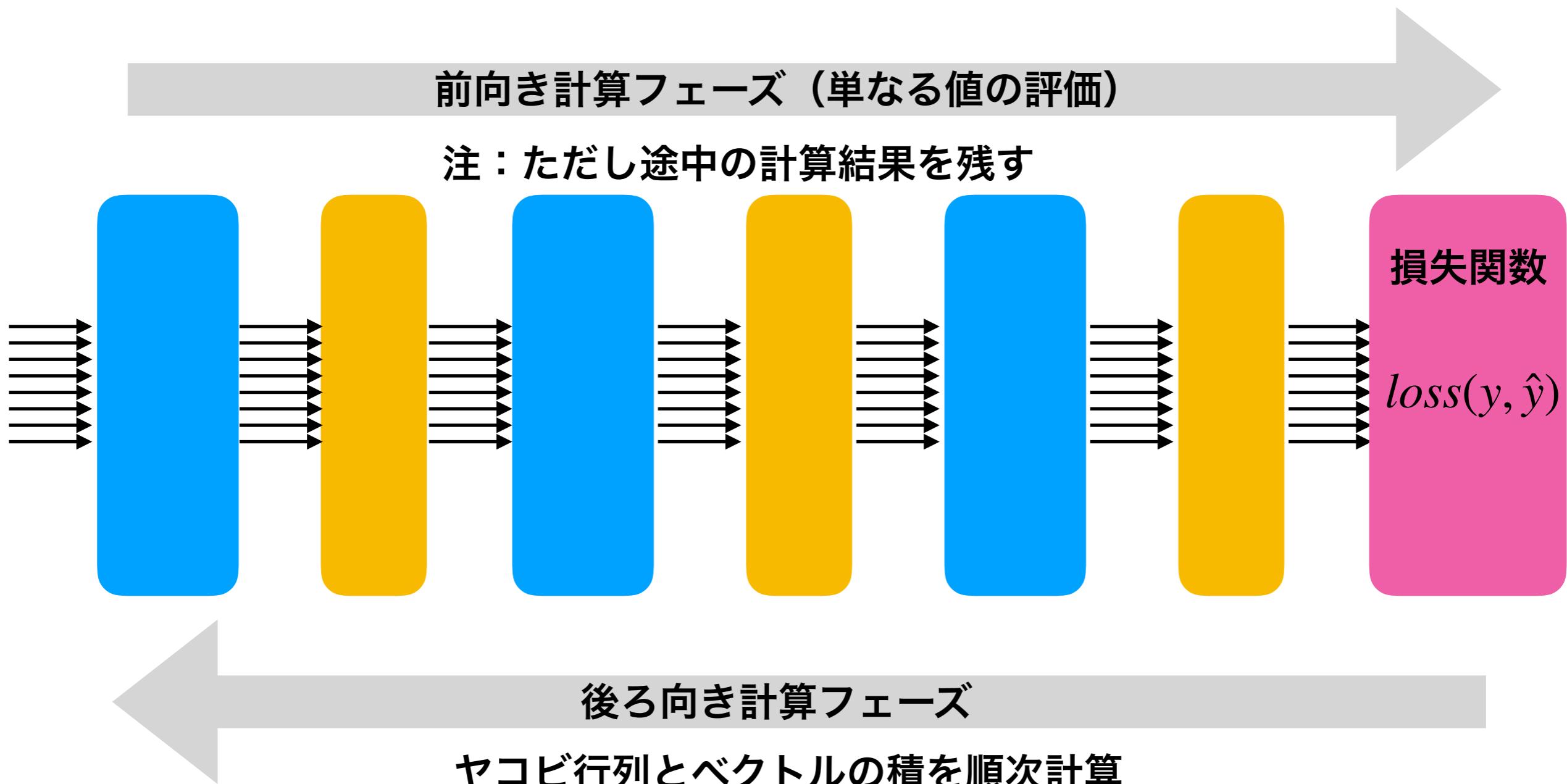
$$a_2 f'(u_1)$$

$$a_3 f'(u_2)$$

後ろ向き計算

# 誤差逆伝播法(backprop)

- ・パラメータの勾配ベクトルを効率良く求めることが目的
- ・微分の連鎖律の利用(BCJRアルゴリズムにとても良く似ている)



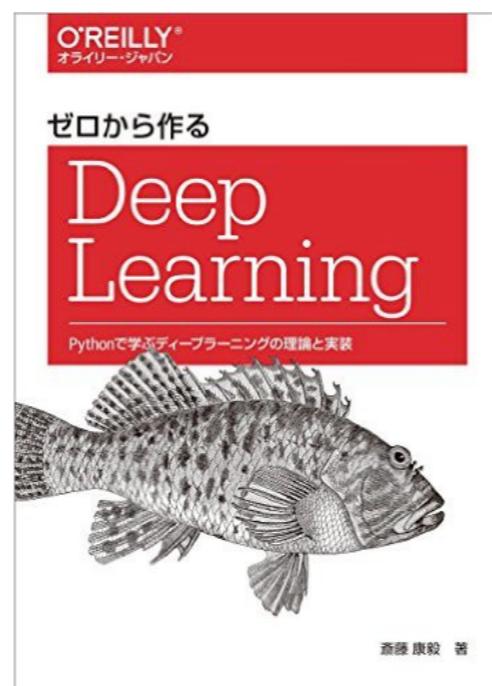
# 逆誤差伝播法（バックプロパゲーション）

80年代に開発された (Rumelhart ら) 学習パラメータの勾配ベクトルの高速算法

- ▶ 前述の例のように「合成関数の微分に関する連鎖律」を利用
- ▶ 前向き計算(推論計算)と後向き計算(誤差逆伝播計算)を順に実行
- ▶ 計算結果として得られる学習パラメータの勾配ベクトルを利用して、学習パラメータを更新(勾配法)
- ▶ 深層学習の学習計算で一番計算時間がかかるっている部分である。ここを正確に動作し、かつ高速なプログラムを書くのはかなり大変(青本・赤本に正確な手続きが書かれているので、これらを見ながら書けば「正確には書ける」)。
- ▶ 幸いなことに、最近のフレームワーク(TensorFlow, Chainer, Caffeなど)では、前向き計算の記述をすれば、逆伝播計算は自動で行ってくれる。その意味では、エンドユーザは逆誤差伝播に関するプログラミングについて頭を悩ます必要はほとんどない。

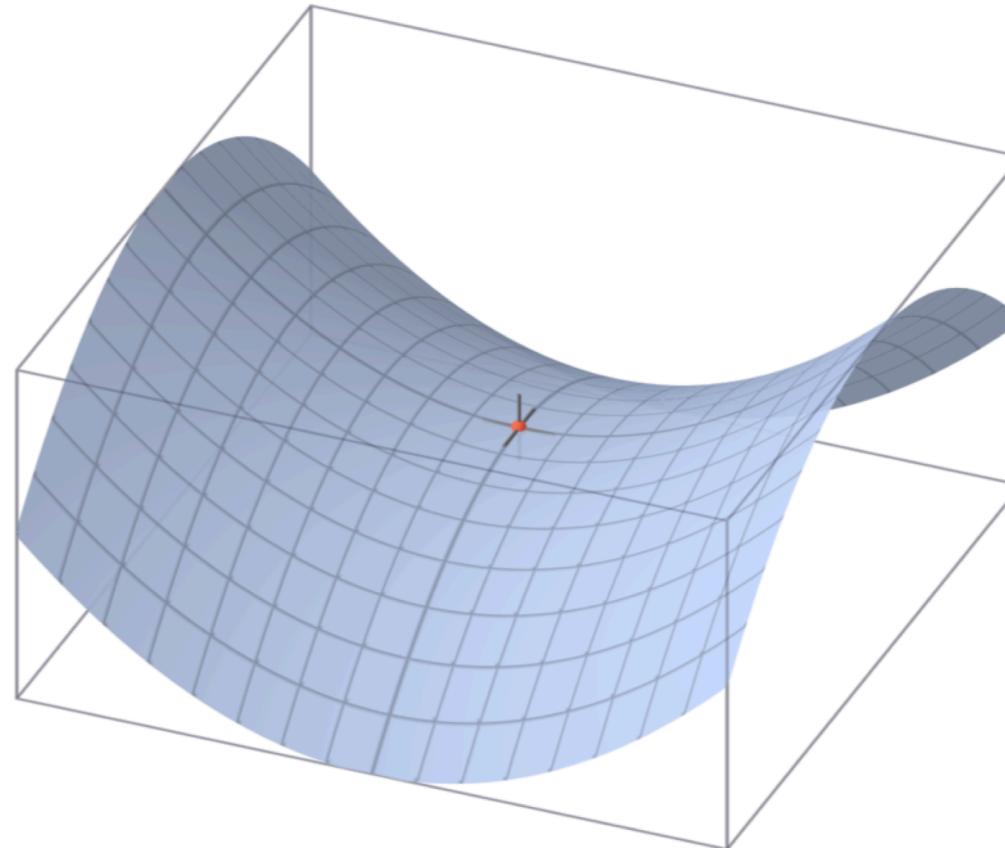
# Backpropを理解するコツ

- 簡単な例で紙の上で前向き計算・後ろ向き計算をシミュレートしてみる
- フルスクラッチでプログラムを書いてみる(下記の本を少し見ながら。。。)



# 鞍点と学習の停滞

勾配ベクトルが、ほとんどゼロベクトルとなる点（停留点）は、「極大点」「極小点」「鞍点」のいずれか。**探索点が鞍点にとらわれているとき、ニューラルネットの学習が停滞する**

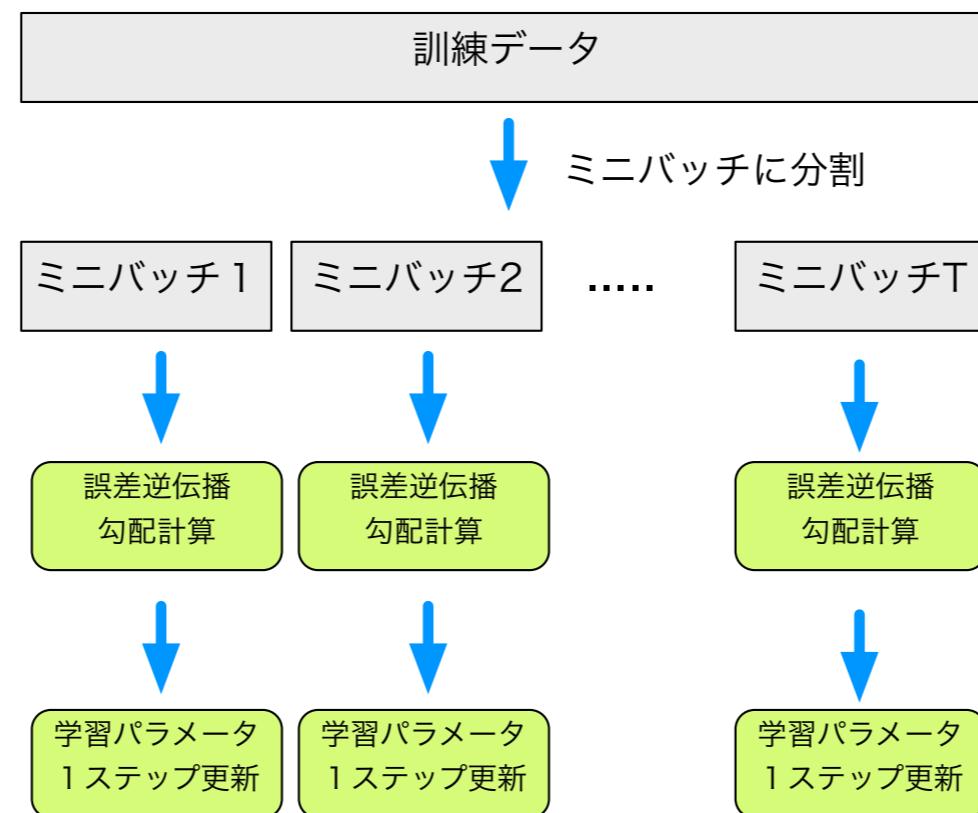


4段以上のネットワークにおいては、停留点による学習の停滞の影響が顕著→学習が進まない！

# 学習の停滞に関する対応策 (1)

## ミニバッチ学習 (確率的勾配法; SGD)

- ▶ 訓練データをミニバッチと呼ばれる組に分割する。
- ▶ ミニバッチごとに (1) 勾配法の実行、(2) 学習パラメータの更新を行う。



ミニバッチごとに目的関数が切りかわる → 確率的ゆらぎの導入  
→ 停留点からの脱出の可能性が高まる

# 確率的勾配法(SGD)

訓練データ  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_T, y_T)\}$

ミニバッチ  $B = \{(x_{b1}, y_{b1}), (x_{b2}, y_{b2}), \dots, (x_{bK}, y_{bK})\}$

目的関数  $G_B(\Theta) = \frac{1}{K} \sum_{k=1}^K loss(y_{bk} - f_\Theta(x_{bk}))$

## 確率的勾配法に基づく最小化

Step 1 (初期点設定)  $\Theta := \Theta_0$

Step 2 (ミニバッチ取得)  $B$  をランダムに生成

Step 3 (勾配ベクトルの計算)  $g := \nabla G_B(\Theta)$

Step 4 (探索点更新)  $\Theta := \Theta - \alpha g$

Step 5 (反復) Step 2 に戻る

バリエーション  
**Momentum**  
**AdaDelta**  
**RMSprop**  
**Adam**

# 確率的勾配法の処理

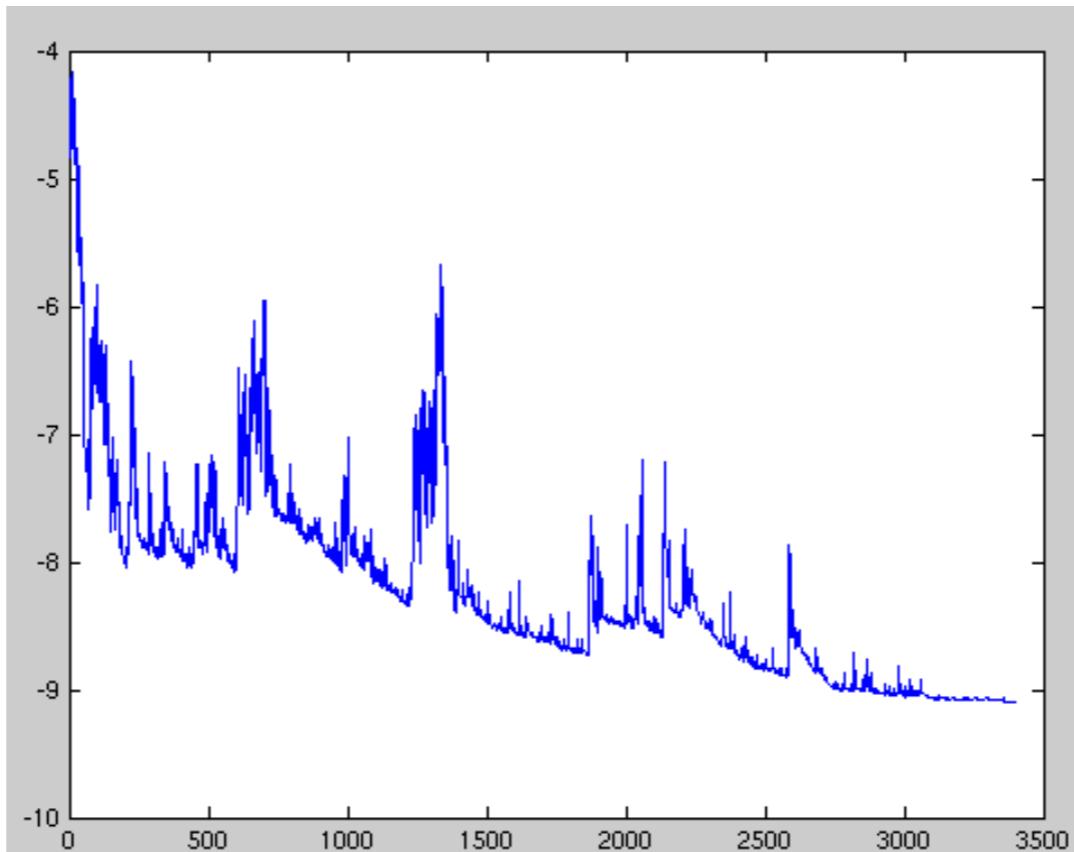
最も基本的な確率的勾配法の基本処理は下記のとおりである。

$\mathbf{B}_i$ : ( $i$  番目のミニバッチ)、 $E$ :誤差関数

勾配の計算  $\mathbf{g} := \nabla E(\mathbf{B}_i)$

探索点更新  $\Theta := \Theta - \alpha \mathbf{g}$

ループ条件 未処理のミニバッチが残っている場合は、勾配計算に戻れ。そうでなければ、 $\Theta$ を出力して終了せよ。



## さまざまな確率的勾配法

最もシンプルな確率的勾配法もしばしば利用されているが、より広い範囲の問題に対して高速に収束する（ことが実験的に確認されている）確率的勾配法の改良法（加速手法）が知られている（だけではなく、よく利用されている）。

- ▶ Momentum (慣性法)
- ▶ Adagrad
- ▶ Adadelta
- ▶ RMSprop
- ▶ Adam

どれが良いかは問題依存だが経験的には、単純な SGD と比較すると **Adam, RMSprop** が使いやすく、精度・収束時間に格段の向上が見られることが多い。これらのいずれもが TensorFlow, Chainer といったフレームワークでは、**最初から実装がついてくる**ので、エンドユーザが実装について悩む必要はない（選択は悩むが。。。）。

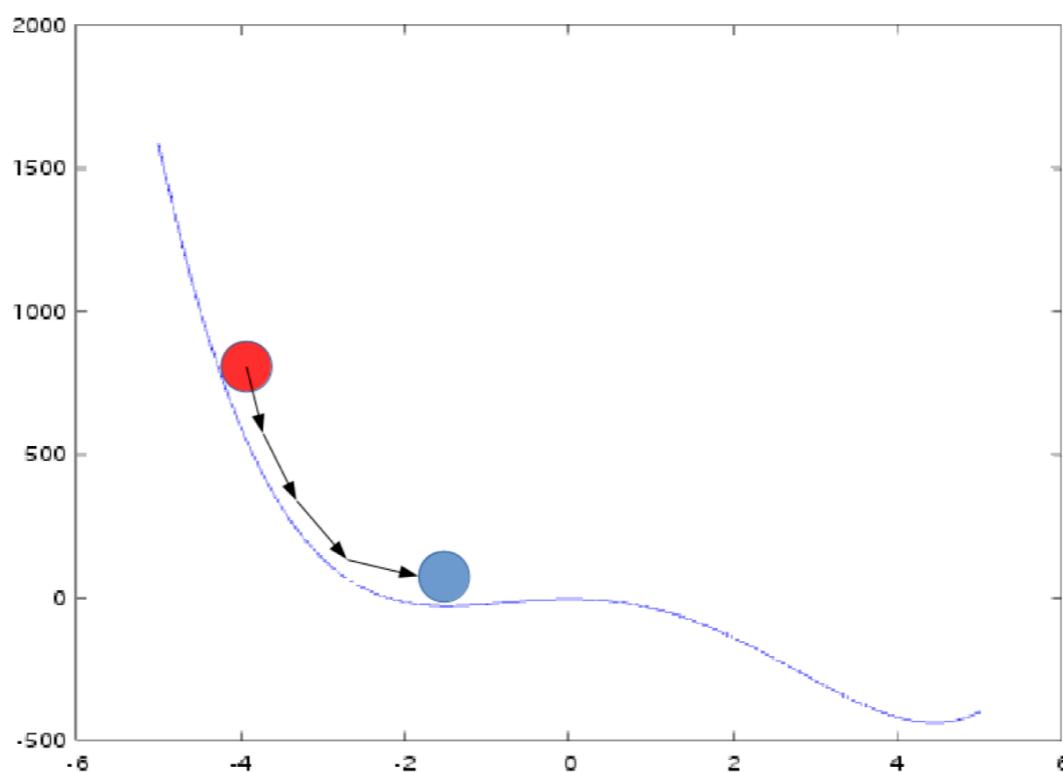
### 参考情報

<http://postd.cc/optimizing-gradient-descent/>

## 学習の停滞に関する対応策 (2)

### 大量の反復計算を行う (量で勝負!)

- ▶ 停留点付近であっても、大量の反復計算（ミニバッチ処理）を行うことで、停留点付近から脱出することが可能（かもしれない）。
- ▶ 大量のデータが必要（google, amazon は強い！）
- ▶ 高速な計算が必要（特に大きな行列の積を高速に計算する必要がある）→ GPU 向けの計算となっている（並列計算）



大量のデータは、「汎化誤差」を小さくするためにも本質的に重要

# 深層学習向けのハードウェア

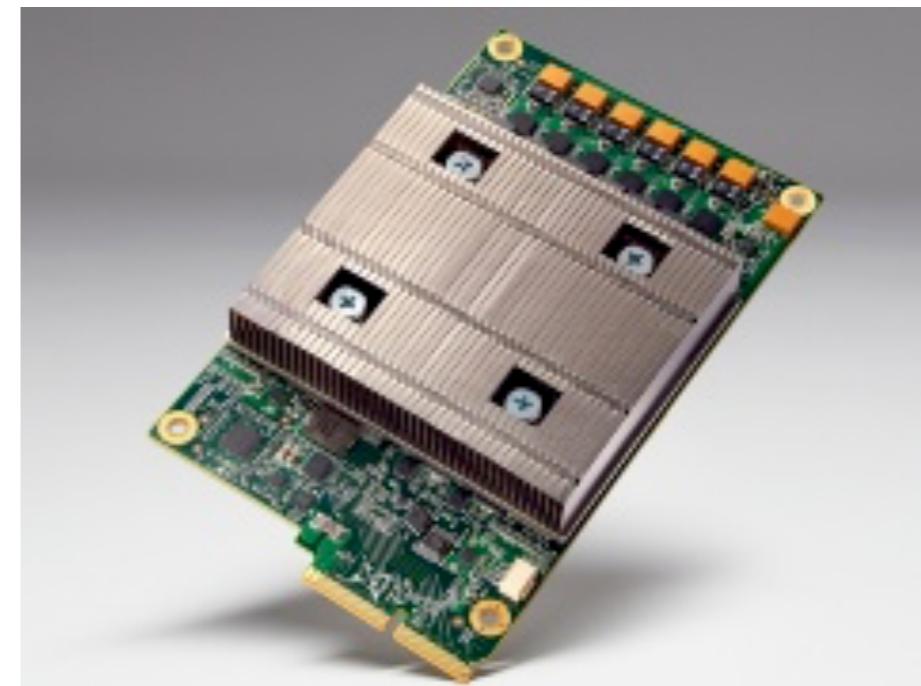
推論フェーズ（前向き計算）、逆伝播フェーズ（後ろ向き計算）のいずれにも大量の行列ベクトル積計算が必要とされる。汎用 CPU 以上に効率の良い計算が可能な GPU、または、専用のハードウェアへの注目が集まっている。

NVIDIA TESLA GPU



出典 <http://www.nvidia.co.jp/object/tesla-servers-jp.html>

Google Tensor Processing Unit (TPU)



出典 <http://itpro.nikkeibp.co.jp/atcl/ncd/14/457163/052001464/>

最低でも 1080ti 1枚程度のGPUパワーがないと実験などを行うのはツライ！

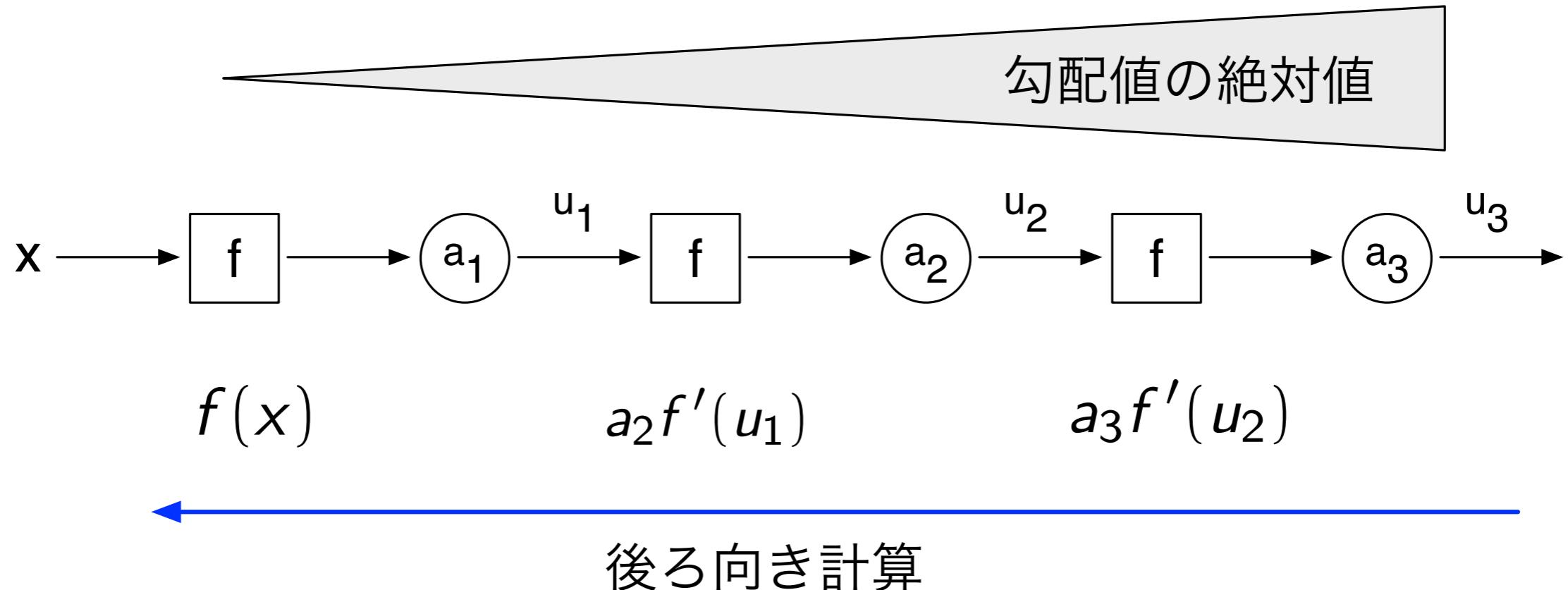
# DL計算環境について

	コスト	メリット	デメリット
Google Colabratoty	無料	環境構築不要	1~2時間以上の連続計算ができない
AWS, GCPなどクラウドを借りる	有料	環境構築ボタンひとつ・メンテ不要	GPU付きインスタンスを借りるとそこそこのお値段
GPUつきマシンを買う	初期投資が必要	手元に実行環境があるのは色々快適	GPUの稼働率はそれほど高くない。。。
CPU	手元のCPUを利用すればタダ	気軽に試せる	遅い(相対的に)

# 勾配消失問題

勾配消失問題とは、入力層に近い層の勾配値がほとんどゼロになってしまい、入力層に近い部分に位置する学習パラメータの更新がほとんど進まない現象を指す。

$$\frac{\partial u_3}{\partial a_1} = a_3 f'(u_2) a_2 f'(u_1) f(x) \quad (22)$$



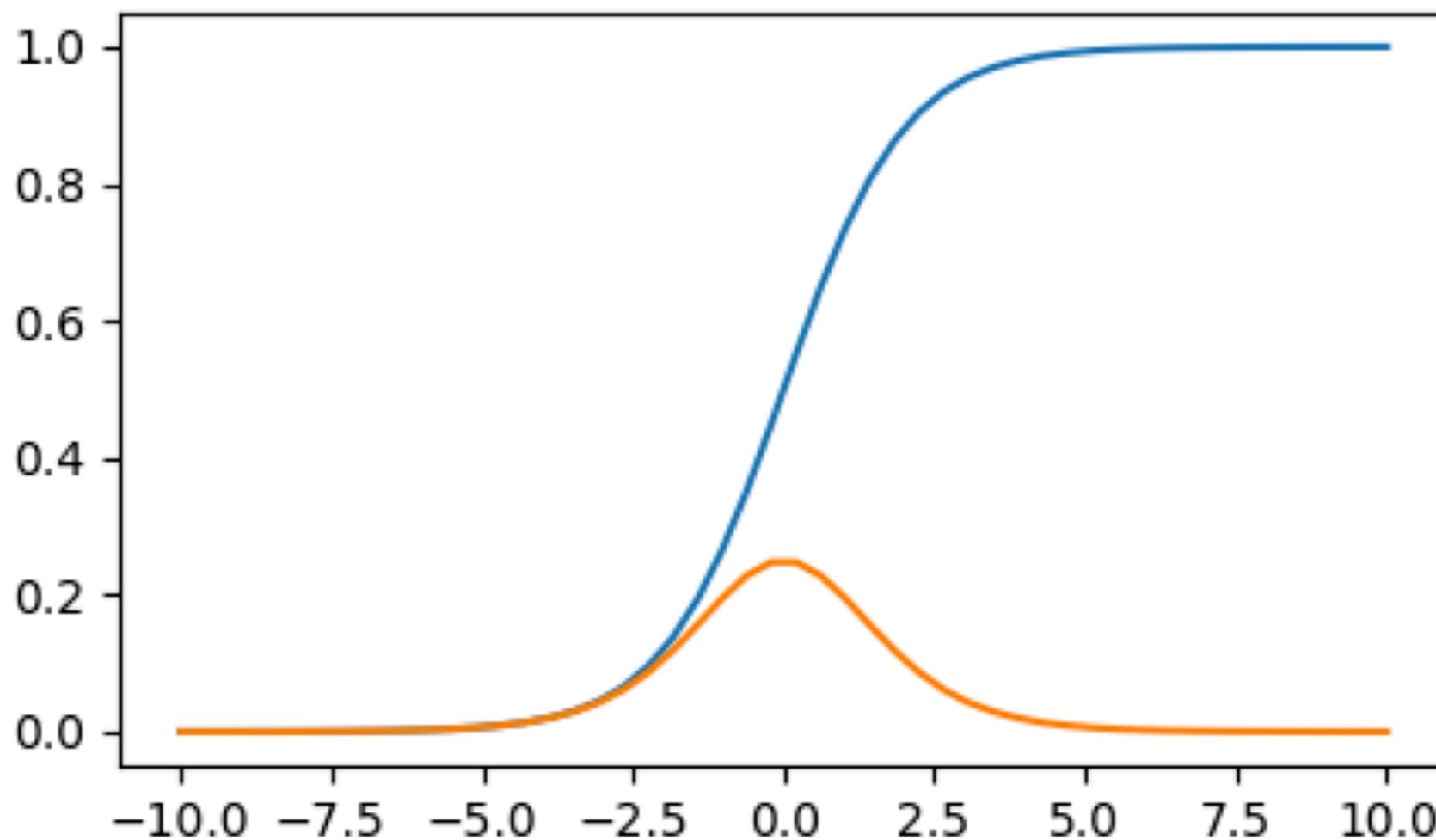
# シグモイド関数とその微分

シグモイド関数

$$f(x) = \frac{1}{1 + e^{-x}}$$

シグモイド関数の導関数

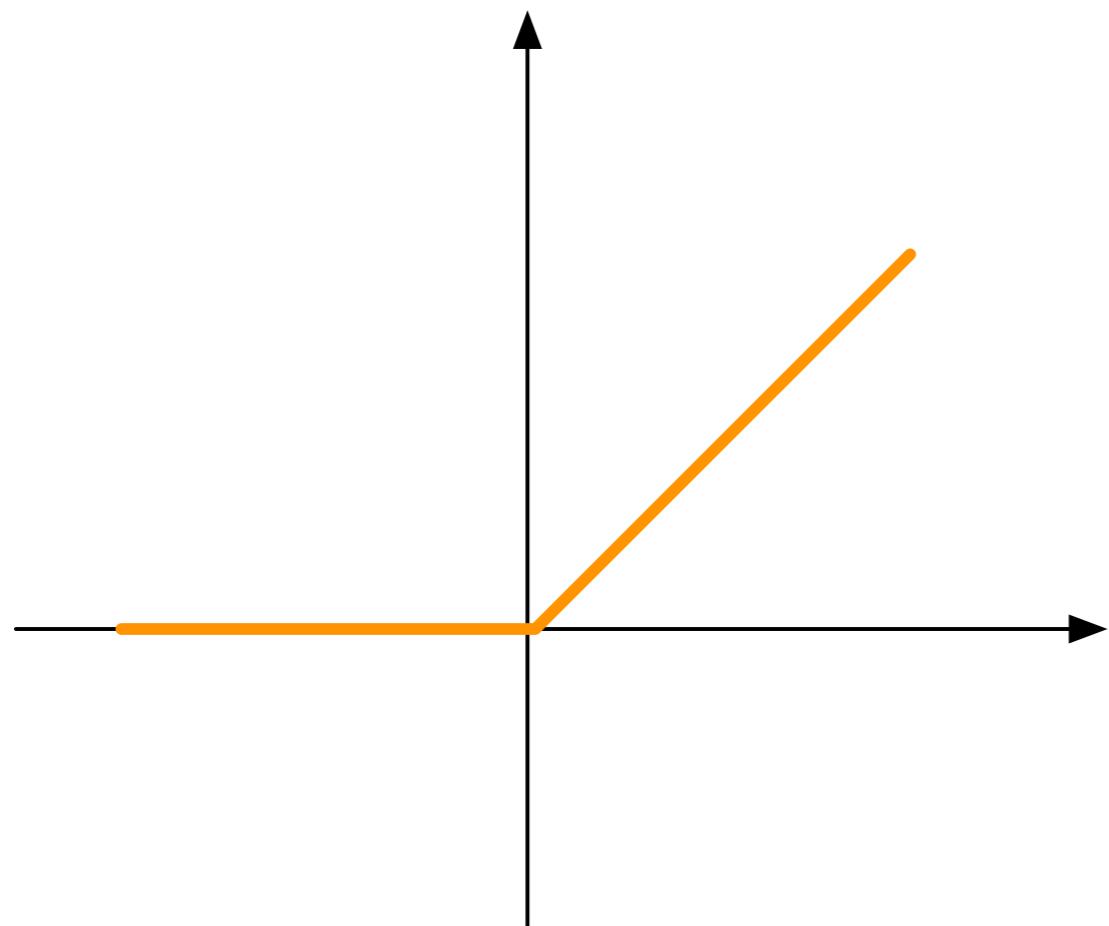
$$f'(x) = (1 - f(x))f(x)$$



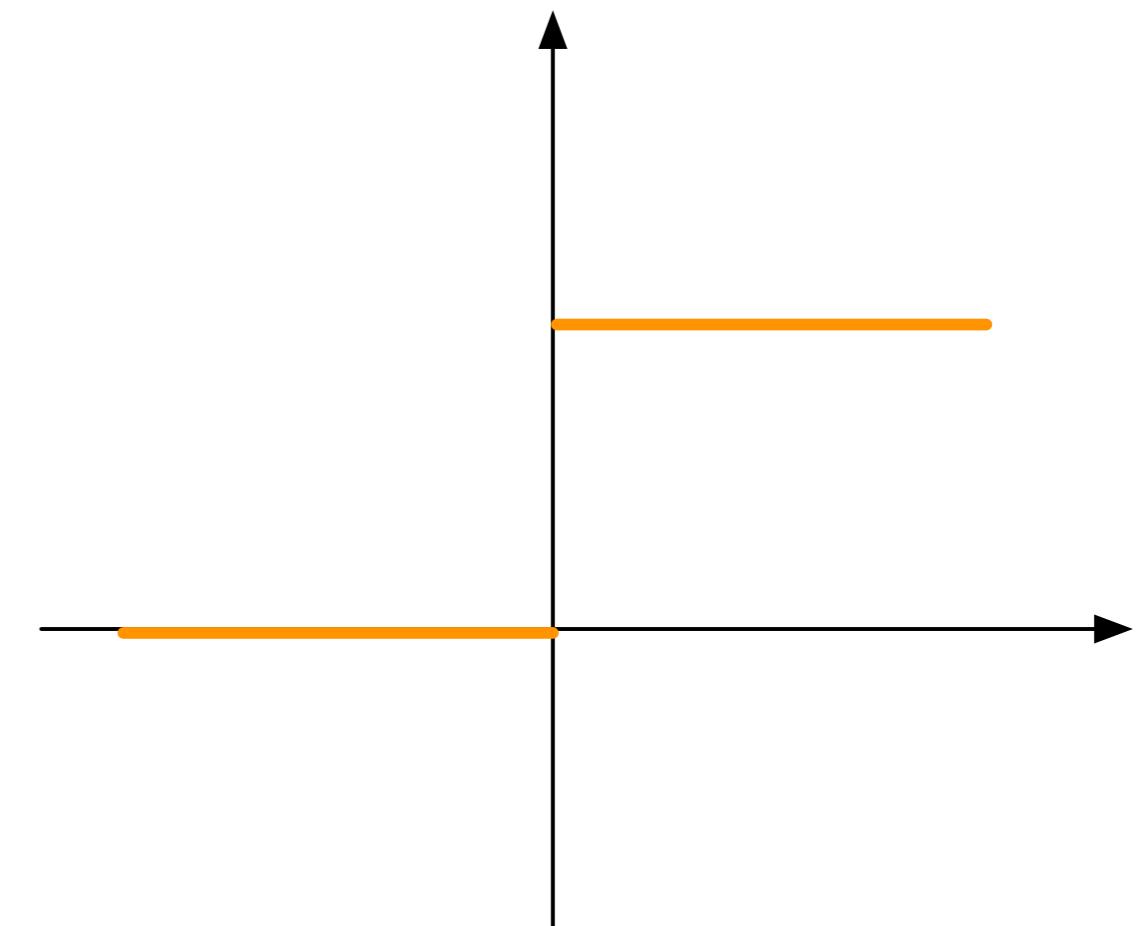
# 勾配消失問題への対策 (1)

## ReLU 関数 (ランプ関数) の利用

ReLU関数



ReLU関数の導関数

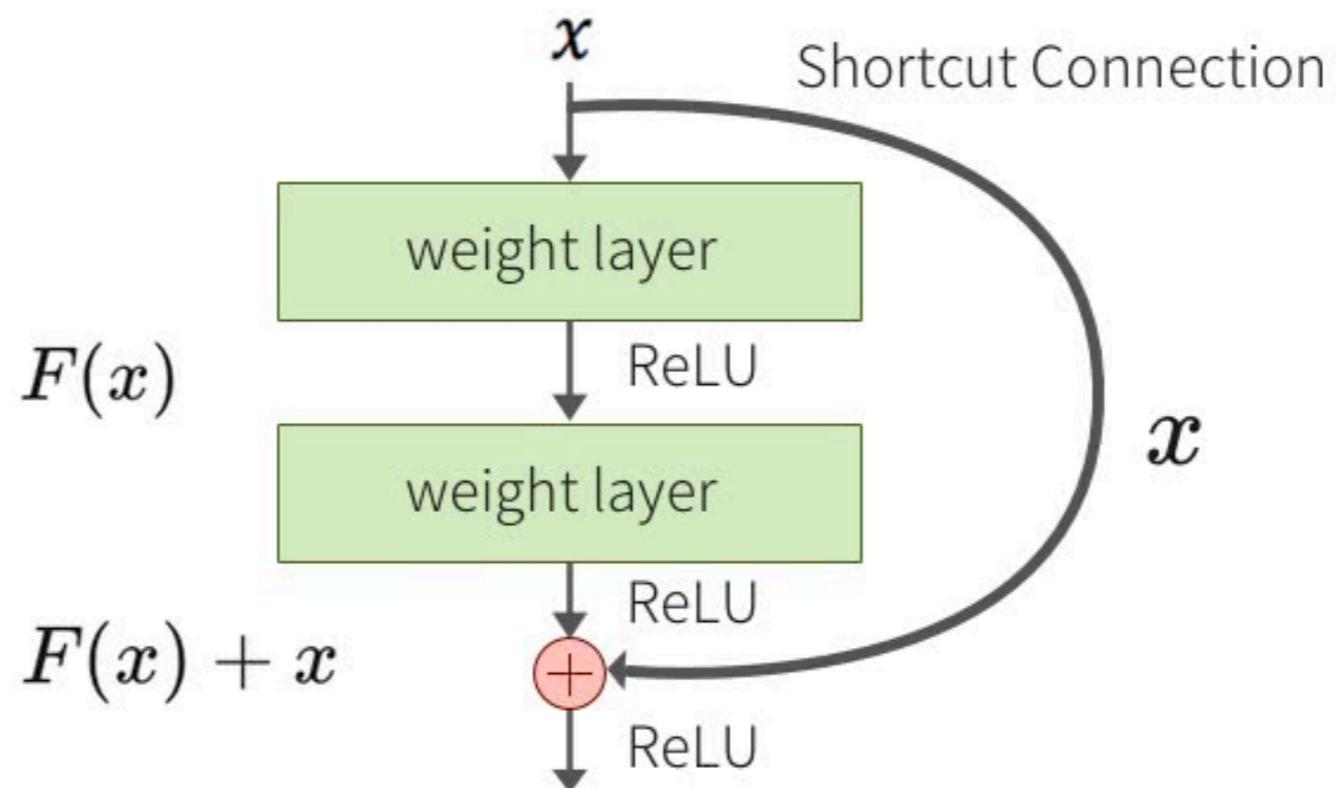


逆伝播フェーズにおける積の値が急速に小さくなることはなくなる。  
→多層(深層)の学習が容易に

# 勾配消失問題への対策 (2)

## Residual network の利用

ある層で求める最適な出力を学習するのではなく、層の入力を参照した残差関数を学習する、 という考え方に基づく。



出典 [https://deeppage.net/deep\\_learning/2016/11/30/resnet.html](https://deeppage.net/deep_learning/2016/11/30/resnet.html)

直感的説明: 逆伝播フェーズにおいて、ショートカットパスに沿って絶対値の大きな勾配情報が流れる

# 深層学習技術の分類

A. 関数近似のための  
最適化技術      B. 汎化誤差を小さくする  
ための技術      C. 表現学習の  
ための技術

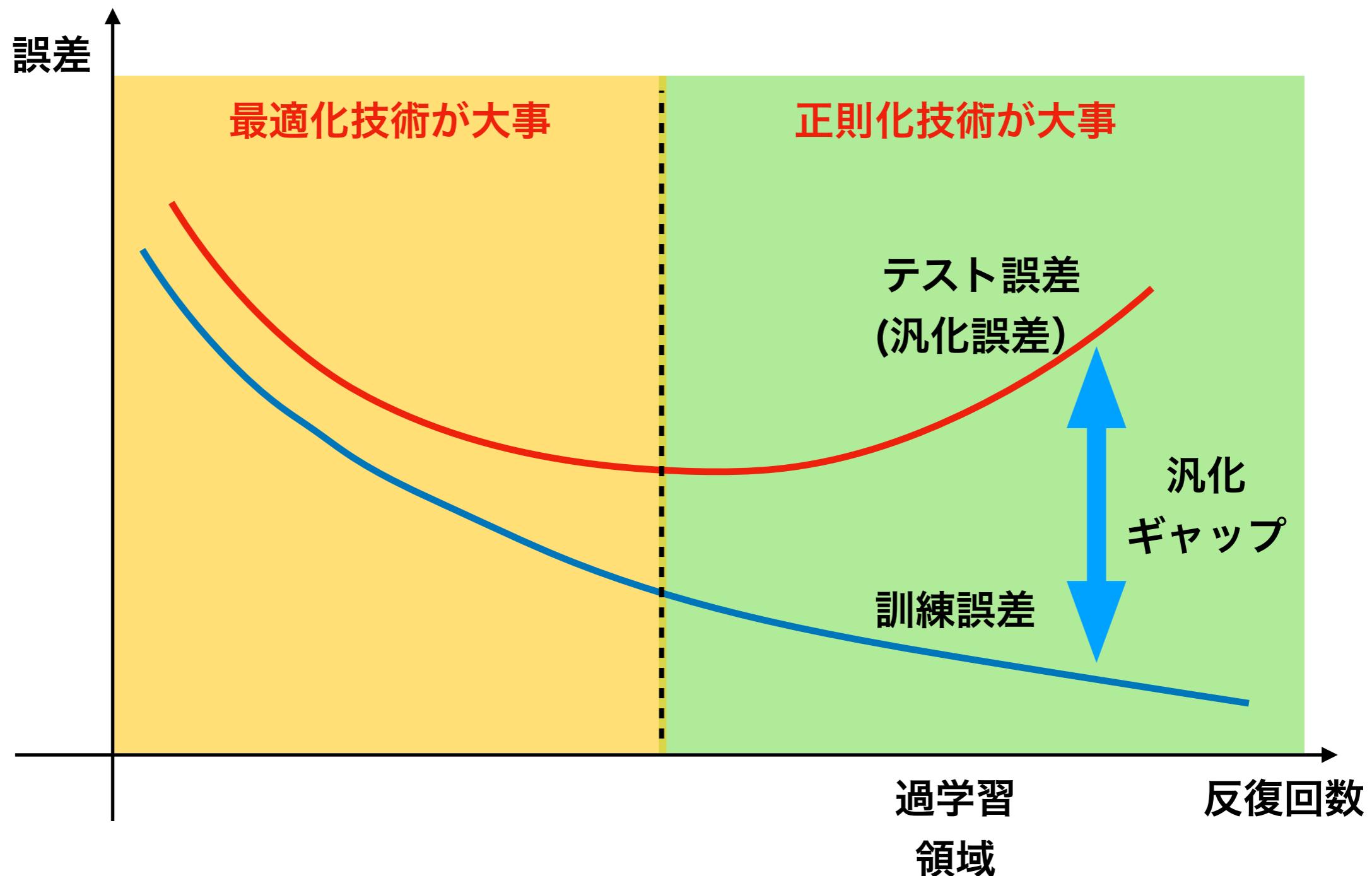
確率的勾配法  
誤差逆伝播法  
勾配消失を抑制  
する活性化関数

正則化  
重み共有  
ドロップアウト  
バッチ正規化  
ビッグデータ  
データ拡張

畳み込み  
ネットワーク  
重み共有  
ネットワーク構造  
埋め込み

# 汎化誤差

訓練データに含まれない初見のデータに対して適切な出力を与えることが望ましい



# DLフレームワーク

- PyTorchによるプログラム例



```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 2)
        self.fc2 = nn.Linear(2, 2)
    def forward(self, x):
        x = F.sigmoid(self.fc1(x))
        x = F.sigmoid(self.fc2(x))
        return x
```

**ネットワークの定義部**

順方向計算のみを記述すればよい。

誤差逆伝播法の後ろ向き計算フェーズは明示的にユーザが書く必要ない

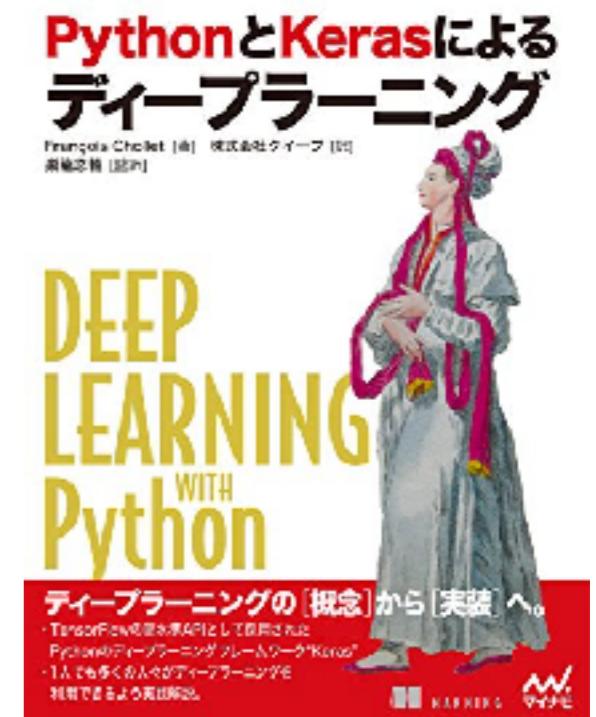
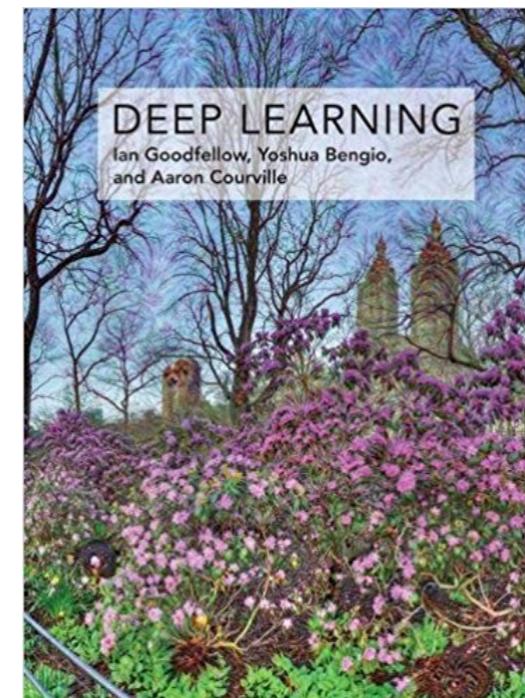
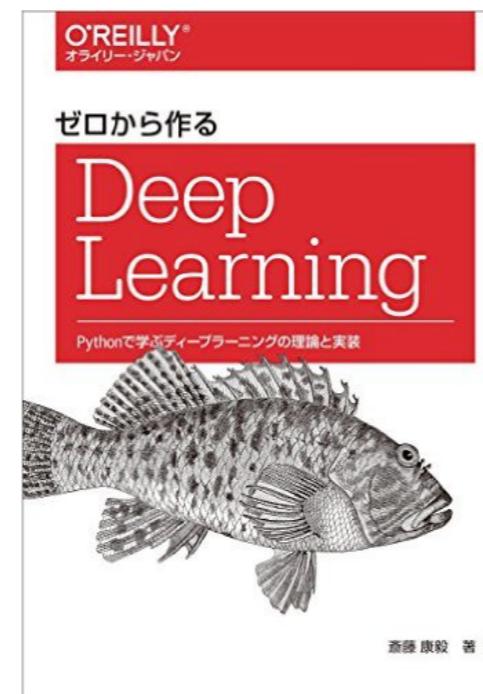
GPUの利用も非常に簡単。  
深層学習以外の分野でも有用！

model = Net() ネットワークのインスタンス化

loss\_func = nn.MSELoss() 損失関数の指定

optimizer = optim.Adam(model.parameters(), lr=0.1) オプティマイザの指定

# 深層学習を学ぶためのリソース



fast.ai <http://course.fast.ai/>

A brief introduction to ML for engineers (arXiv 版あります)

<https://bit.ly/2sdNEJ4>

# 深層学習を学ぶコツ

- 薄く記述の正確な本(例:岡谷本)で概要を掴む
- 世間の評判を調査し、フレームワークを選択  
(TensorFlow, PyTorch, Chainerの3択)
- 簡単なプログラムを作成（コードをgithubから拾ってきてまず写経でもよい）実行してみる
- Goodfellow本から興味ある部分を読んでいく

# Google Colaboratoryを使ってみる

- Jupyter notebookベースの機械学習環境
- 環境設定不要なので気軽に機械学習の学習・研究に利用できる

# Google Colabratoty



機械学習の教育と研究を促進するために Google  
が開発したツール



無償でクラウド上のマシンを利用可能



ブラウザベース



環境構築不要でフレームワークが使える



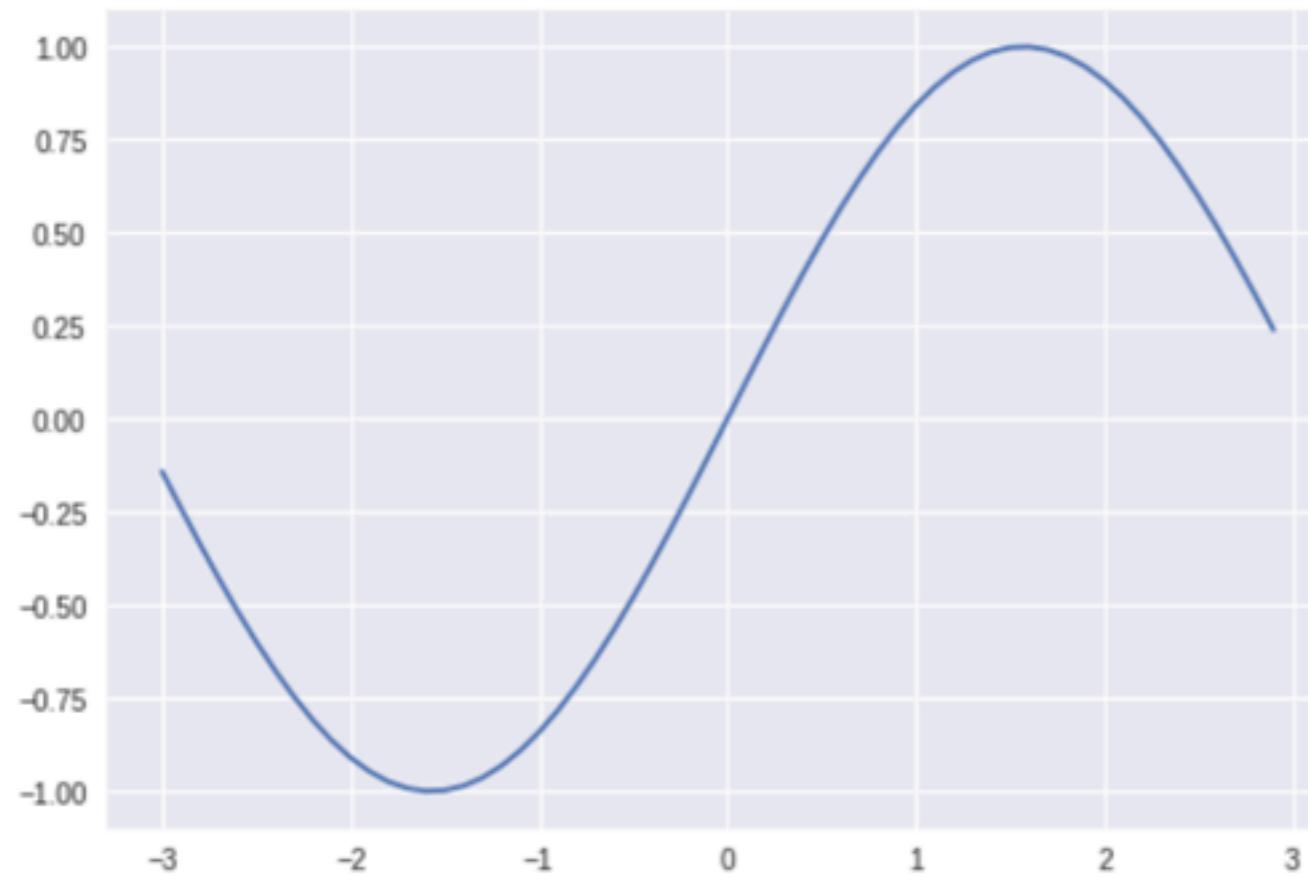
GPUが利用できる



```
[3] import numpy as np
     import matplotlib.pyplot as plt

     x = np.arange(-3, 3, 0.1)
     y = np.sin(x)
     plt.plot(x, y)
```

↳ [<matplotlib.lines.Line2D at 0x7fdea61a9470>]



# 利用できる環境

-  必要なものはブラウザとgoogleアカウントのみ
-  Python 2.6/3.6 が利用可能
-  機械学習に必要なライブラリはだいたい  
プリインストール済み
-  pipなどをを利用して自由にライブラリを導入,  
更新可能

# 使ってみよう

ブラウザで下記のURLにアクセスする

<https://colab.research.google.com/>

windowsユーザはchromeを使ってください。  
macユーザはsafariでOK

詳細な情報は次のQiitaの記事が詳しいです：  
【秒速で無料GPUを使う】深層学習実践Tips on

Colaboratory

[https://qiita.com/tomo\\_makes/items/b3c60b10f7b25a0a5935](https://qiita.com/tomo_makes/items/b3c60b10f7b25a0a5935)

<https://bit.ly/2sdNEJ4>

# PYTHON 3 の新しいノートブックをクリック

Colaboratory へようこそ

Colaboratory は、機械学習の教育や研究の促進を目的とした Google 研究プロジェクトです。完全にクラウドで実行される Jupyter ノートブック環境なしにご利用いただけます。

Colaboratory ノートブックは [Google ドライブ](#) に保存され、Google ドキュメントや Google スプレッドシートと同じように共有できます。Colaboratory ノートブックは、Python 3 で書かれた Jupyter ノートブックです。

例 最近のノートブック GOOGLE ドライブ GITHUB

ノートブックを挿入する

タイトル	最初に開いた日時	最終閲覧
Hello, Colaboratory	2018年2月6日 0分前	
Untitled7.ipynb	15 分前	15 分前
backprop.ipynb	2018年5月9日	2018年5月11日
Untitled6.ipynb	2018年5月9日	2018年5月9日
TISTA_sample.ipynb	2018年3月7日	2018年5月9日

PYTHON 3 の新しいノートブック キャンセル

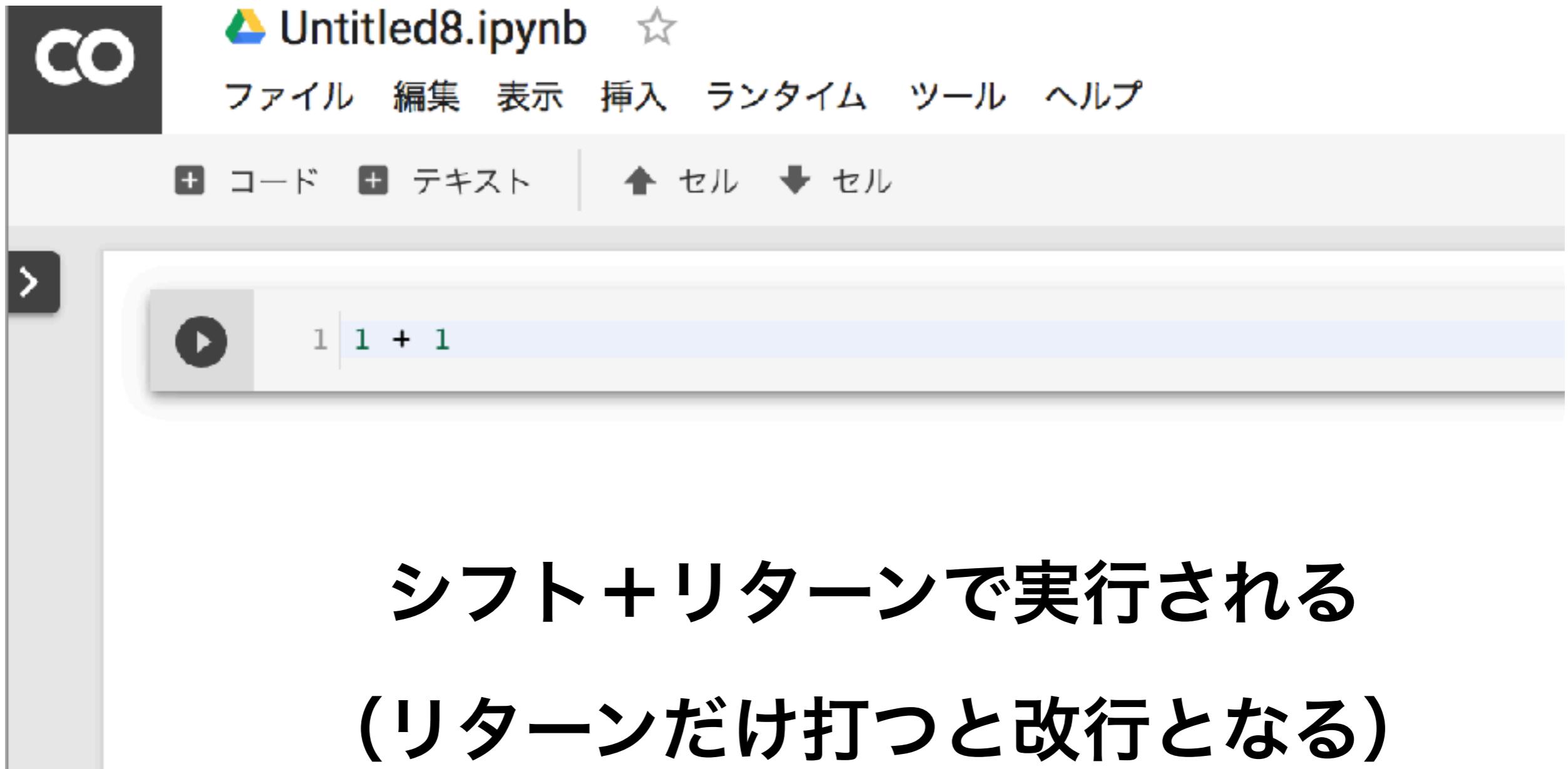
```
2 import numpy as np
3
4 with tf.Session():
5     input1 = tf.constant(1.0, shape=[2, 3])
```

# 新しいノートブックが開かれる

## プルダウンメニュー



# セルに入力してみる



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Untitled8.ipynb
- Menu Bar:** ファイル 編集 表示 挿入 ランタイム ツール ヘルプ
- Toolbar Buttons:** + コード, + テキスト, ↑ セル, ↓ セル
- Cell Area:** A code cell containing the expression `1 | 1 + 1`. The cell is currently in "Code" mode, indicated by the highlighted button.
- Output Area:** Not visible in the screenshot.

**Text Below Screenshot:**

シフト+リターンで実行される  
(リターンだけ打つと改行となる)



## Untitled8.ipynb ☆

ファイル 編集 表示 挿入 ランタイム ツール ヘルプ

+ コード + テキスト | ↑ セル ↓ セル

[1] 1 | 1 + 1

⇨ 2

[2] 1 | 10 \* 3.14159

⇨ 31.4159



1 |

# for 文を使ってみる

The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar includes a 'Untitled8.ipynb' tab, a star icon, and a menu bar with Japanese labels: ファイル, 編集, 表示, 挿入, ランタイム, ツール, ヘルプ. Below the menu is a toolbar with buttons for 'コード' (Code) and 'テキスト' (Text), and cell navigation arrows.

**Cell 1:**

```
[1] 1 | 1 + 1
```

Output:

```
2
```

**Cell 2:**

```
[2] 1 | 10 * 3.14159
```

Output:

```
31.4159
```

**Cell 3:**

```
▶ 1 | for i in range(5):  
 2 |   print(i)
```

Output:

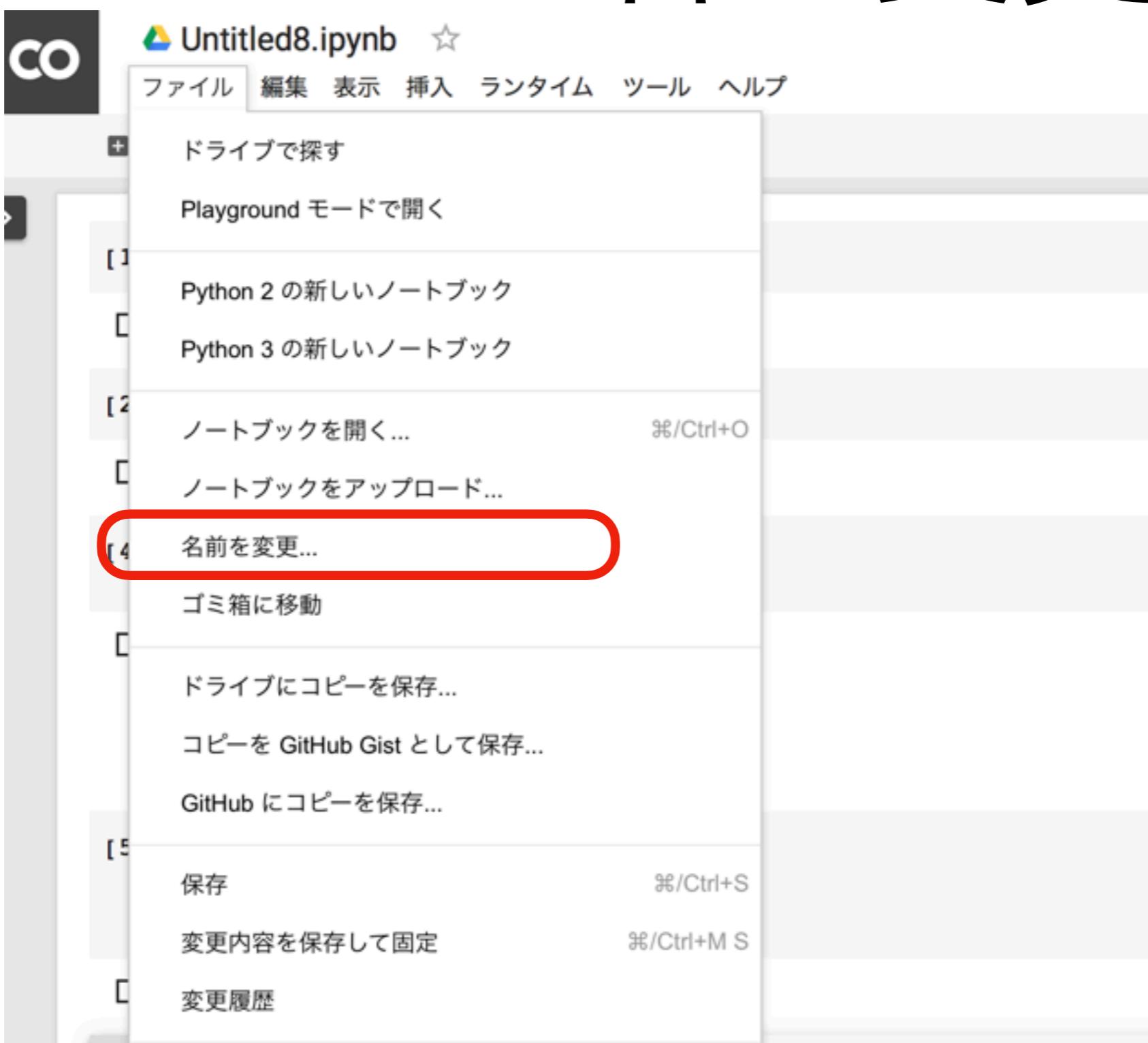
```
0  
1  
2  
3  
4
```

# for文を使ってみる

```
[5] 1 sum = 0
     2 for i in range(101):
     3     sum = sum + i
     4 print(sum)
```

```
→ 5050
```

# ファイル名の変更

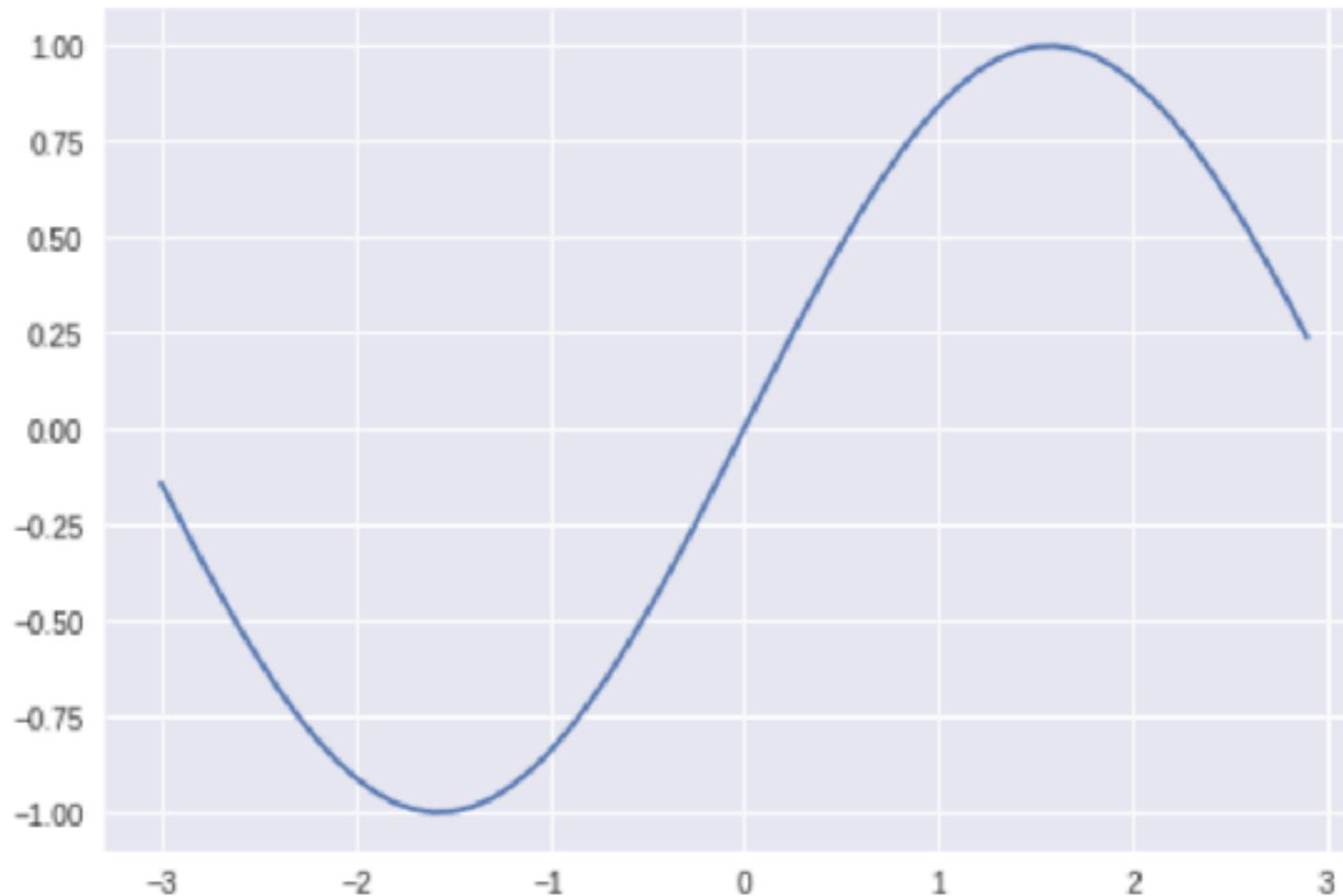


# 図をプロットしてみる

numpy をimport  
matplotlibを  
import

```
[8] 1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x = np.arange(-3, 3, 0.1)
5 y = np.sin(x)
6 plt.plot(x,y)
```

↳ [〈matplotlib.lines.Line2D at 0x7fa8c42af3c8〉]



# 最小化の手法: 勾配法 (gradient descent method)

## 制約無し最適化問題

$$\underset{\mathbf{x} \in \mathbb{R}^n}{\text{minimize}} \ f(\mathbf{x}) \quad (11)$$

### 勾配法のステップ

Step 1 (初期点設定)  $\mathbf{x} := \mathbf{x}_0$

Step 2 (勾配の計算)  $\mathbf{g} := \nabla f(\mathbf{x})$

Step 3 (探索点更新)  $\mathbf{x} := \mathbf{x} - \alpha \mathbf{g}$  ( $\alpha$  は学習係数)

Step 4 (反復) Step 2 に戻る

(注)

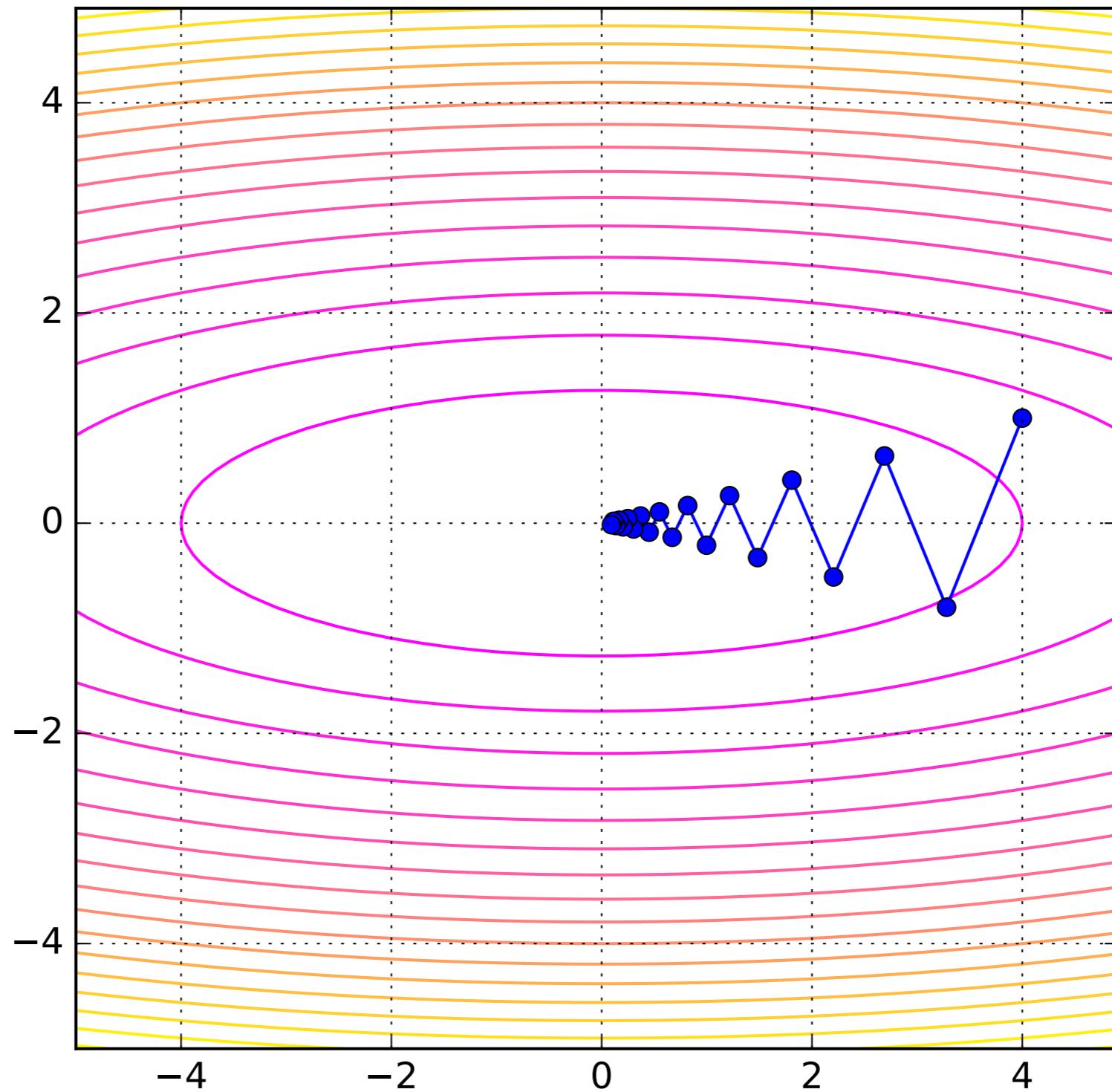
- ▶  $\nabla f(\mathbf{x})$  は勾配ベクトル (gradient vector) である。例えば、 $f(x_0, x_1)$  の場合、

$$\nabla f(x_0, x_1) = \left( \frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1} \right)^T$$

である。勾配ベクトルは  $f$  の等高線に直交する。

# 最小化の軌跡

$$f(\mathbf{x}) = \frac{1}{2}x_0^2 + 5x_1^2, \quad \mathbf{x}_0 = (4, 1)^T, \alpha = 0.18$$



# 勾配法のコード

$x^2 + y^2$  の最小化



```
1 def grad_x(x):
2     return 2.0 * x
3 def grad_y(y):
4     return 2.0 * y
5 xt = 5
6 yt = 5
7 alpha = 0.1
8 for i in range(50):
9     xt = xt - alpha * grad_x(xt)
10    yt = yt - alpha * grad_y(yt)
11    print(i, xt, yt)
```

チュートリアルHPのgradient\_descent.ipynbをクリック  
「このプログラムgoogleが作ったものではありません。。。」  
などと言われるが構わずOKして進める

<https://bit.ly/2sdNEJ4>

# PyTorchの概要

## ライトニングpytorch入門

<https://qiita.com/sh-tatsuno/items/42fccff90c98103dffc9>

とりあえず「backpropのできるmatlabっぽいもの」という理解でも結構です

テンソル=多次元の行列

```
[17] 1 | x = torch.zeros(5, 3)
```

```
[18] 1 | print(x)
```

```
↪  
 0  0  0  
 0  0  0  
 0  0  0  
 0  0  0  
 0  0  0  
[torch.FloatTensor of size 5x3]
```

```
[19] 1 | y = torch.rand(5, 3)
```

```
[20] 1 | print(y)
```

```
↪  
 0.5924  0.1891  0.3995  
 0.5054  0.4481  0.6360  
 0.5679  0.2776  0.2153  
 0.9880  0.2935  0.1306  
 0.1766  0.8389  0.3627  
[torch.FloatTensor of size 5x3]
```

```
[21] 1 | z = 2 * y
```

```
[22] 1 | print(z)
```

```
↪  
 1.1847  0.3782  0.7989  
 1.0109  0.8962  1.2720  
 1.1358  0.5552  0.4305  
 1.9760  0.5871  0.2613  
 0.3533  1.6777  0.7255  
[torch.FloatTensor of size 5x3]
```

# ニューラルネットワーク学習プログラム



AND関数を学習するプログラム

チュートリアルHPのbackprop.ipynbをクリック

## PyTorchのインストール



```
1 # http://pytorch.org/
2 from os import path
3 from wheel.pep425tags import get_abbr_impl, get_impl_ver, get_abi_tag
4 platform = '{}{}-{}'.format(get_abbr_impl(), get_impl_ver(), get_abi_tag())
5
6 accelerator = 'cu80' if path.exists('/opt/bin/nvidia-smi') else 'cpu'
7
8 !pip install -q http://download.pytorch.org/whl/{accelerator}/torch-0.3.0.post
9 import torch
```

### ▼ 必要なパッケージのインポート

```
[ ] 1 import torch
2 from torch.autograd import Variable
3 import torch.nn as nn # ネットワーク構築用
4 import torch.optim as optim # 最適化関数
5 import torch.nn.functional as F # ネットワーク用の様々な関数
6
```

## ▼ グローバル変数の定義

```
[ ] 1 mb_size = 10
```

## ▼ ニューラルネットワークの定義

```
[ ] 1
2 class Net(nn.Module):
3     def __init__(self):
4         super(Net, self).__init__()
5         self.fc1 = nn.Linear(2, 2) # 名前はlinear だが Ax + b のアフィン変換の形
6         self.fc2 = nn.Linear(2, 2)
7
8     def forward(self, x):
9         x = F.sigmoid(self.fc1(x))
10        x = F.sigmoid(self.fc2(x))
11        return x
12
```

## ▼ インスタンス作成

```
[ ] 1 model = Net() # モデルインスタンス作成
2
3 # Loss関数の指定
4 loss_func = nn.MSELoss()
5
6 # Optimizerの指定
7 optimizer = optim.Adam(model.parameters(), lr=0.1)
8
9 # データ全てのトータルロス
10 running_loss = 0.0
```

## ▼ 訓練ループ

```
[ ] 1 # 訓練ループ
2 for i in range(1000):
3     # フィードするデータの作成
4     inputs = torch.bernoulli(0.5 * torch.ones(mb_size, 2)) # 確率0.5で1となるペルヌーイ分布
5     labels = torch.Tensor(mb_size, 2)
6     for j in range(mb_size):
7         if (inputs[j, 0] == 1.0) and (inputs[j, 1] == 1.0):
8             labels[j, 0] = 1.0
9             labels[j, 1] = 0.0
10        else:
11            labels[j, 0] = 0.0
12            labels[j, 1] = 1.0
13
14    # Variableに変形(モデルに入力するときはvariableにする)
15    inputs, labels = Variable(inputs), Variable(labels)
16
17    optimizer.zero_grad() # optimizerの初期化
18
19    outputs = model(inputs) # 推論計算
20    loss = loss_func(outputs, labels) # 損失関数の定義
21
22    loss.backward() # バックプロパゲーション
23    optimizer.step() # パラメータ更新
24
25    # ロスの表示
26    # print statistics
27    running_loss += loss.data[0]
28    if i % 10 == 9: # print every 10 mini-batches
29        print('[%d] loss: %.3f' %
30              (i + 1, running_loss / 10))
31    running_loss = 0.0
```

# 損失関数値がどんどん小さくなっていく！

```
↳ [ 10] loss: 0.203  
[ 20] loss: 0.192  
[ 30] loss: 0.134  
[ 40] loss: 0.147  
[ 50] loss: 0.103  
[ 60] loss: 0.078  
[ 70] loss: 0.041  
[ 80] loss: 0.030  
[ 90] loss: 0.026  
[100] loss: 0.017  
[110] loss: 0.014  
[120] loss: 0.010  
[130] loss: 0.010  
[140] loss: 0.008  
[150] loss: 0.006  
[160] loss: 0.006  
[170] loss: 0.005  
[180] loss: 0.005  
[190] loss: 0.004  
[200] loss: 0.004  
[210] loss: 0.003  
[220] loss: 0.003  
[230] loss: 0.003  
[240] loss: 0.003  
[250] loss: 0.002  
[260] loss: 0.002  
[270] loss: 0.002  
[280] loss: 0.002  
[290] loss: 0.002  
[300] loss: 0.002  
[310] loss: 0.002  
... ...
```

## ▼ 学習結果の評価

```
[7] 1 # 性能の評価  
2 inputs = Variable(torch.Tensor(1, 2))  
3 inputs.data[0, 0] = 0.0  
4 inputs.data[0, 1] = 0.0  
5  
6 outputs = model(inputs)  
7 print('0 & 0 = %.4f' % (outputs.data[0, 0]))  
8  
9 inputs.data[0, 0] = 0.0  
10 inputs.data[0, 1] = 1.0  
11 outputs = model(inputs)  
12 print('0 & 1 = %.4f' % (outputs.data[0, 0]))  
13  
14 inputs.data[0, 0] = 1.0  
15 inputs.data[0, 1] = 0.0  
16 outputs = model(inputs)  
17 print('1 & 0 = %.4f' % (outputs.data[0, 0]))  
18  
19 inputs.data[0, 0] = 1.0  
20 inputs.data[0, 1] = 1.0  
21 outputs = model(inputs)  
22 print('1 & 1 = %.4f' % (outputs.data[0, 0]))  
23
```

```
↳ 0 & 0 = 0.0001  
0 & 1 = 0.0067  
1 & 0 = 0.0051  
1 & 1 = 0.9809
```

# MNIST文字認識

講義HPのmnist.ipynbをクリック

休憩です

# 無線物理層における深層学習

- 第1部：通信工学研究者のための深層学習クラッシュコース（前提知識を仮定せず）(約80分)
  - 深層学習の基本
  - 深層学習を学ぶためのリソース
  - Google Colaboratory & PyTorch 紹介
- 休憩（10分～15分）
- 第2部：無線物理層における深層学習(約80分)
  - 自己符号化器に基づくEnd-to-Endアプローチ
  - データ駆動アプローチに基づく反復アルゴリズムの改善
  - 研究事例の紹介

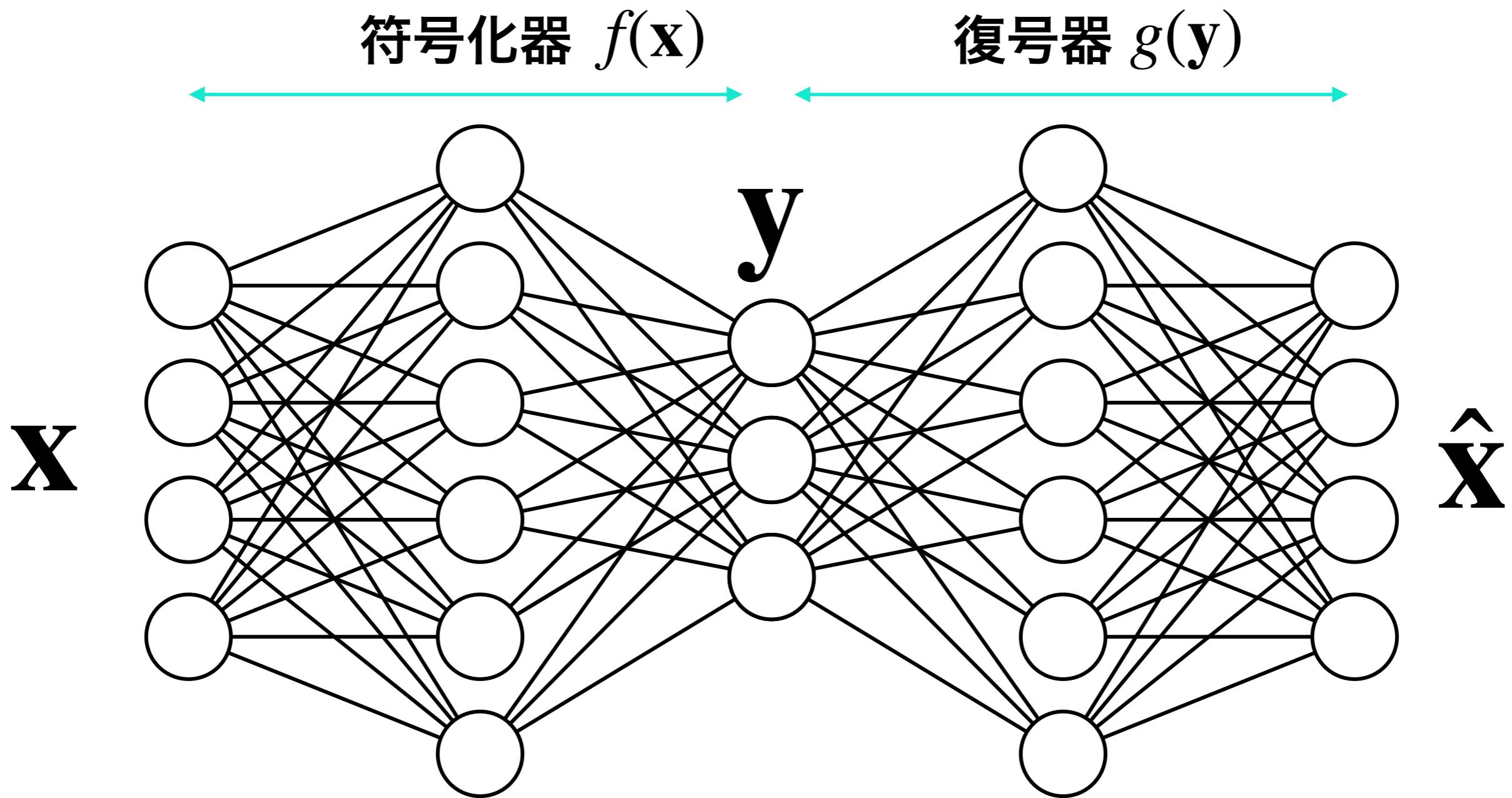
# 無線物理層における深層学習

- 深層学習技術は、AI・パターン認識等の分野にとどまらず工学における普遍性の高い技術であるという認識の広まり
- 無線物理層に残された課題と新しい可能性
  - モデリングの困難な通信系へのアプローチ
  - リアルタイム最適化が困難な資源配分やビームフォーミング
  - ブロックベース設計 vs End-to-End設計
  - データ駆動アプローチによる既知アルゴリズムの改善の可能性
  - 今後登場するニューラル計算特化型ハードウェアとの親和性（行列ベクトル積をベースとする信号処理）

# 自己符号化器に基づく End-to-Endアプローチ

- 通信路全体を自己符号化器でモデリング
- 通信路に関する事前知識や知見が不要

# 自己符号化器



$d(\mathbf{x}, \hat{\mathbf{x}})$ を小さくするようにネットワークを学習  
興味ある中間表現  $y$ を抽出

# 通信路モデル

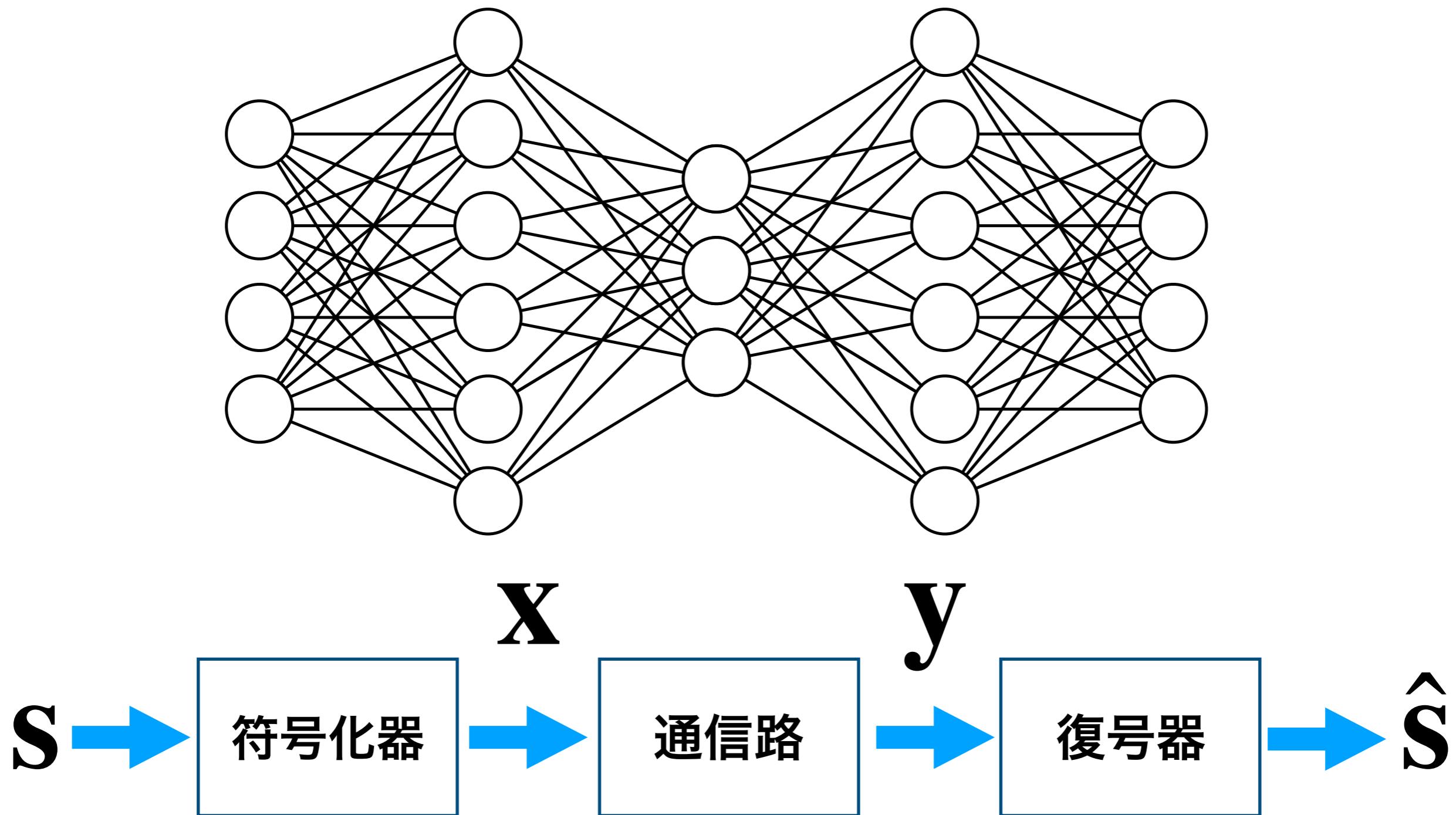


$$s \in \{1, 2, \dots, M\}$$

$$\mathbf{x} \in \mathbb{C}^n, \mathbb{E} \left[ \|\mathbf{x}\|_2^2 \leq n \right]$$

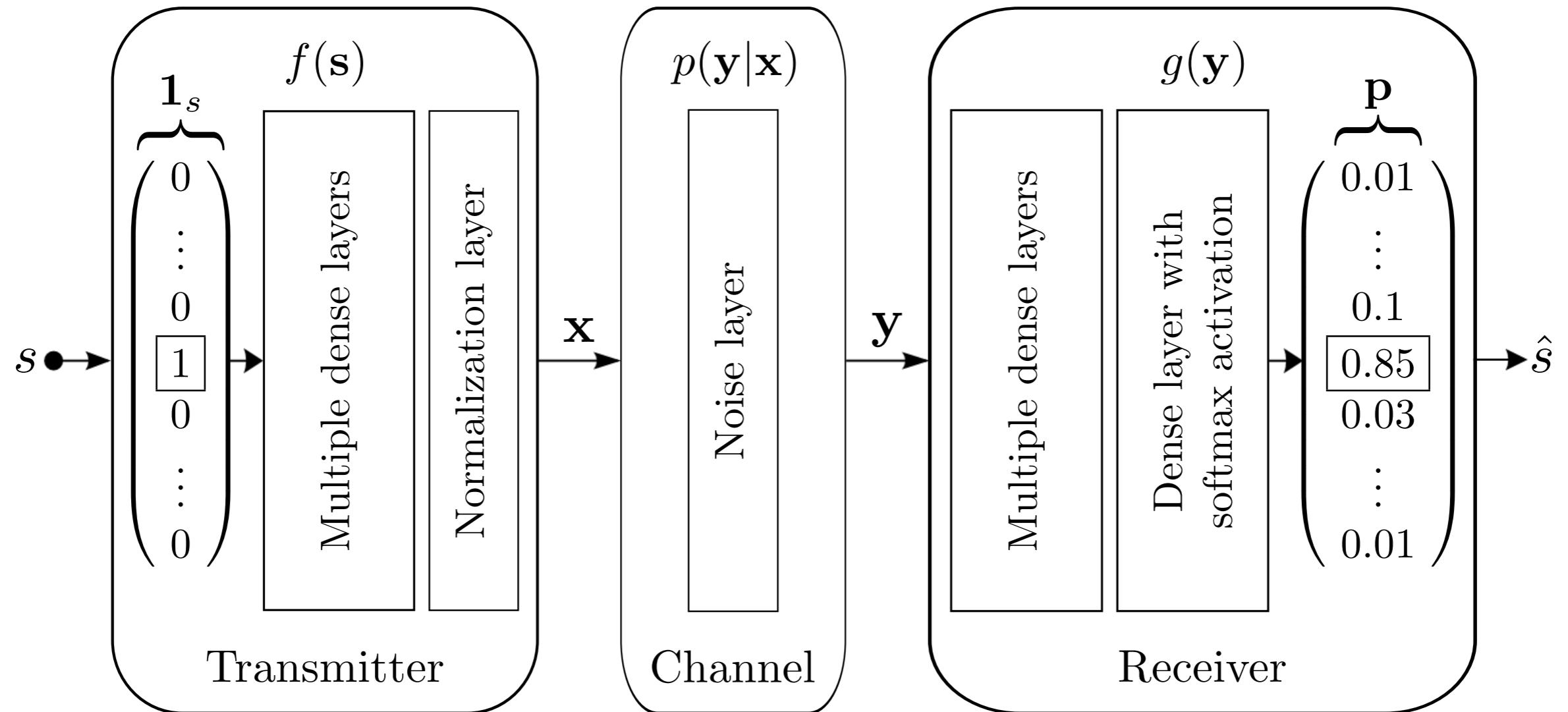
$$\mathbf{y} \sim p(\mathbf{y} | \mathbf{x})$$

# 自己符号化器に基づく通信系の表現



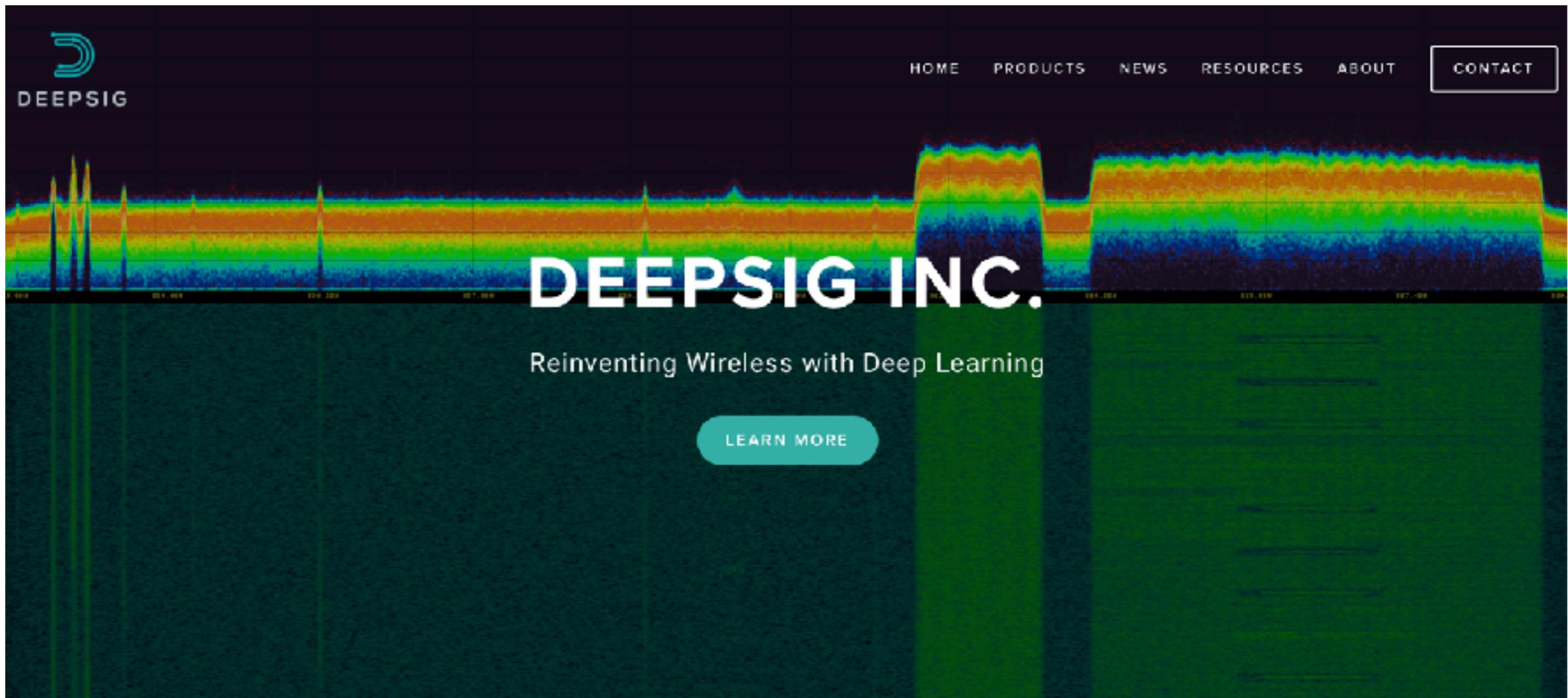
T.O'Shea, J. Hoydis, “An introduction to deep learning for the physical layer”  
IEEE Trans. Cog. Commun. Netw., Dec. 2017

# End-to-End アプローチ



Cited from: T.O'Shea, J. Hoydis, “An introduction to deep learning for the physical layer”

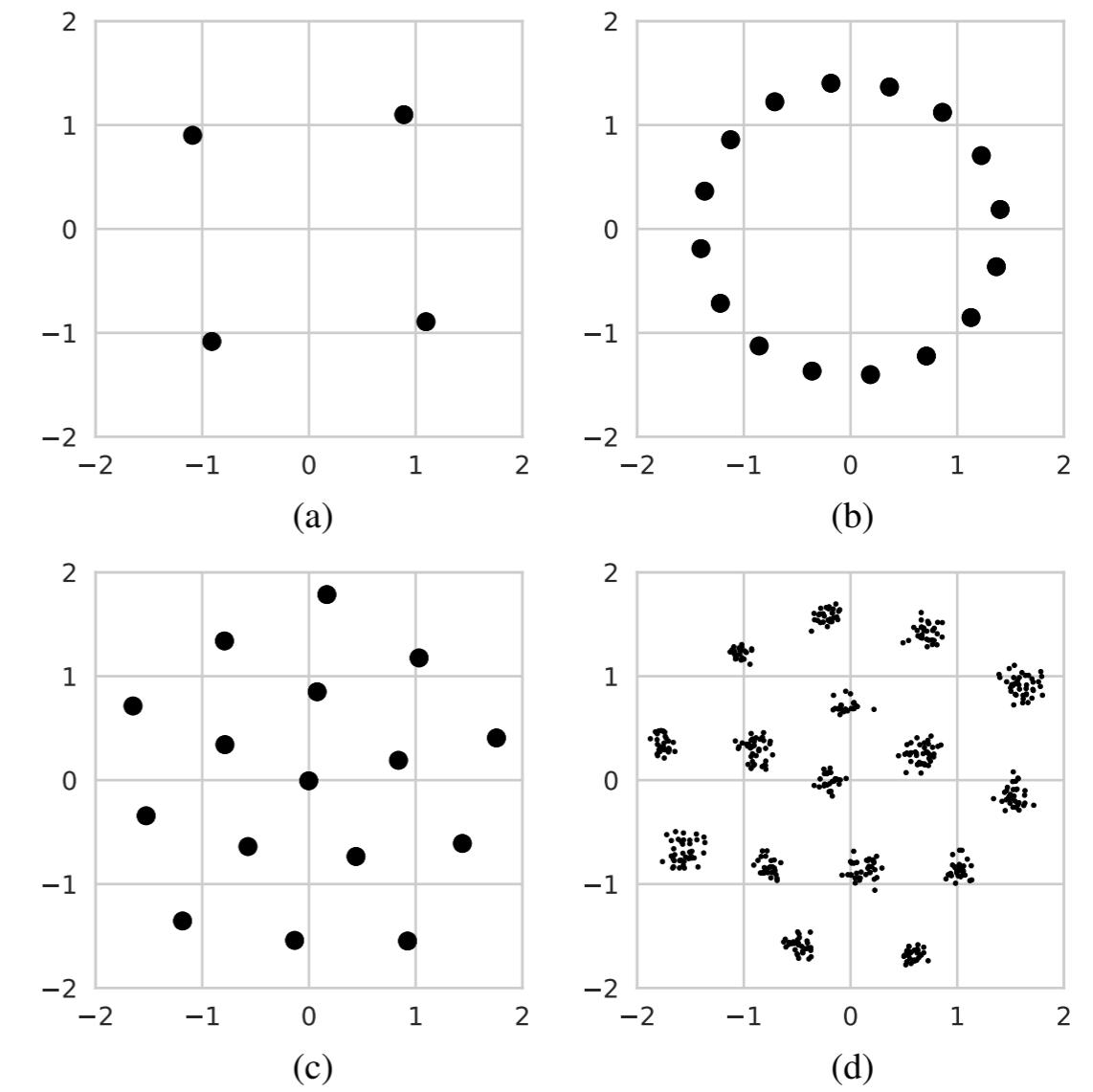
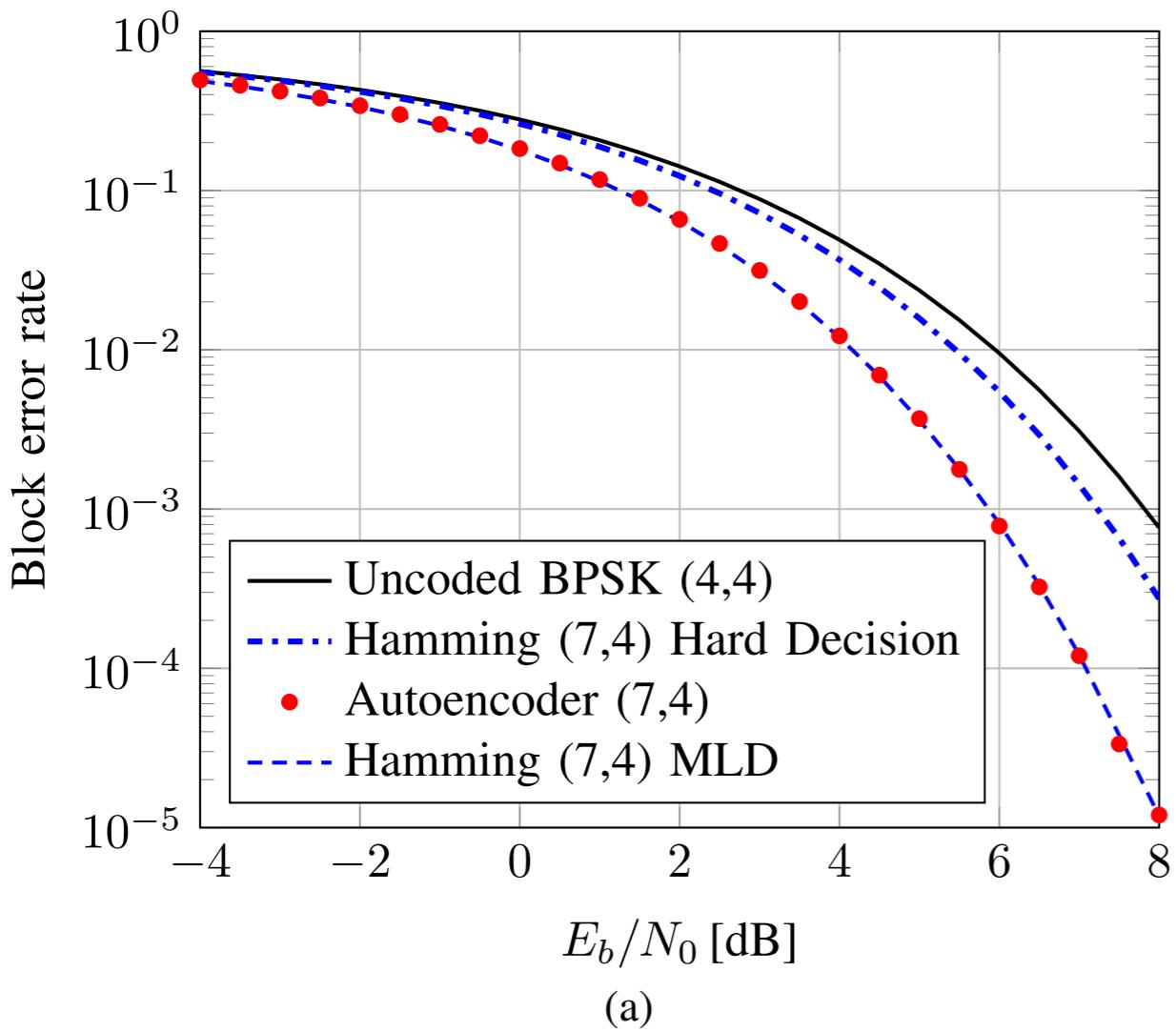
IEEE Trans. Cog. Commun. Netw., Dec. 2017



We are pioneering the application of deep learning to wireless.

Our approach to signal processing design uses AI to learn optimized models directly from data rather than manually designing specialized algorithms, creating communications systems that excel in complex environments.





Cited from: T.O'Shea, J. Hoydis, “An introduction to deep learning for the physical layer”  
 IEEE Trans. Cog. Commun. Netw., Dec. 2017

# 自己符号化器アプローチに関する補足

- 変調(信号点配置)設計手法とみることもできる
- 通信路に関する事前知識が必要ない
- $n$ が小さいところでは、既存方式に対して競争力がある。ただし、スケーラビリティが課題
- MAC, broadcast, 干渉通信路への適用も可能
- 実通信路をシミュレートする場合はバックプロパゲーションが利用できない→通信路をGANで近似

# 主な生成モデル(確率モデル)

	有名なモデル	特徴	メリット
敵対的生成モデル	GAN	生成ネットワークと識別ネットワークを競わせる	リアルな画像サンプルを生成
変分型	VAE	変分推論を用いる	サンプリングが高速
自己回帰モデル	NADE, Wavenet	同時分布のチェイン分解に基づく。対数尤度の最大化	対数尤度を計算可能・サンプリング也可能だが若干遅い
逆関数ベース（正規化フロー）	NICE, Glow	逆関数の取りやすい生成ネットワーク	サンプリング・対数尤度どちらにも対応できる

# データ駆動チューニング (Unfolding approach)

- 既存のアルゴリズムの処理を時間方向に展開 (unfolding)
- 深層学習技術により内部パラメータを調整 (チューニング)
- 収束速度の向上 (データ駆動加速)

# 可微分プログラミング



LeCunは、入出力を伴う処理(NNとは限らない)に対して、深層学習技術が適用可能であることを示唆している

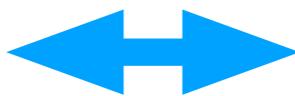
… important point is that people are now building a new kind of software by **assembling networks of parameterized functional blocks** and by training them from examples using some form of gradient-based optimization…



処理全体が微分可能であれば、内部に含まれるパラメータをbackprop+SGDで最適化できる  
→**可微分プログラミング**  
(differentiable programming )

# Software 2.0

ソースコード



データ

コンパイラ

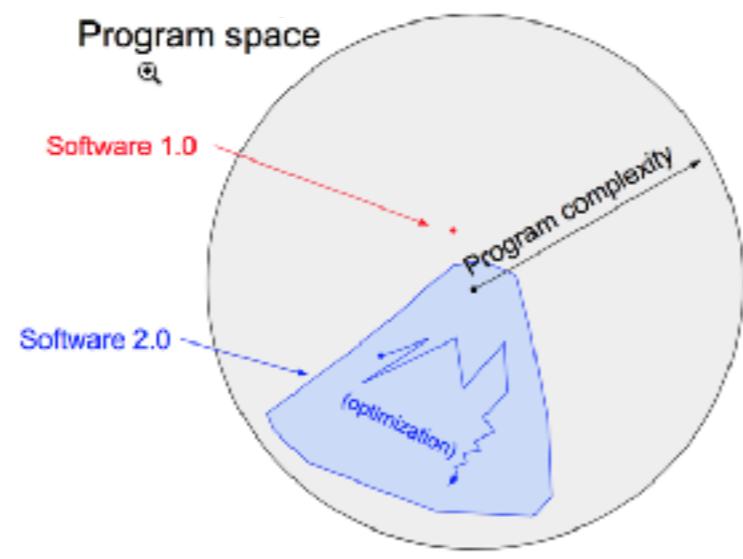


深層学習

実行形式



推論器

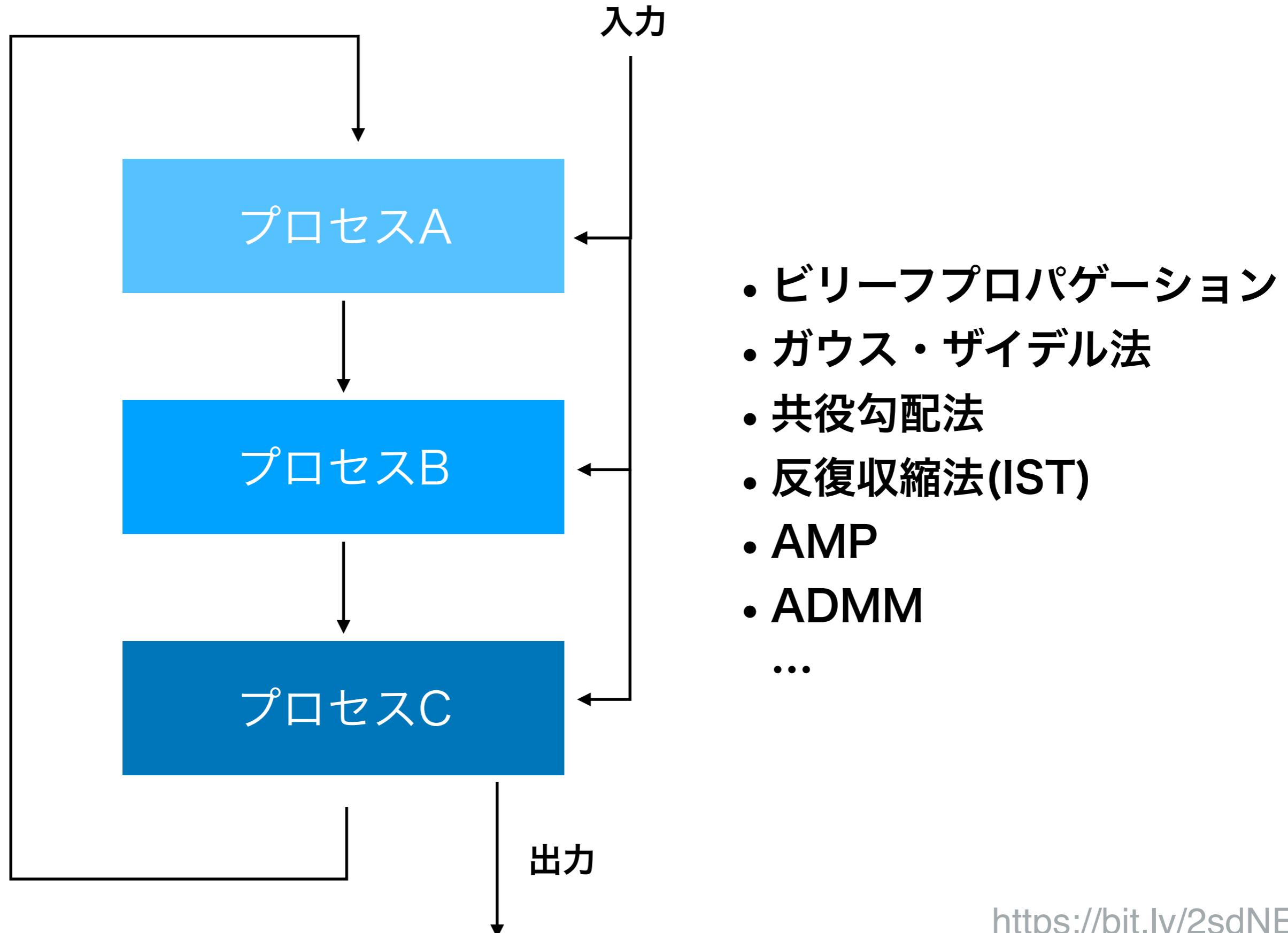


A. Karpathy

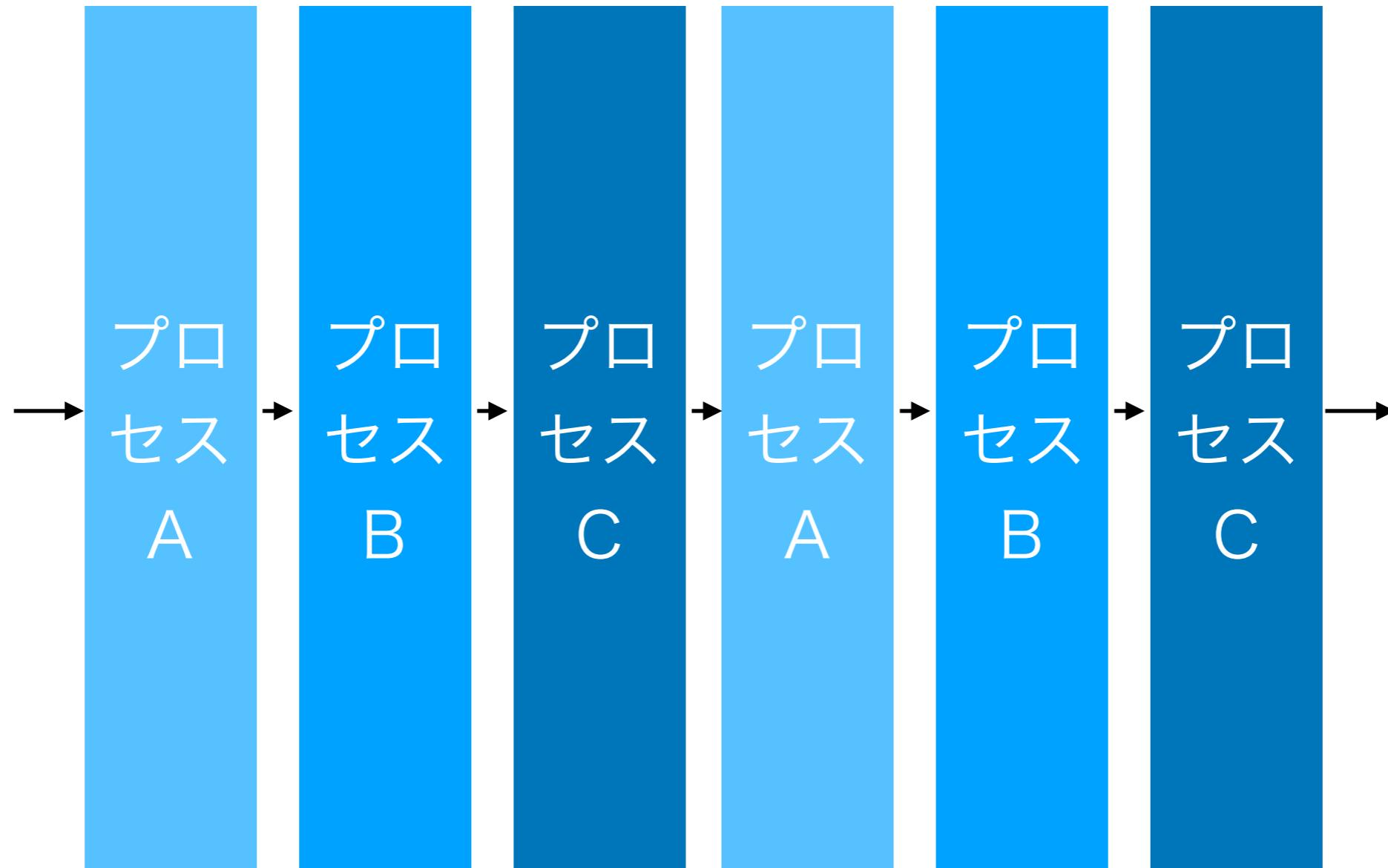
<https://medium.com/@karpathy/software-2-0-a64152b37c35>

<https://bit.ly/2sdNEJ4>

# データ駆動アプローチによる反復アルゴリズムの改善

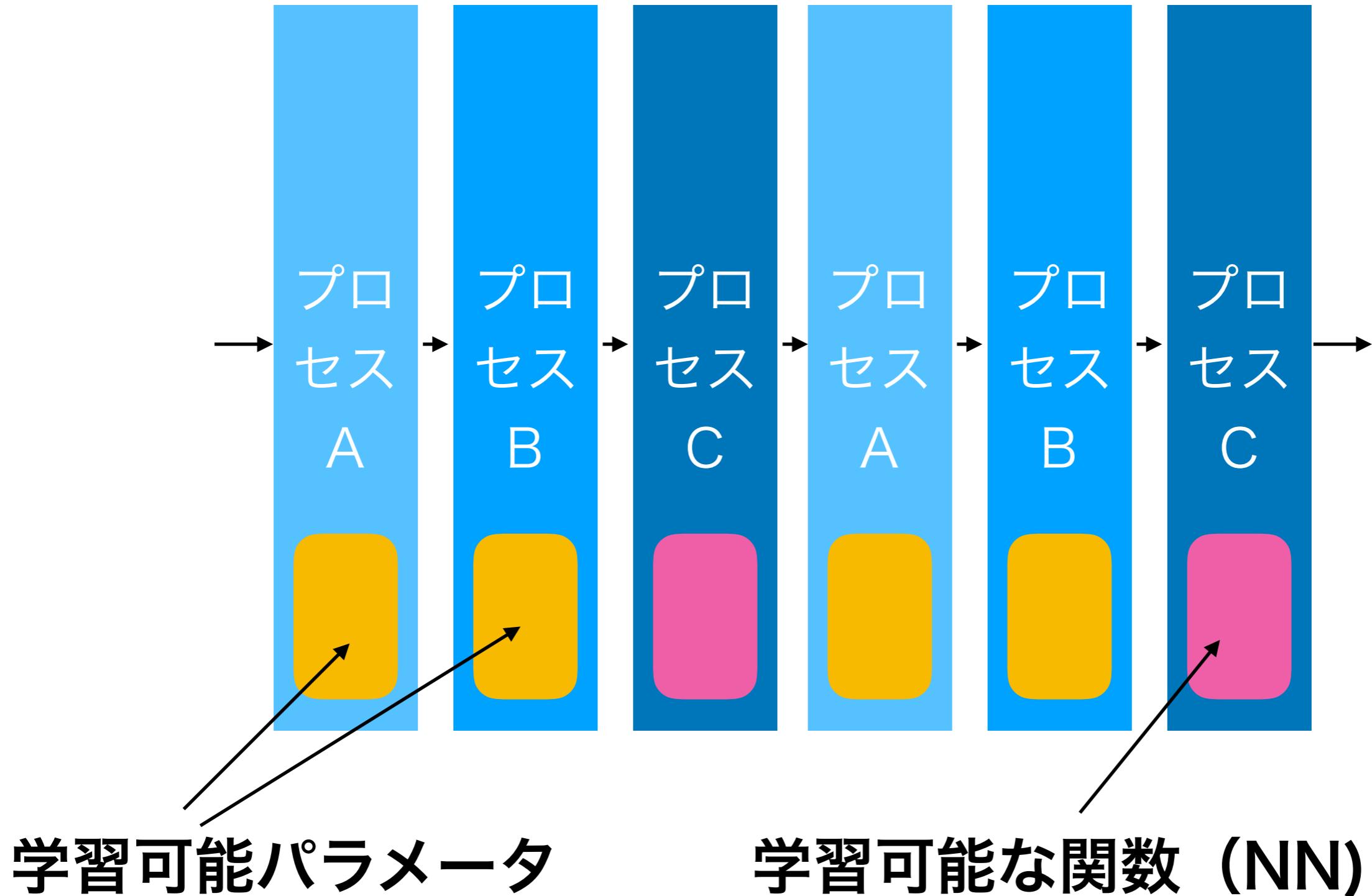


# 信号フローを時間方向に展開

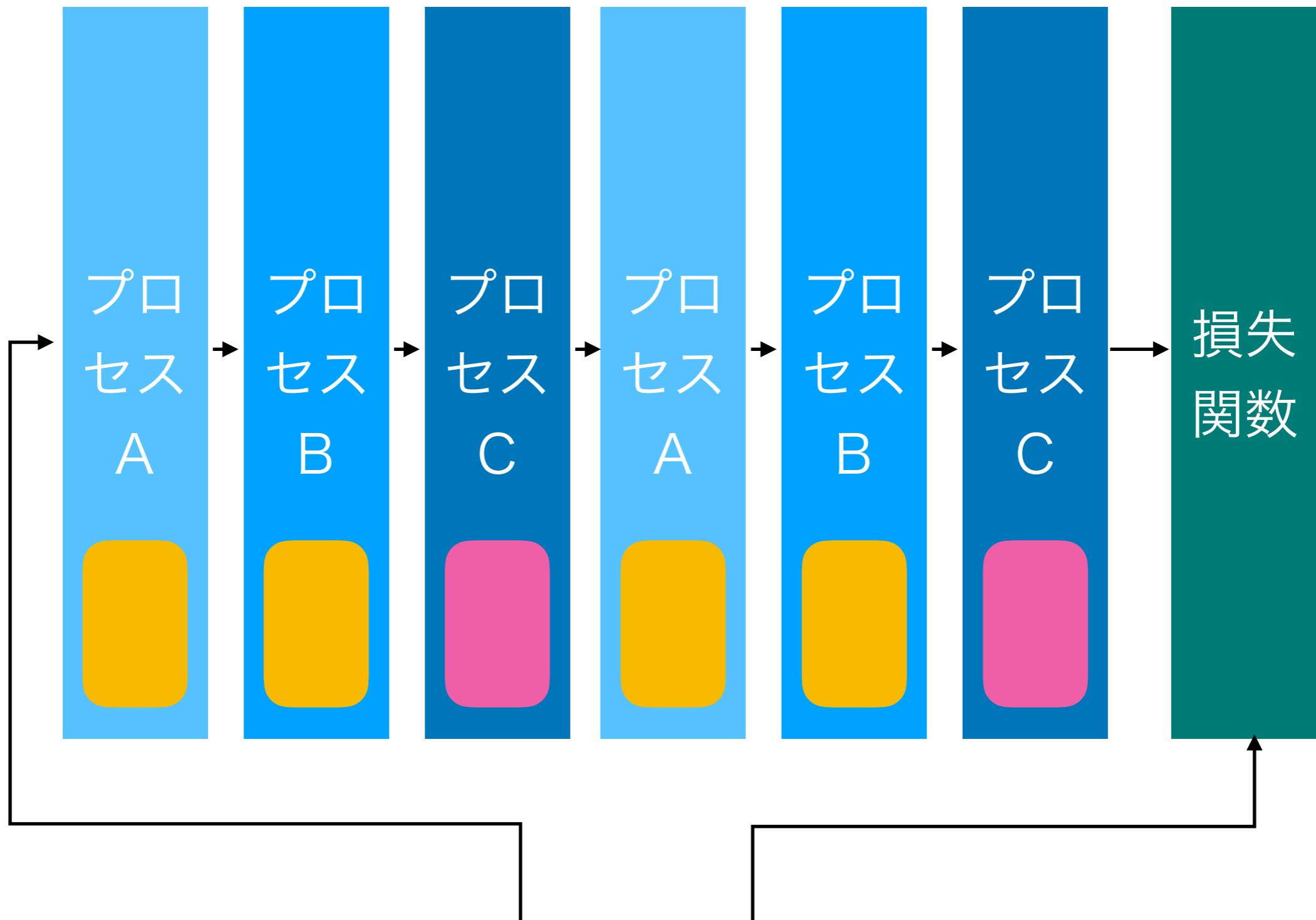


各プロセスが微分可能であり、かつ、その導関数が  
ほぼ至るところでゼロでなければ、**backprop可能**

# 学習可能パラメータを埋め込む

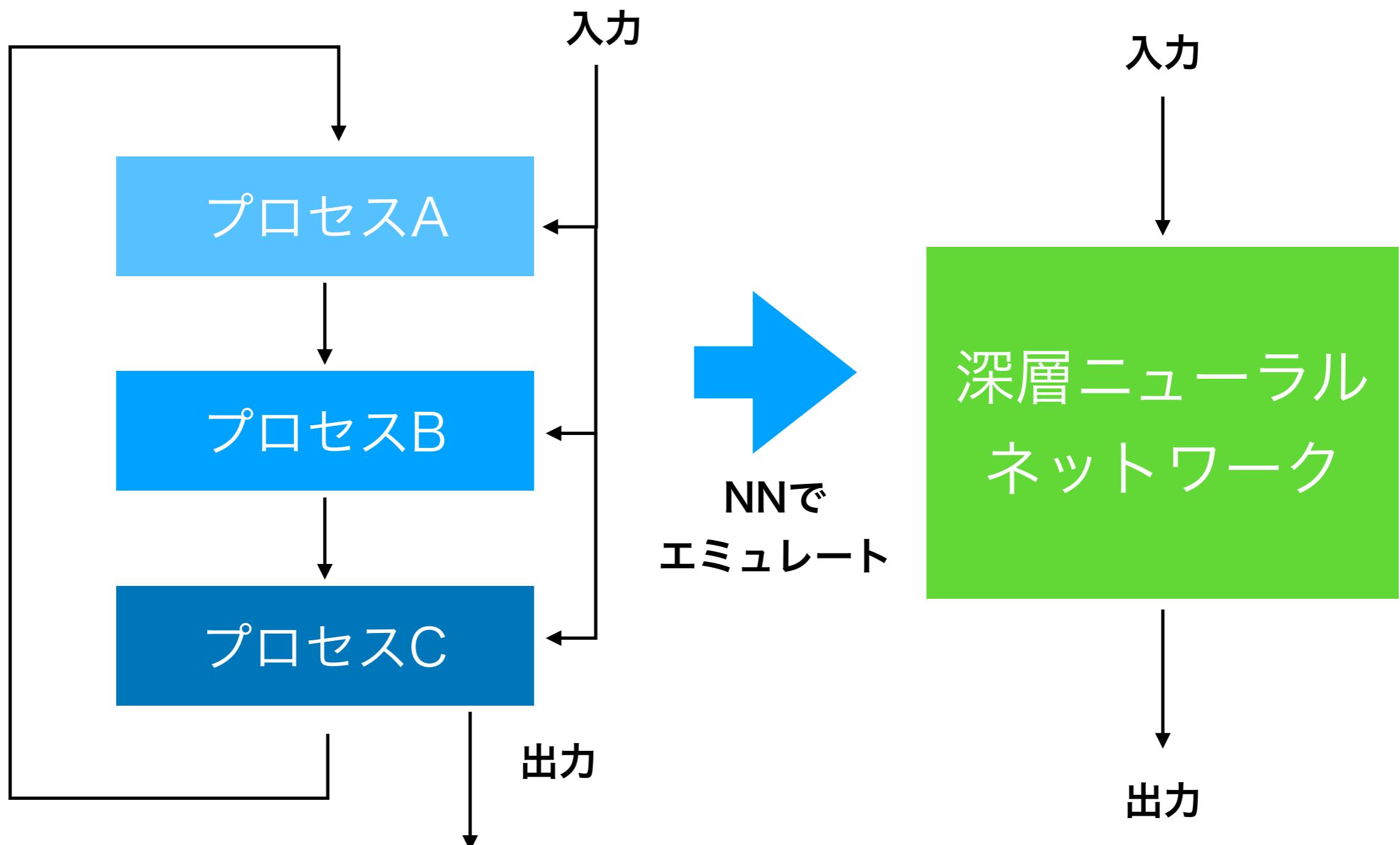


# 学習フェーズ

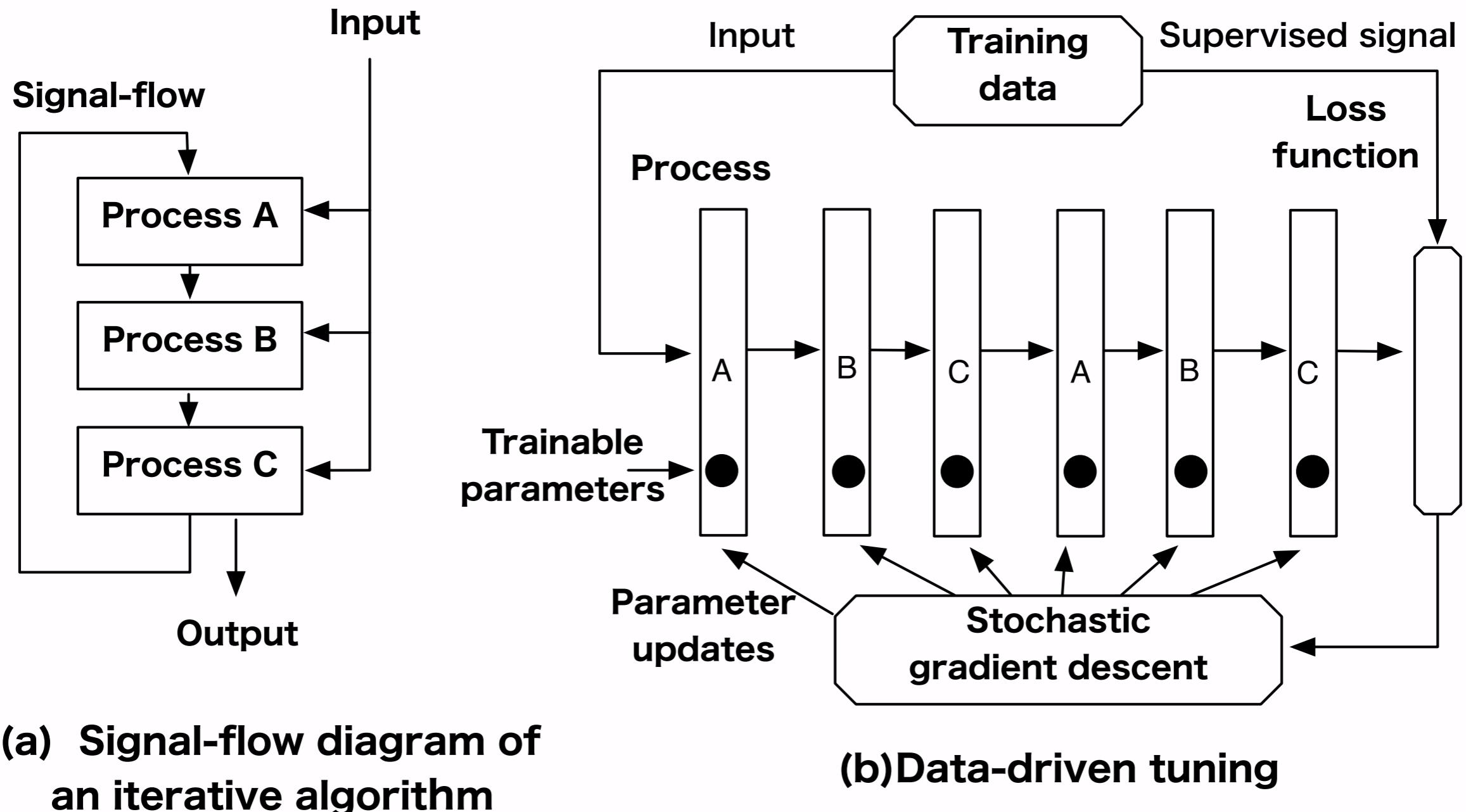


{観測信号, 元信号}

# End-to-End アプローチ



# データ駆動アプローチ



(a) Signal-flow diagram of an iterative algorithm

(b) Data-driven tuning

# データ駆動アプローチのメリット

- 多数の内部パラメータの値を合理的に設定できる  
(例えば反復ごとに独立パラメータを導入できる)
- 既知の優れたアルゴリズムをベースに選ぶなど、  
従来の知見を活かしたアルゴリズム設計が可能
- 対象とする通信系の特性に合わせたオンライン  
でのファインチューニングも可能
- パラメータ最適化の結果から知見が得られること  
もある

# データ駆動アプローチの例(1)

- 2次元最小化問題(凸関数)を例にあげる
- 勾配法の利用(初期値はランダムに選ぶ)

目的関数  $f(x_1, x_2) = x_1^2 + qx_2^2$

通常の勾配法(GD)の基本ステップ

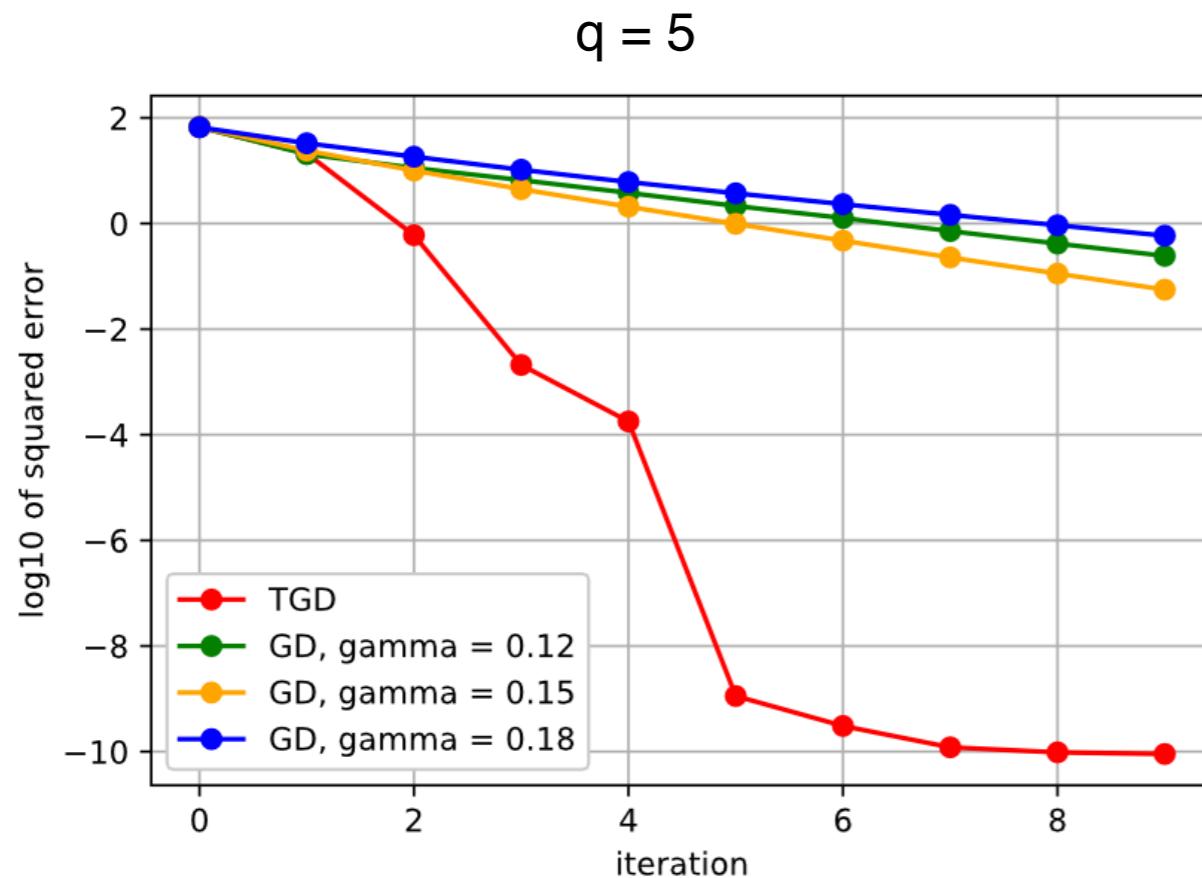
$$s_{t+1} = s_t - \gamma \nabla f(s_t)$$

学習可能パラメータを含む勾配法(TGD)の基本ステップ

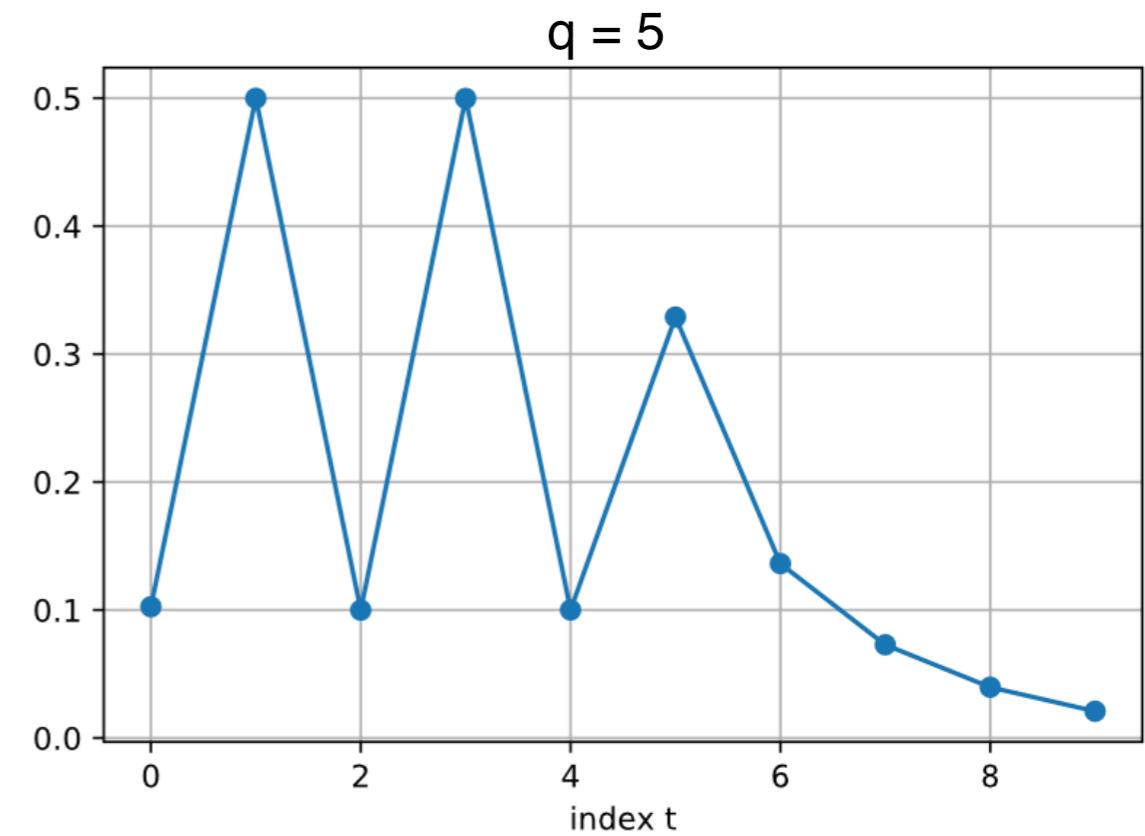
$$s_{t+1} = s_t - \underline{\gamma_t} \nabla f(s_t).$$

# 2次元最小化問題(凸関数)

真値からの誤差



学習可能パラメータの値

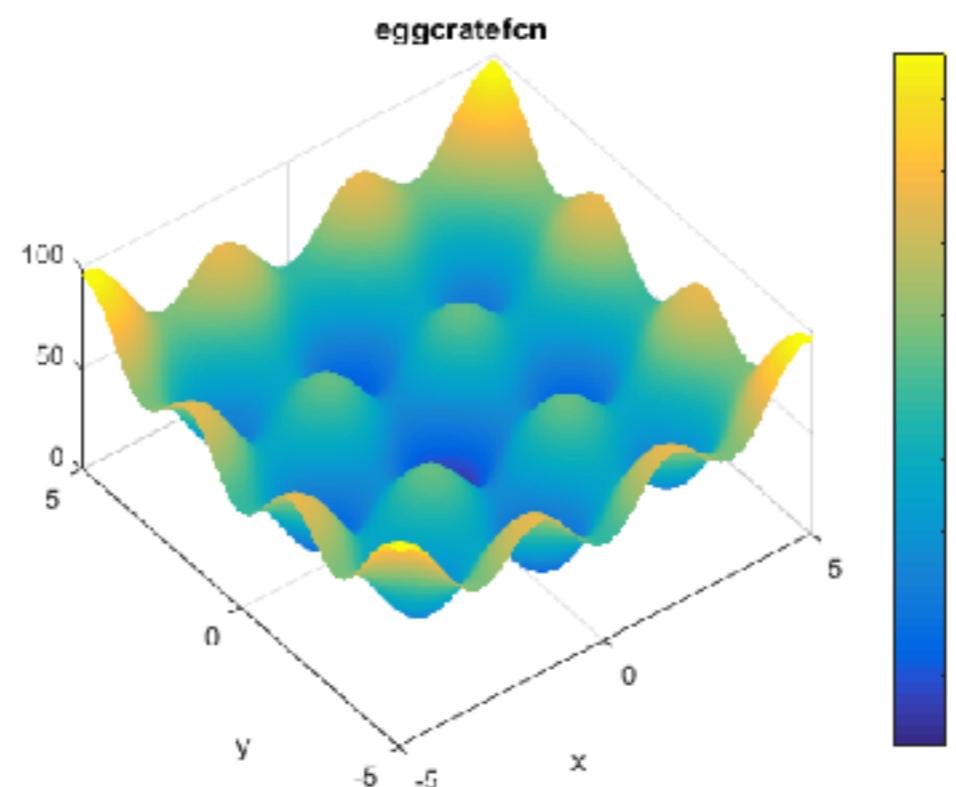


- ステップサイズ固定よりも速い収束
- ジグザグ型のパラメータ値

# 多峰性関数の場合

卵入れ関数 (egg crate function)

$$f(x_1, x_2) = x_1^2 + x_2^2 + 25(\sin^2(x_1) + \sin^2(x_2))$$

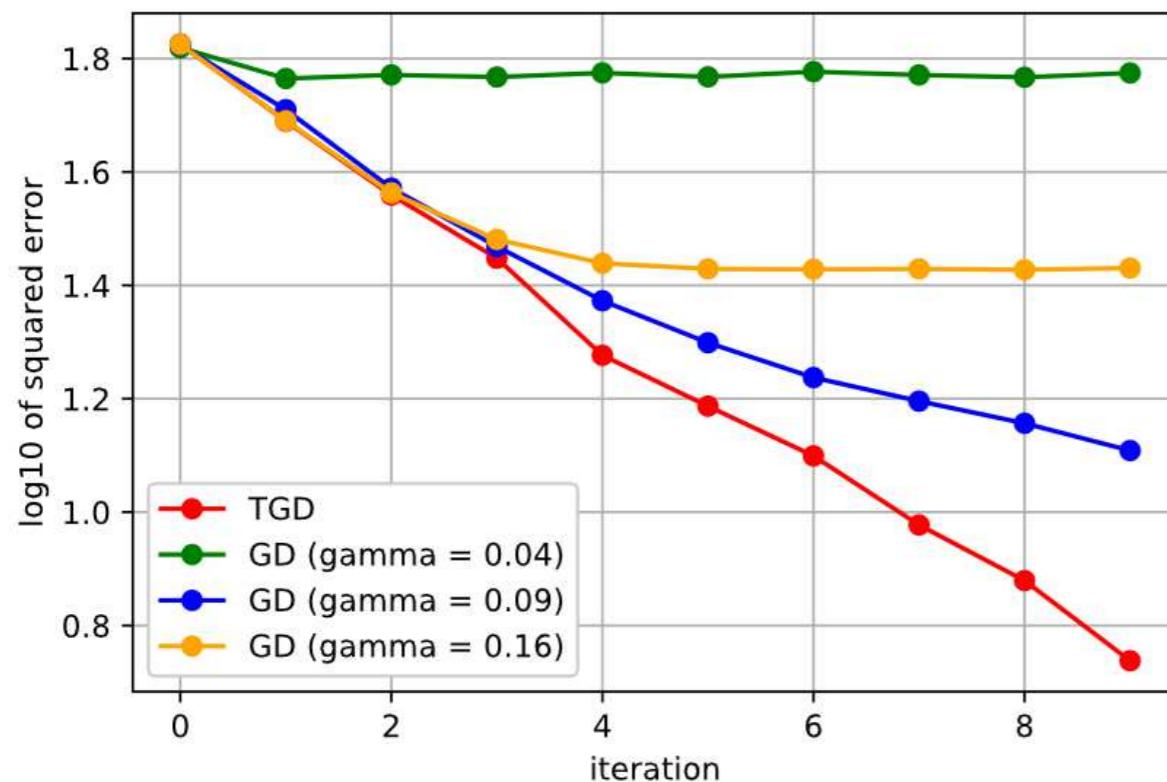


cited from: <http://benchmarkfcns.xyz/benchmarkfcns/eggcratefcn.html>

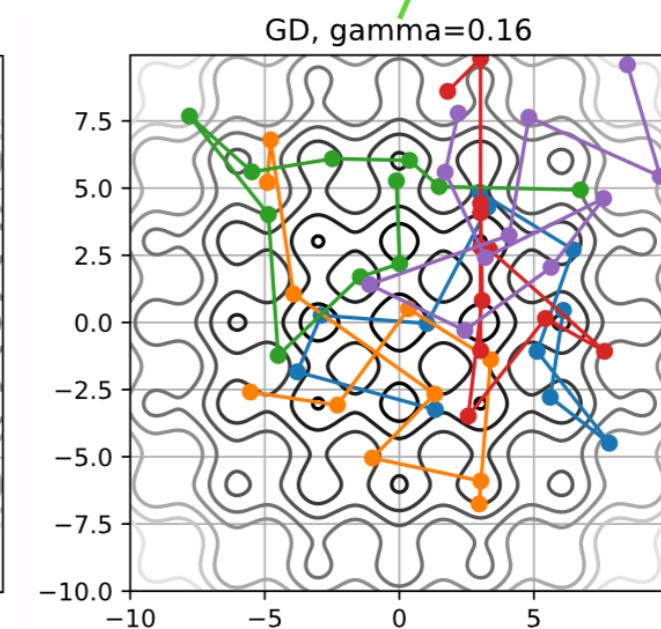
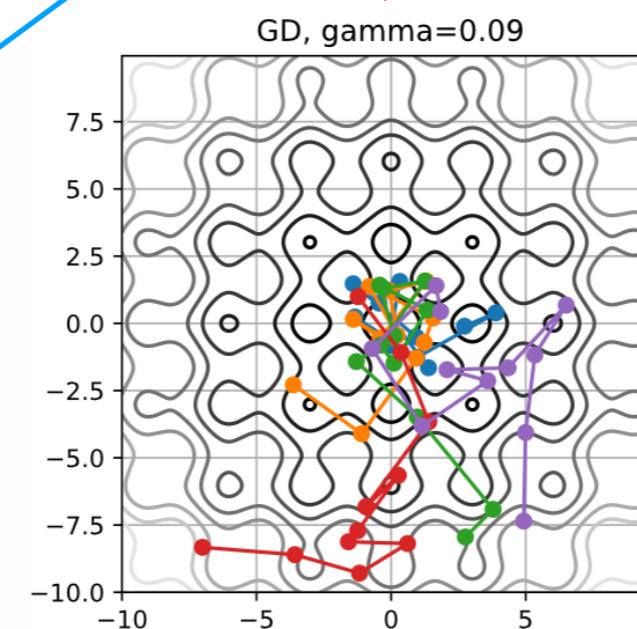
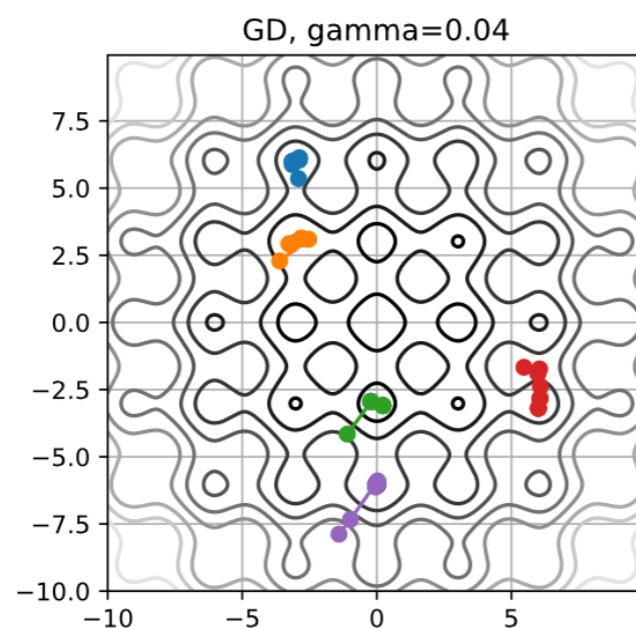
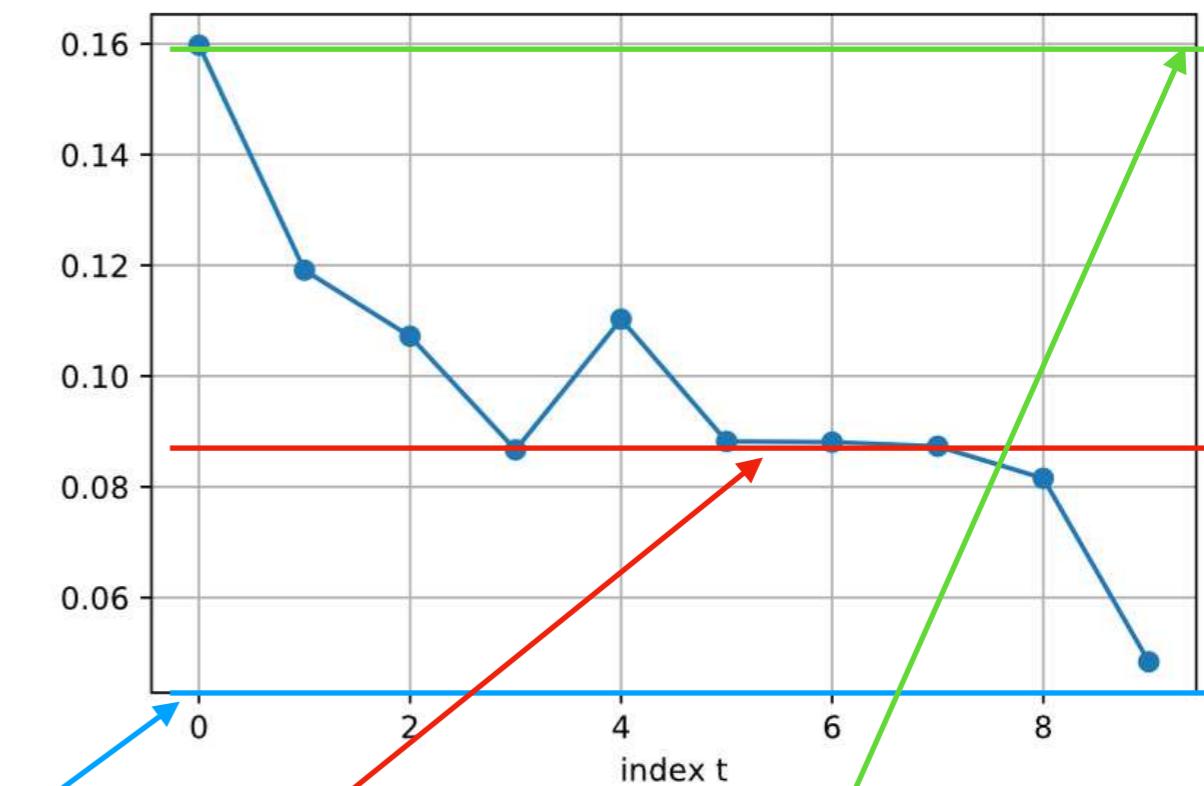
<https://bit.ly/2sdNEJ4>

# 多峰性関数の場合

真値からの誤差



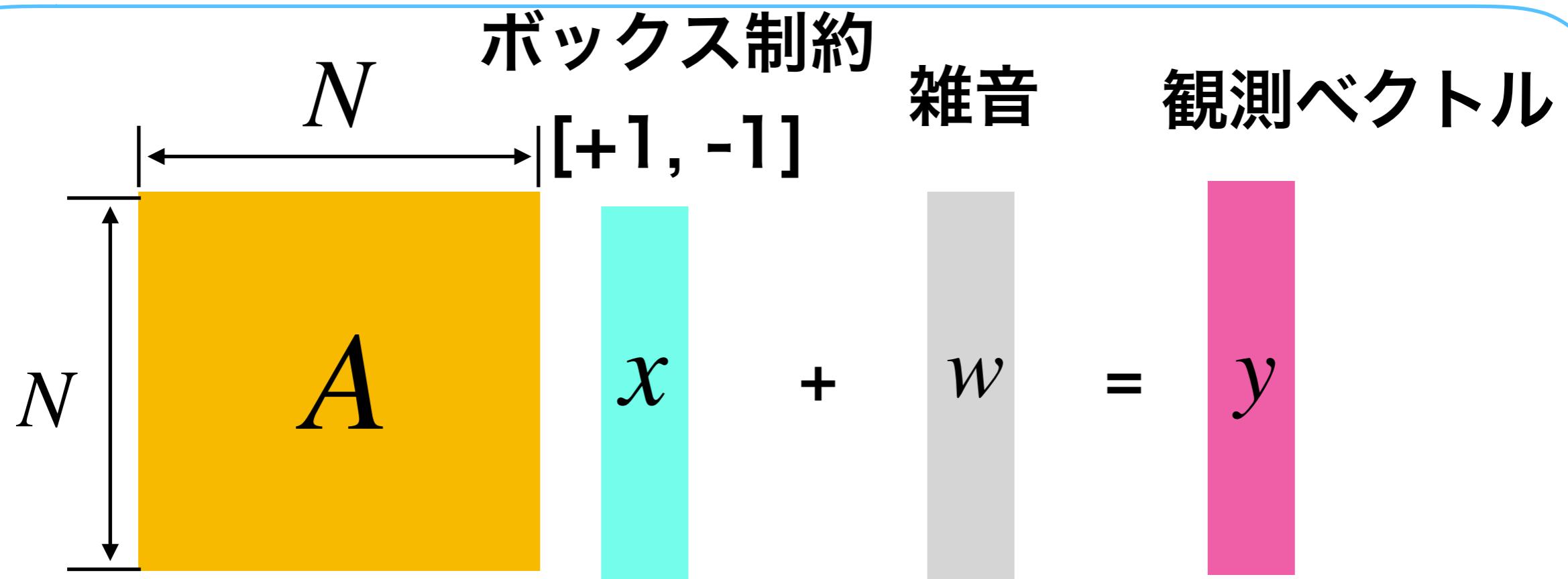
学習可能パラメータの値



# 観察

- 訓練プロセスにより適切なステップサイズパラメータが選択されている
- 反復ごとに独立したステップサイズパラメータが本質的に重要
- 訓練プロセスにより学習された結果は一種の「最小化のための戦略」と見ることができる
- 学習された戦略は必ずしも自明ではない

# データ駆動アプローチの例(2)



問題： $y$ から  $x$  を再現せよ。

$$\min_x \|y - Ax\|_2^2 \text{ subject to } x \in [-1, 1]^n$$

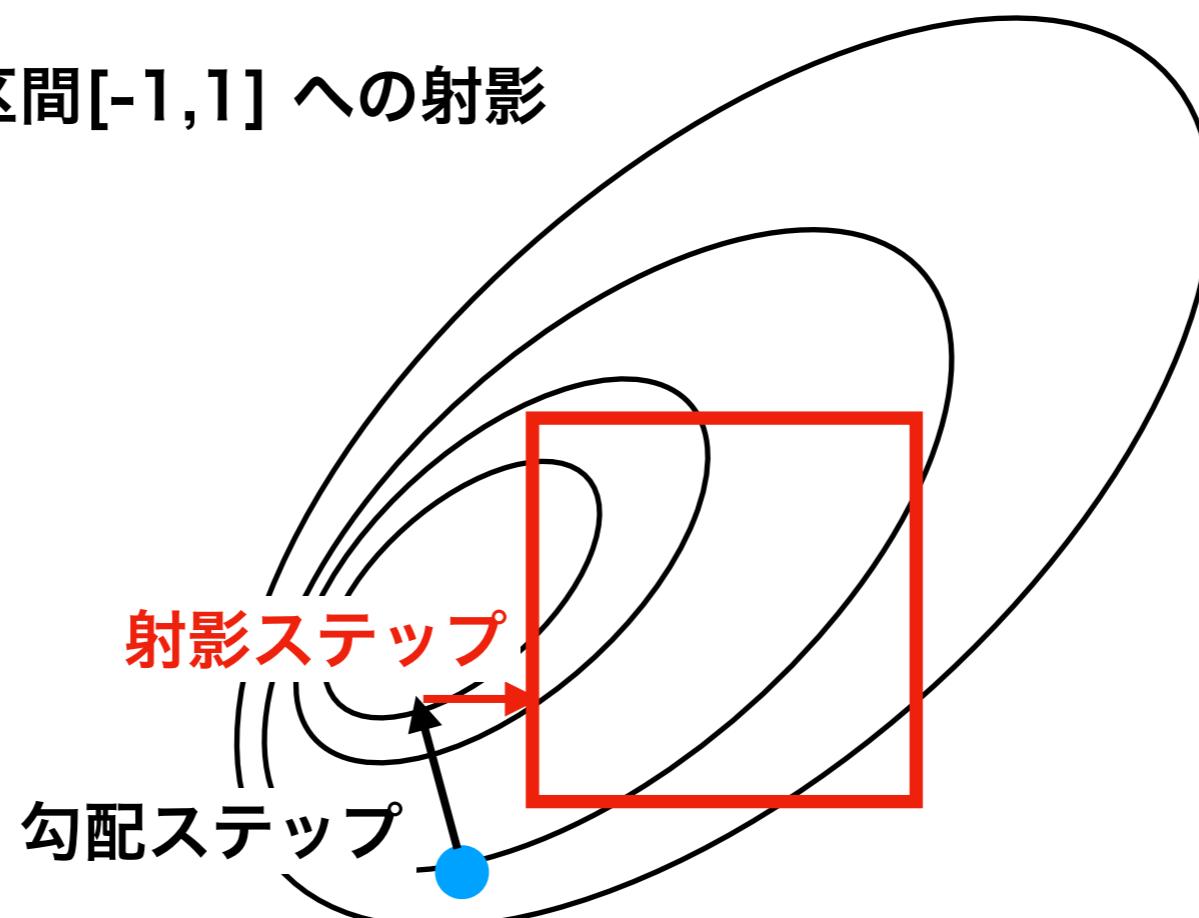
# 射影勾配法

制約付き凸最適化において、しばしば用いられる最小化技法

$$r_t = s_t + \gamma A^T(y - As_t), \quad (\text{勾配ステップ})$$

$$s_{t+1} = \varphi(\xi r_t) \quad (\text{射影ステップ})$$

$\varphi$ は閉区間[-1,1]への射影

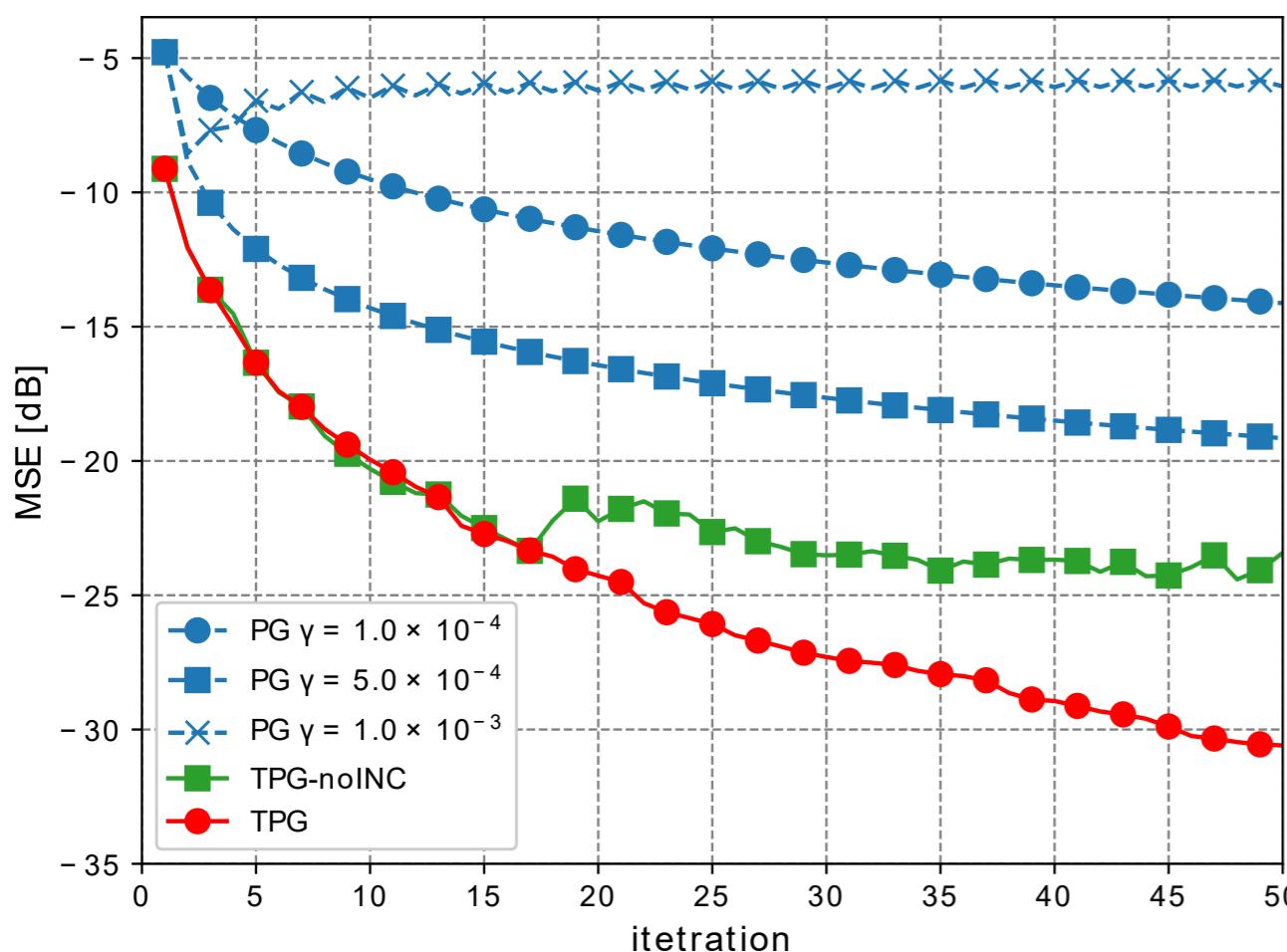


# 学習可能パラメータを含む射影勾配法

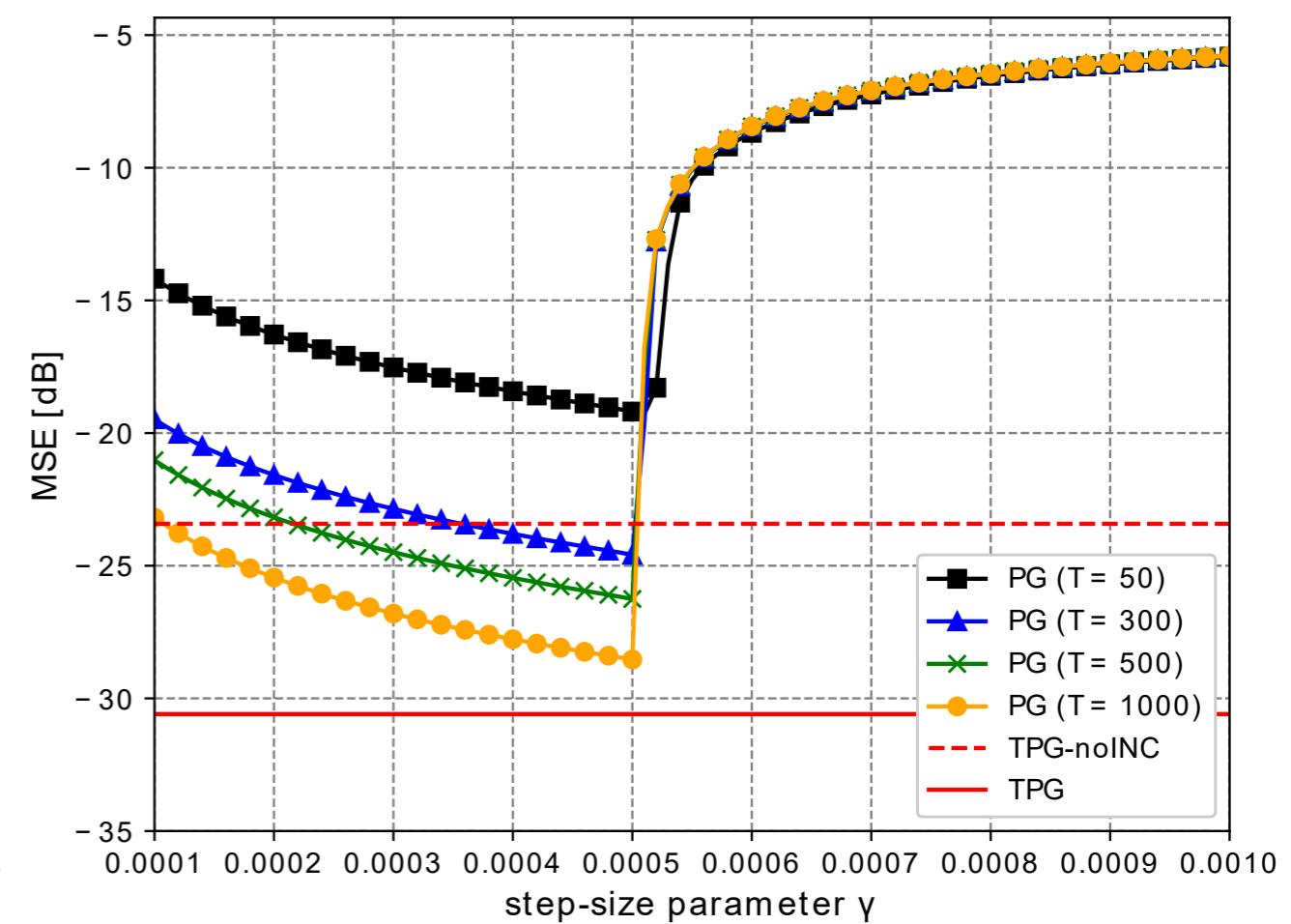
$$\mathbf{r}_t = \mathbf{s}_t + \underline{\gamma_t} \mathbf{A}^T (\mathbf{y} - \mathbf{A}\mathbf{s}_t), \quad (\text{勾配ステップ})$$

$$\mathbf{s}_{t+1} = \varphi(\xi \mathbf{r}_t) \quad (\text{射影ステップ})$$

真値からの誤差



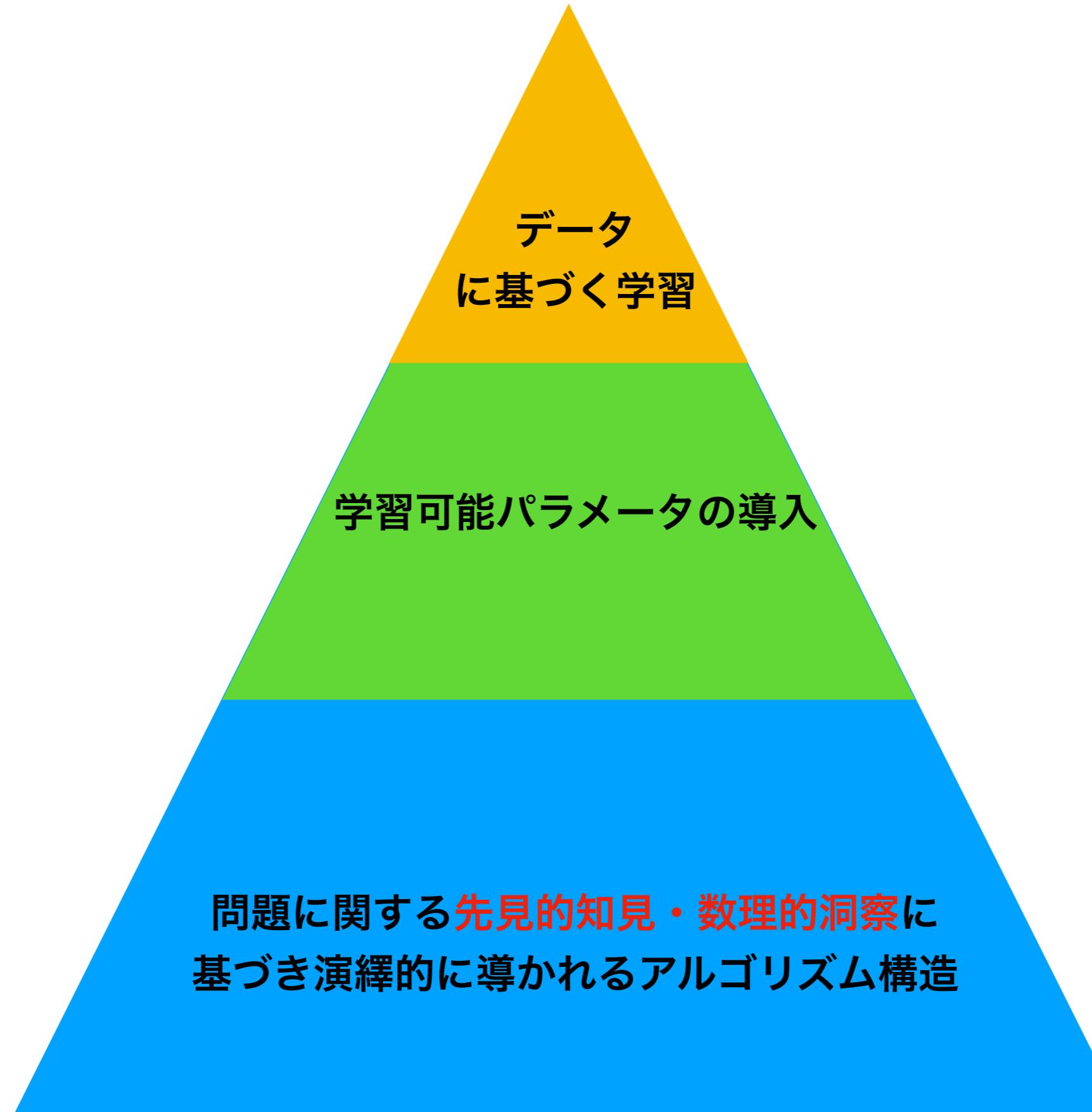
真値からの誤差



# 観察

- ステップサイズパラメータを訓練プロセスにより学習することで、収束のスピードの大幅な向上が認められる→「収束加速現象」
- 層を一層ごと追加していくインクリメンタル学習が効果的
- 固定ステップパラメータでは到達できない誤差水準を達成している

# データ駆動アプローチに基づく反復アルゴリズムの改善



# アルゴリズム設計フロー

- ・ ターゲット問題を凸・非凸最適化問題として定式化する
- ・ 推定・検出アルゴリズムを (1) 射影勾配法, (2) 近接射影勾配法, (3) ペナルティ関数法, (4) ADMMをベースとして構成する
- ・ 学習可能パラメータを導入する
- ・ ランダムに生成した訓練サンプルで学習(オフライン学習)  
または、実信号で学習(オンライン学習)

# 本研究室の関連研究を紹介します



## スパース信号再現アルゴリズム

Ito, Takabe, W, “Trainable ISTA for Sparse Signal Recovery”,  
IEEE ICC Workshop, May, 2018 (arXiv:1801.01978,  
<https://github.com/wadayama/TISTA>)



## 過負荷MIMO 検出

Takabe, Imanishi, W, Hayashi,  
“Trainable Projected Gradient Detector for Massive Overloaded MIMO  
Channels: Data-driven Tuning Approach”, arXiv:1812.10044

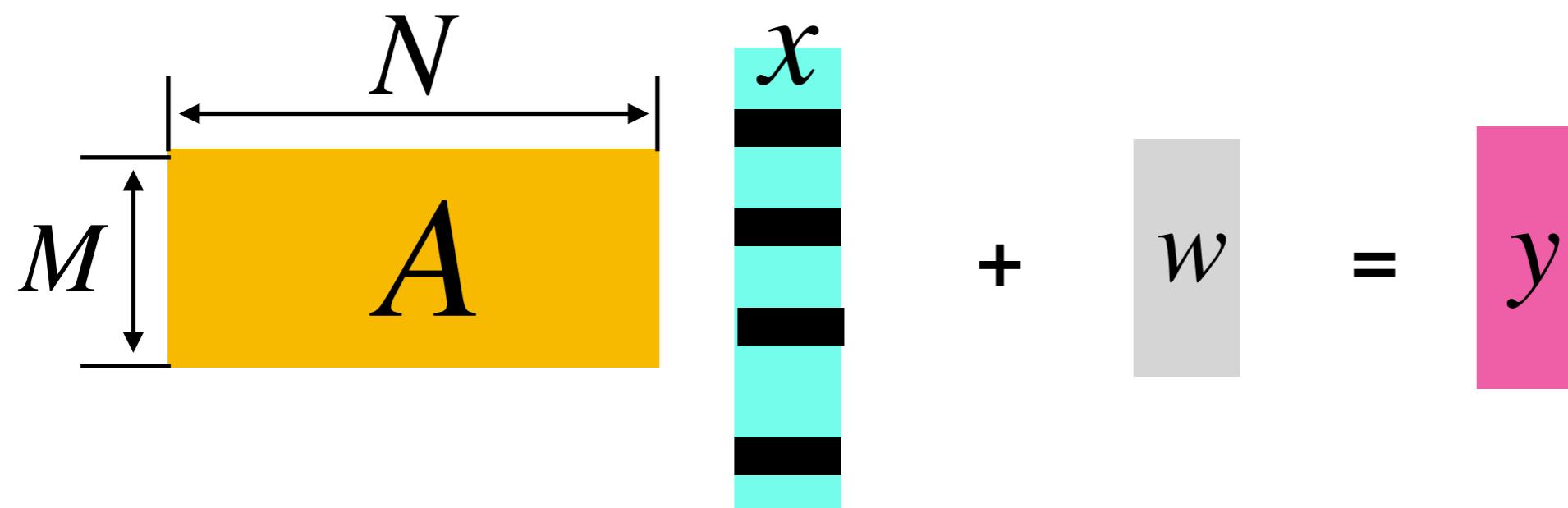


## LDPC符号に適した量子化器設計

W, Takabe, “Quantizer Optimization based on Neural Quantizer  
for Sum-Product Decoder ”, IEEE Globecom, 2018

# 圧縮センシングの問題設定

観測行列 疎ベクトル ノイズ 観測ベクトル



観測ベクトルを見て、疎ベクトルを可能な限り正確に  
推定したい



「未知変数の数 > 方程式の本数」となる**劣決定系問題**

# 無線通信における圧縮センシング

- 通信路推定
- 到来波方向推定
- スペクトルセンシング
- NOMA

A User's Guide to Compressed Sensing for Communications Systems

K.HAYASHI, M. NAGAHARA and T. TANAKA

IEICE TRANS. COMMUN., VOL.E96-B, NO.3 MARCH 2013

Recommended !

# 無線通ニューラルネットに基づくスパース信号再現信における圧縮センシング

Ito, W, SITA2016

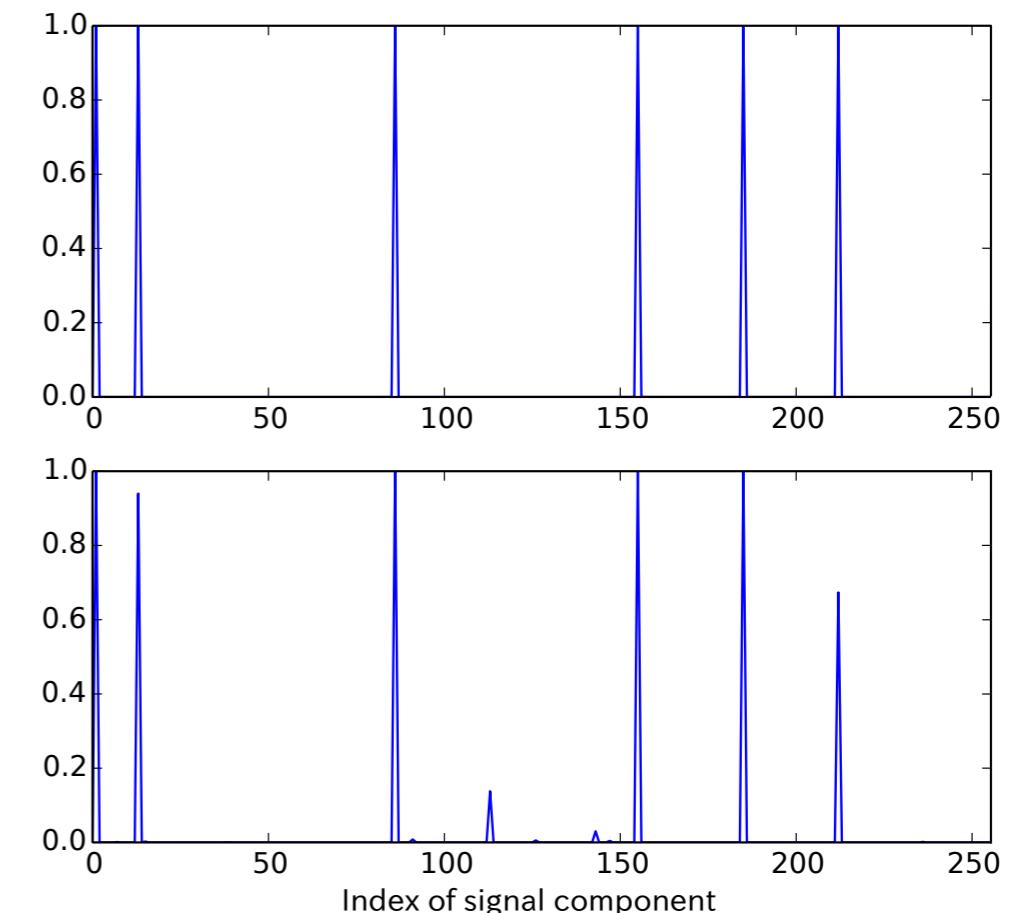
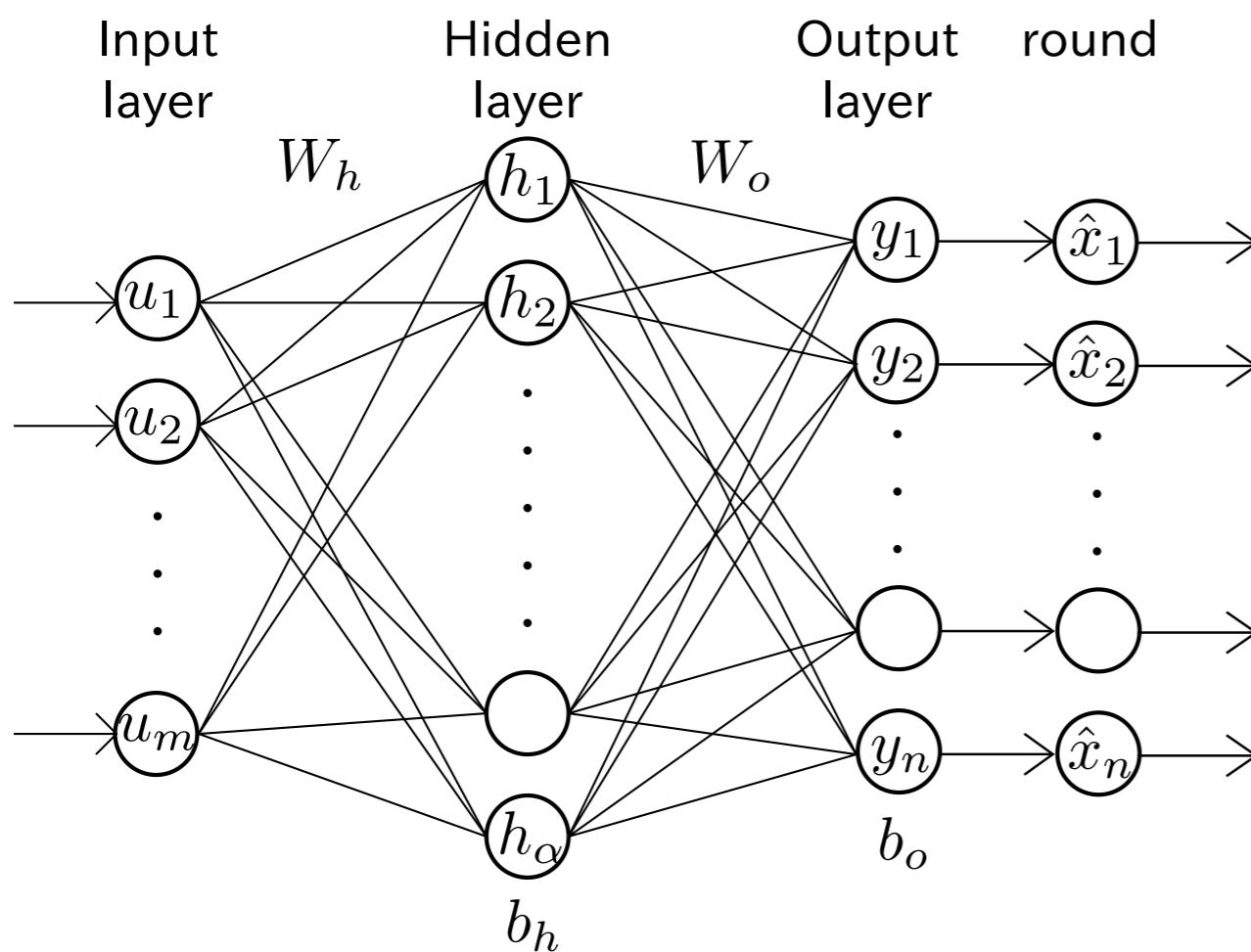


Fig. 2. Sparse signal recovery for a 6-sparse vector. (top: the original sparse signal  $x$ , bottom: the output  $y = \Phi_{\theta^*}(x)$  from the trained neural network  $n = 256, m = 120$ )

End-to-Endアプローチ

どうも大きなシステムにうまくスケールしない→方向転換

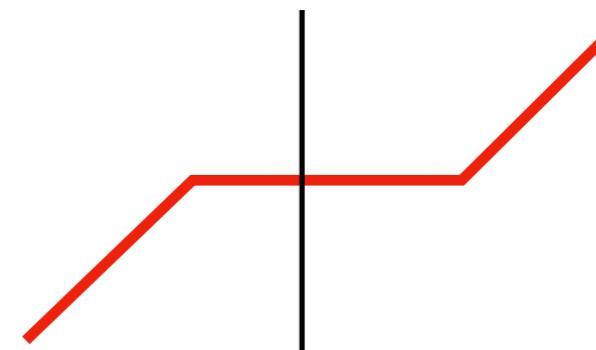
# 既存手法: ISTA



ISTAはよく知られている反復型信号再現アルゴリズム

$$\begin{aligned} r_t &= s_t + \beta A^T (y - As_t) \\ s_{t+1} &= \eta(r_t; \tau), \end{aligned}$$

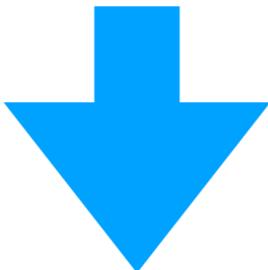
$\eta$ はソフトしきい値関数



# ISTAの導出

LASSO

$$\hat{x} = \left\| y - Ax \right\|_2^2 + \lambda \left\| x \right\|_1$$



近接勾配法

(proximal gradient descent)

$$r_t = s_t + \beta A^T (y - As_t) \quad \text{勾配法ステップ}$$

$$s_{t+1} = \eta(r_t; \tau), \quad \text{近接射影ステップ}$$

(注) L1正則化項に対応する近接射影写像がソフトしきい値関数となる

<https://bit.ly/2sdNEJ4>

# LISTA

## 可微分アプローチの先駆け

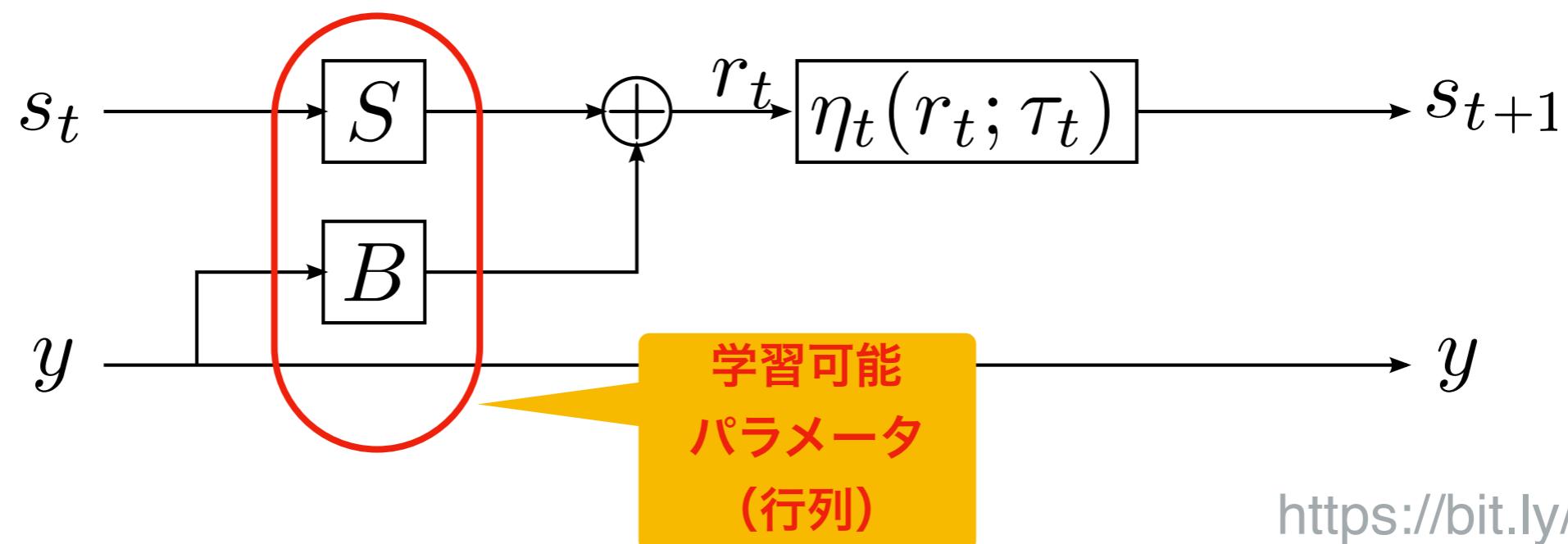
K. Gregor, and Y. LeCun,

``Learning fast approximations of sparse coding,"

Proc. 27th Int. Conf. Machine Learning}, pp. 399--406, 2010.

$$r_t = \underbrace{B s_t}_\text{S} + \underbrace{S y}_\text{y}$$

$$s_{t+1} = \eta(r_t; \tau_t)$$



# 提案手法: TISTA

$$\begin{aligned} r_t &= s_t + \gamma_t W(y - As_t), \\ s_{t+1} &= \eta_{MMSE}(r_t; \tau_t^2), \\ \nu_t^2 &= \max \left\{ \frac{\|y - As_t\|_2^2 - M\sigma^2}{\text{trace}(A^T A)}, \epsilon \right\}, \\ \tau_t^2 &= \frac{\nu_t^2}{N} (N + (\gamma_t^2 - 2\gamma_t)M) + \frac{\gamma_t^2 \sigma^2}{N} \text{trace}(WW^T) \end{aligned}$$

$W$ は $A$ の疑似逆行列

# 提案手法: TISTA

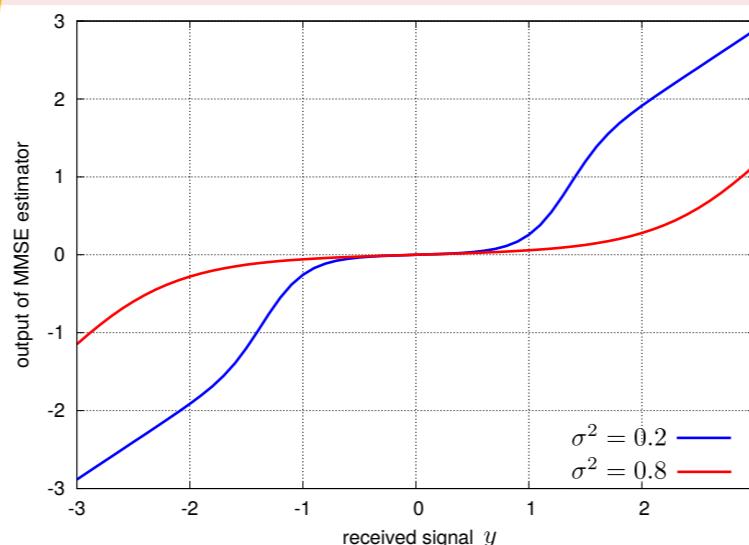
学習可能  
パラメータ  
(スカラー)

近接勾配  
ライクな処理

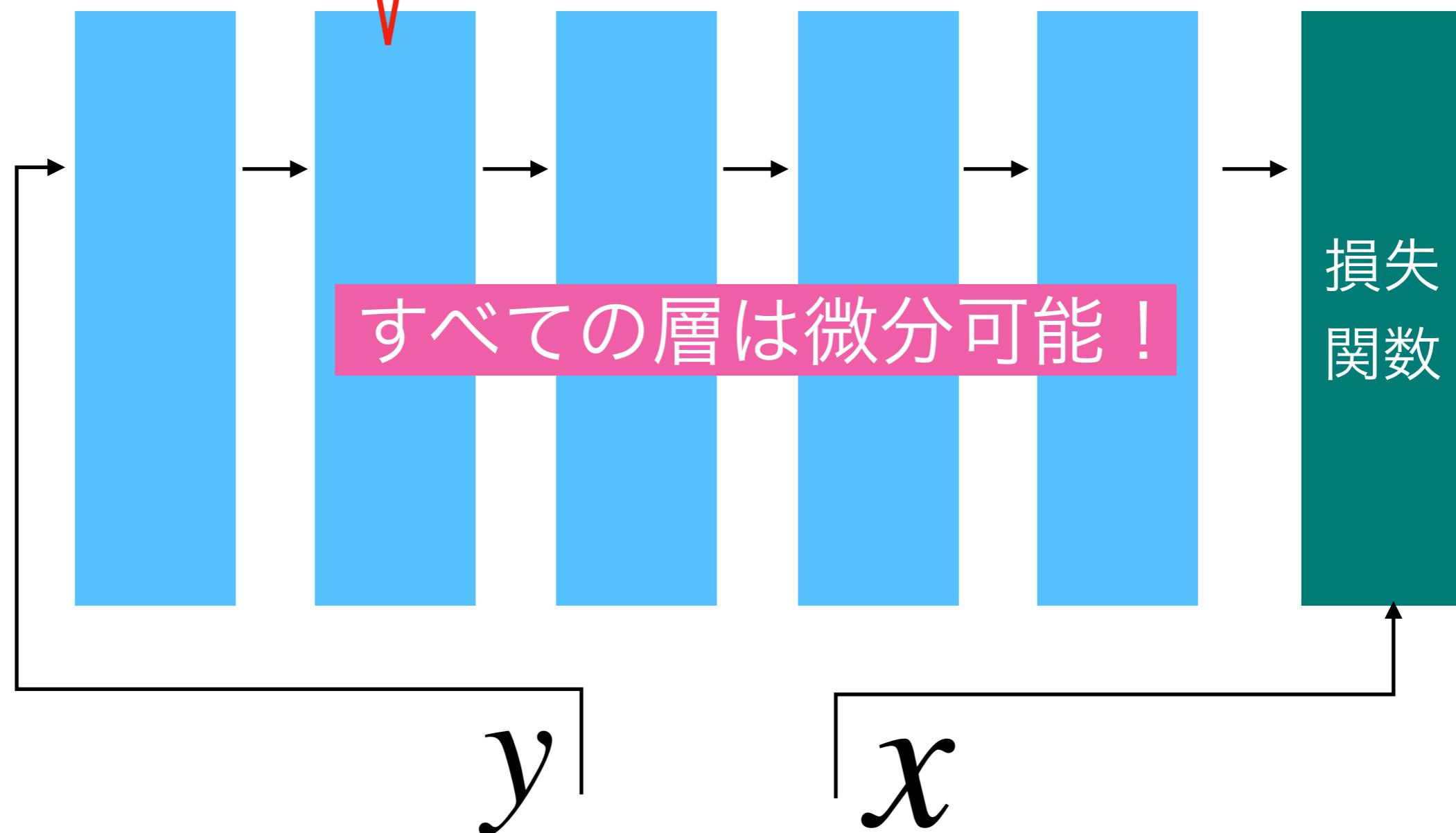
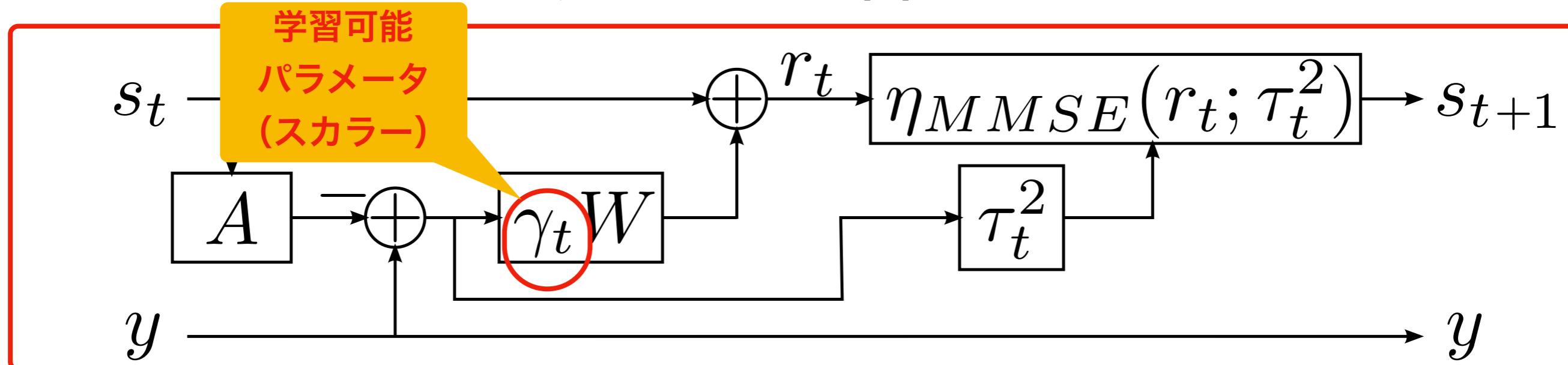
誤差分散の  
推定式

$$\begin{aligned} r_t &= s_t + \gamma_t W(y - As_t), \\ s_{t+1} &= \eta_{MMSE}(r_t; \tau_t^2), \\ v_t^2 &= \max \left\{ \frac{\|y - As_t\|_2^2 - M\sigma^2}{\text{trace}(A^T A)}, \epsilon \right\}, \\ \tau_t^2 &= \frac{v_t^2}{N} (N + (\gamma_t^2 - 2\gamma_t)M) + \frac{\gamma_t^2 \sigma^2}{N} \text{trace}(WW^T) \end{aligned}$$

こんな感じの  
関数→



# TISTAの1ラウンド分のブロック図



# 学習パラメータ数の比較

	TISTA	LISTA	LAMP
# of params	$T$	$T(N^2 + MN + 1)$	$T(NM + 2)$

- [2] M. Borgerding and P. Schniter, “Onsager-corrected deep learning for sparse linear inverse problems,” *2016 IEEE Global Conf. Signal and Inf. Proc. (GlobalSIP)*, Washington, DC, Dec. 2016, pp. 227-231.

# TISTAの復元性能 (10%が非ゼロ元となる状況)

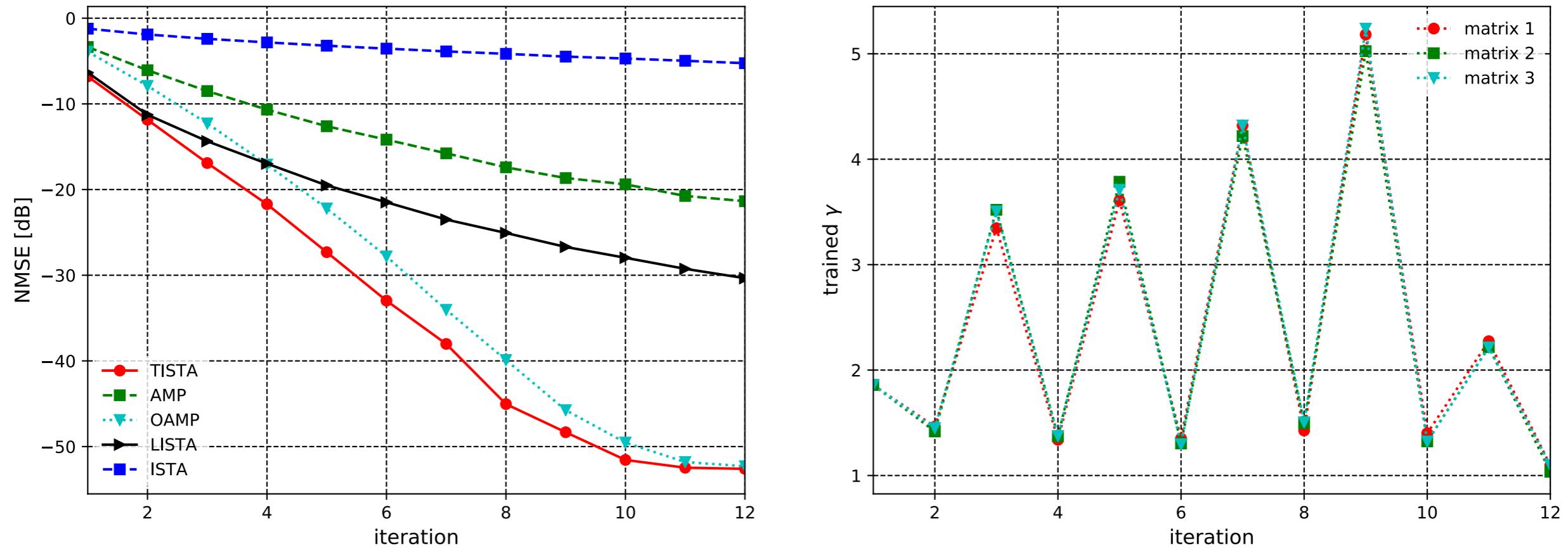


Figure 7: Sparse signal recovery for  $(n, m) = (300, 150)$ ,  $p = 0.1$  and SNR= 50 dB. (Left) MSE performances as a function of iteration steps. (Right) Trained values of  $\gamma_t$  under three different measurement matrices. The initial value is set to 1.0.

# TISTA vs OAMP (1)

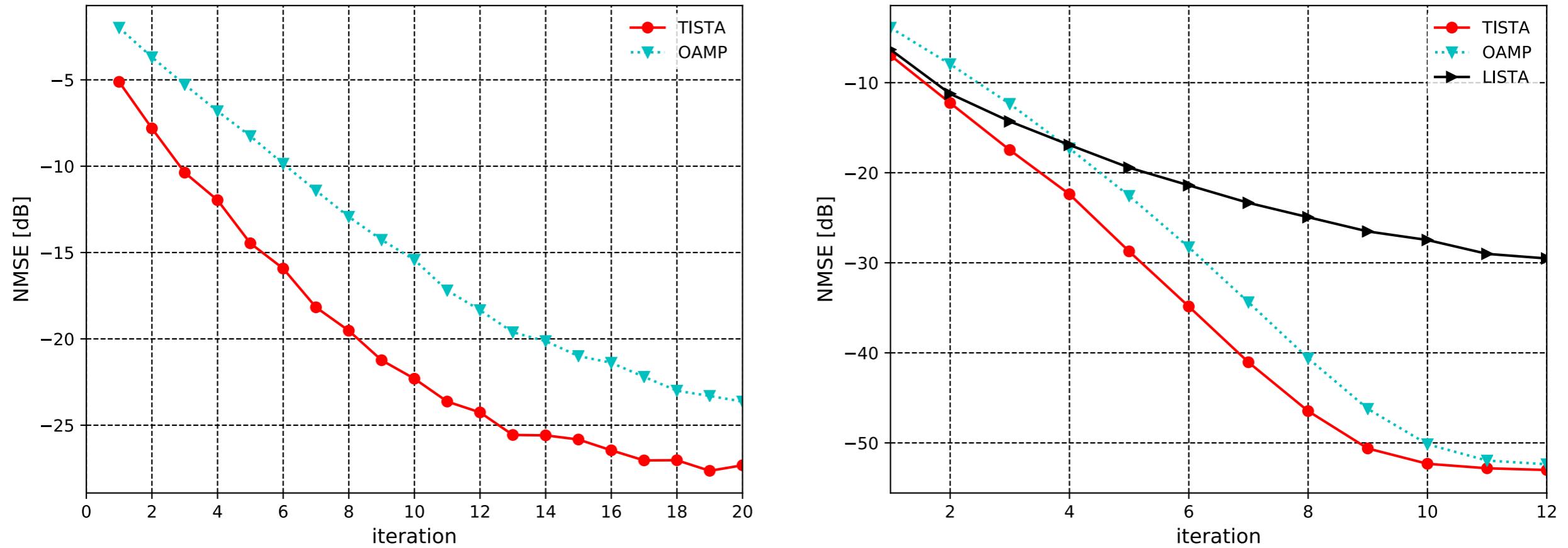


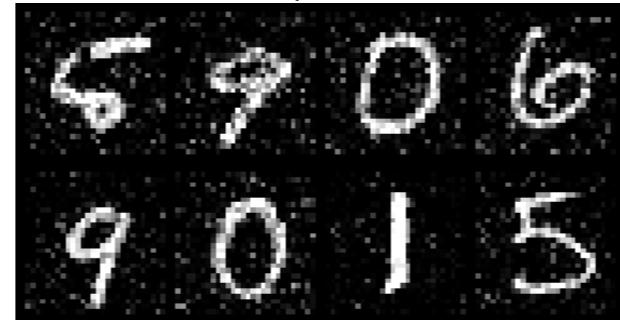
Figure 8: (Left) MSE performance of OAMP and TISTA for  $(n, m) = (300, 150)$ ,  $p = 0.18$  and SNR= 30 dB. (Right) MSE performances for  $(n, m) = (300, 150)$ ,  $p = 0.1$  and SNR= 30 dB with binary sensing matrix  $\mathbf{A} \in \{1, -1\}^{m \times n}$  as a function of iteration steps.

# TISTA vs OAMP (2)

MNISTデータ  
セットに対する  
スパース信号再現

人工データだけではなく  
実データでも良好な特性

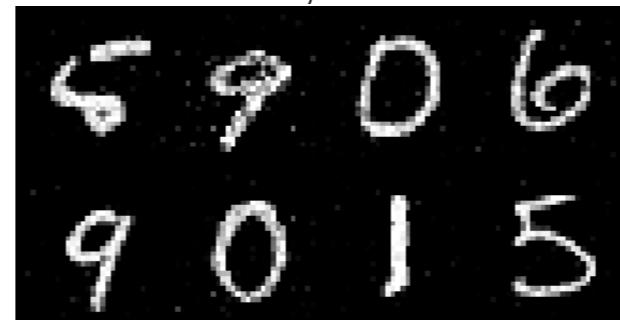
TISTA t = 5, MSE=0.0258



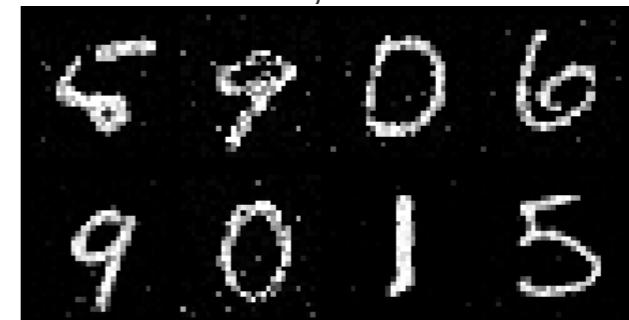
OAMP t = 5, MSE=0.0335



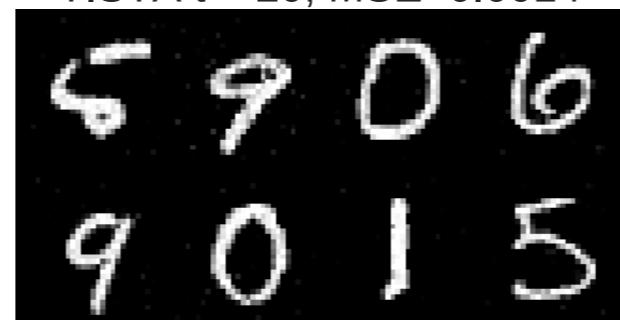
TISTA t = 10, MSE=0.0065



OAMP t = 10, MSE=0.0165



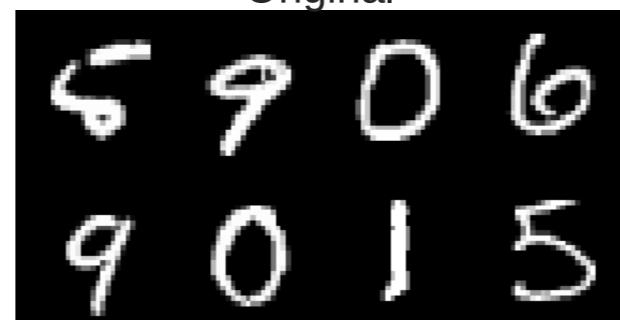
TISTA t = 20, MSE=0.0014



OAMP t = 20, MSE=0.0089



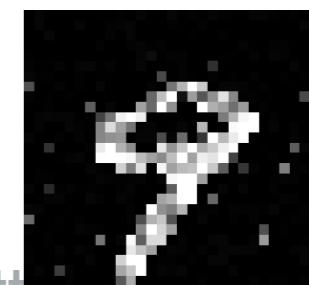
Original



t=20

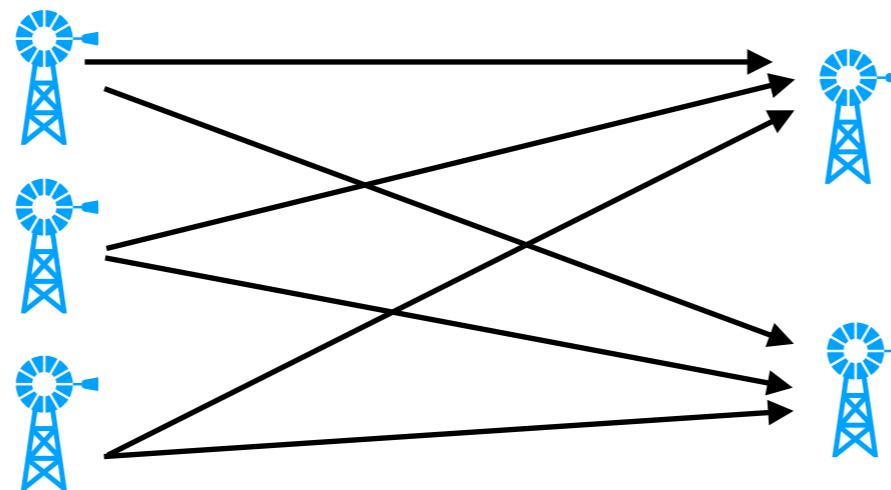
TISTA

OAMP



# 過負荷MIMO検出問題

送信アンテナ数 > 受信アンテナ数



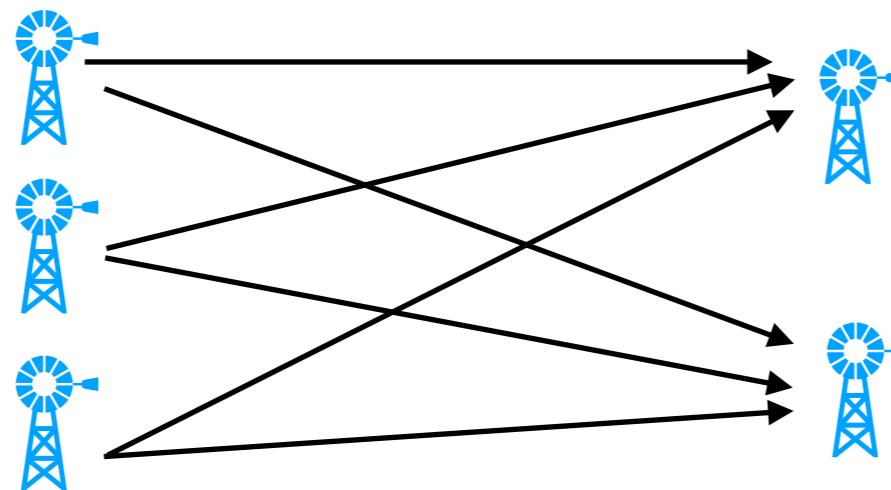
基地局側は多くのアンテナを持つ余裕があるが、  
端末側はコスト・消費電力などの点で、そんなに多くの  
アンテナを持ってない



mmWave MIMOなどでは、アンテナの小型化・  
高集積化が進む

# 過負荷MIMO検出問題

送信アンテナ数 > 受信アンテナ数



観測行列 2値ベクトル ノイズ 観測ベクトル

$$\begin{matrix} & \xleftarrow{N} \\ \xleftarrow{M} & H & \end{matrix} \quad x + w = y$$

劣決定系問題

# MIMO通信路モデル

複素モデル

$$\tilde{y} = \tilde{H}\tilde{x} + \tilde{w},$$

実数モデル

$$y = Hx + w,$$

$$y \triangleq \begin{bmatrix} \operatorname{Re}(\tilde{y}) \\ \operatorname{Im}(\tilde{y}) \end{bmatrix} \in \mathbb{R}^M, \quad H \triangleq \begin{bmatrix} \operatorname{Re}(\tilde{H}) & -\operatorname{Im}(\tilde{H}) \\ \operatorname{Im}(\tilde{H}) & \operatorname{Re}(\tilde{H}) \end{bmatrix},$$

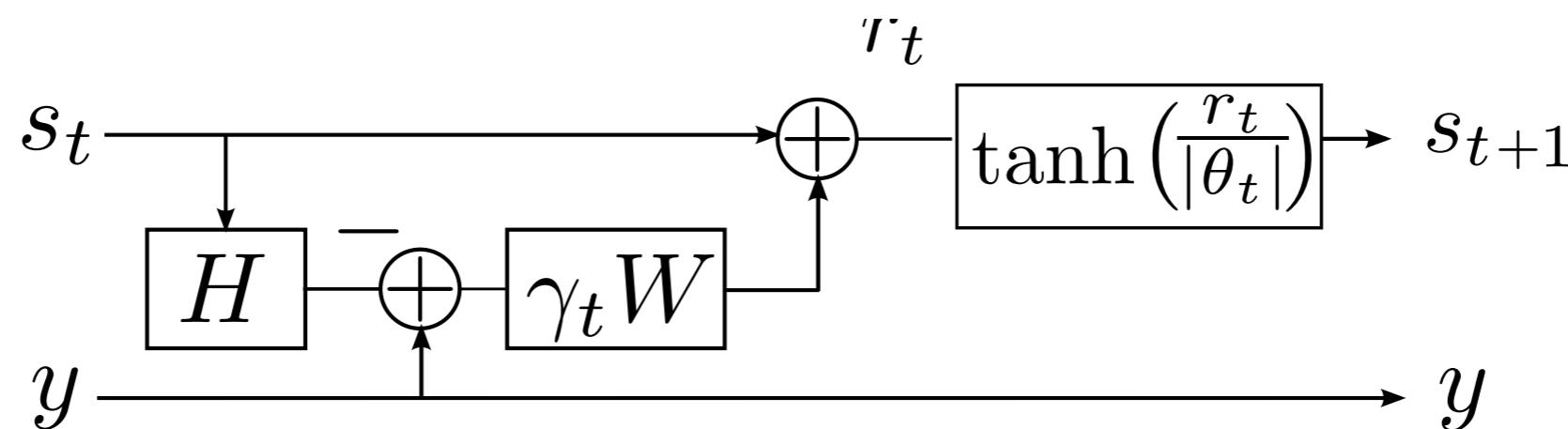
$$x \triangleq \begin{bmatrix} \operatorname{Re}(\tilde{x}) \\ \operatorname{Im}(\tilde{x}) \end{bmatrix} \in \mathbb{S}^N, \quad w \triangleq \begin{bmatrix} \operatorname{Re}(\tilde{w}) \\ \operatorname{Im}(\tilde{w}) \end{bmatrix} \in \mathbb{R}^M,$$

# 提案手法：TPG-Detector

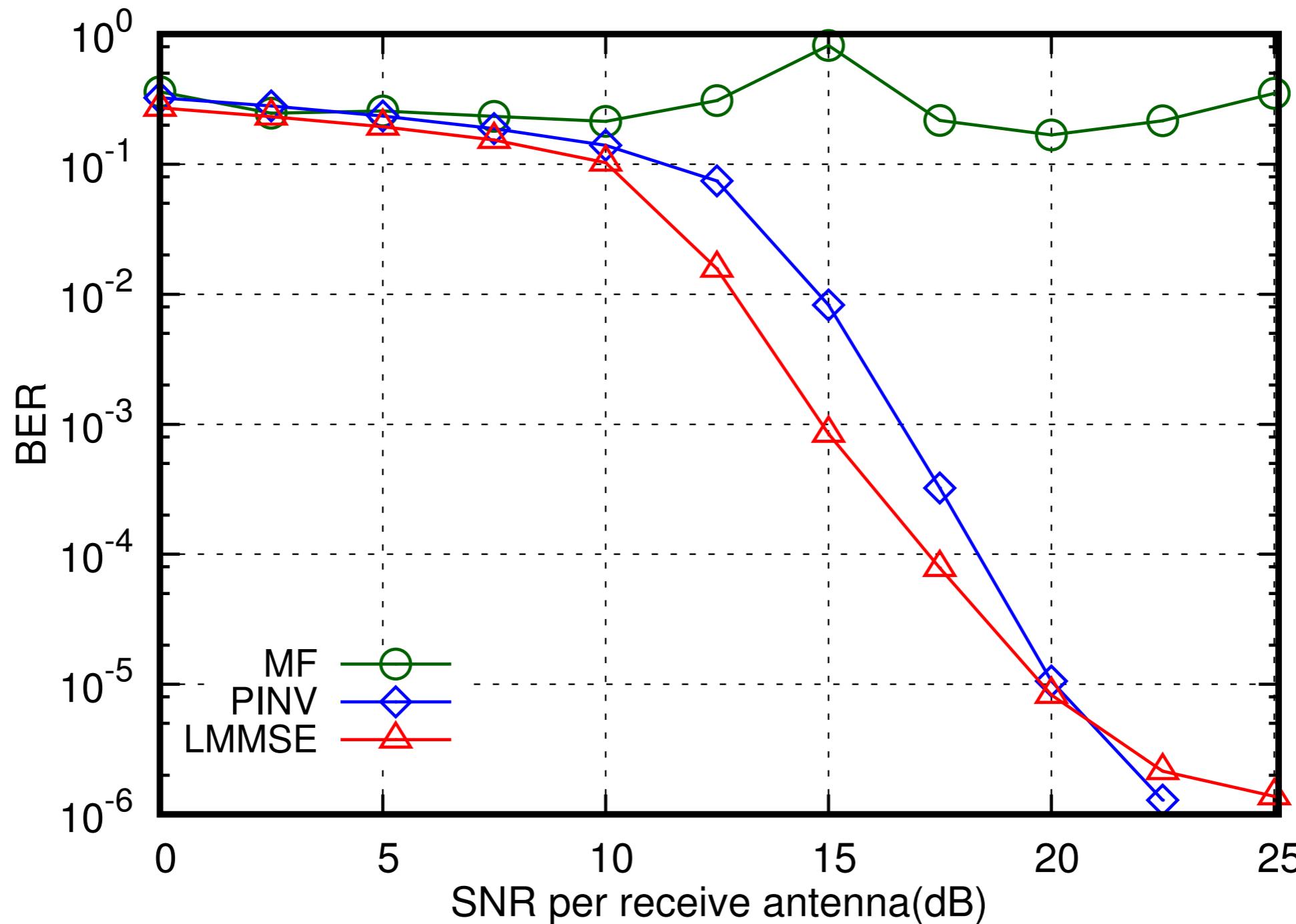
$$r_t = s_t + \underline{\gamma_t} W(y - Hs_t),$$

$$s_{t+1} = \tanh\left(\frac{r_t}{|\underline{\theta_t}|}\right),$$

$$W \triangleq H^T(HH^T + \underline{\alpha I})^{-1}$$



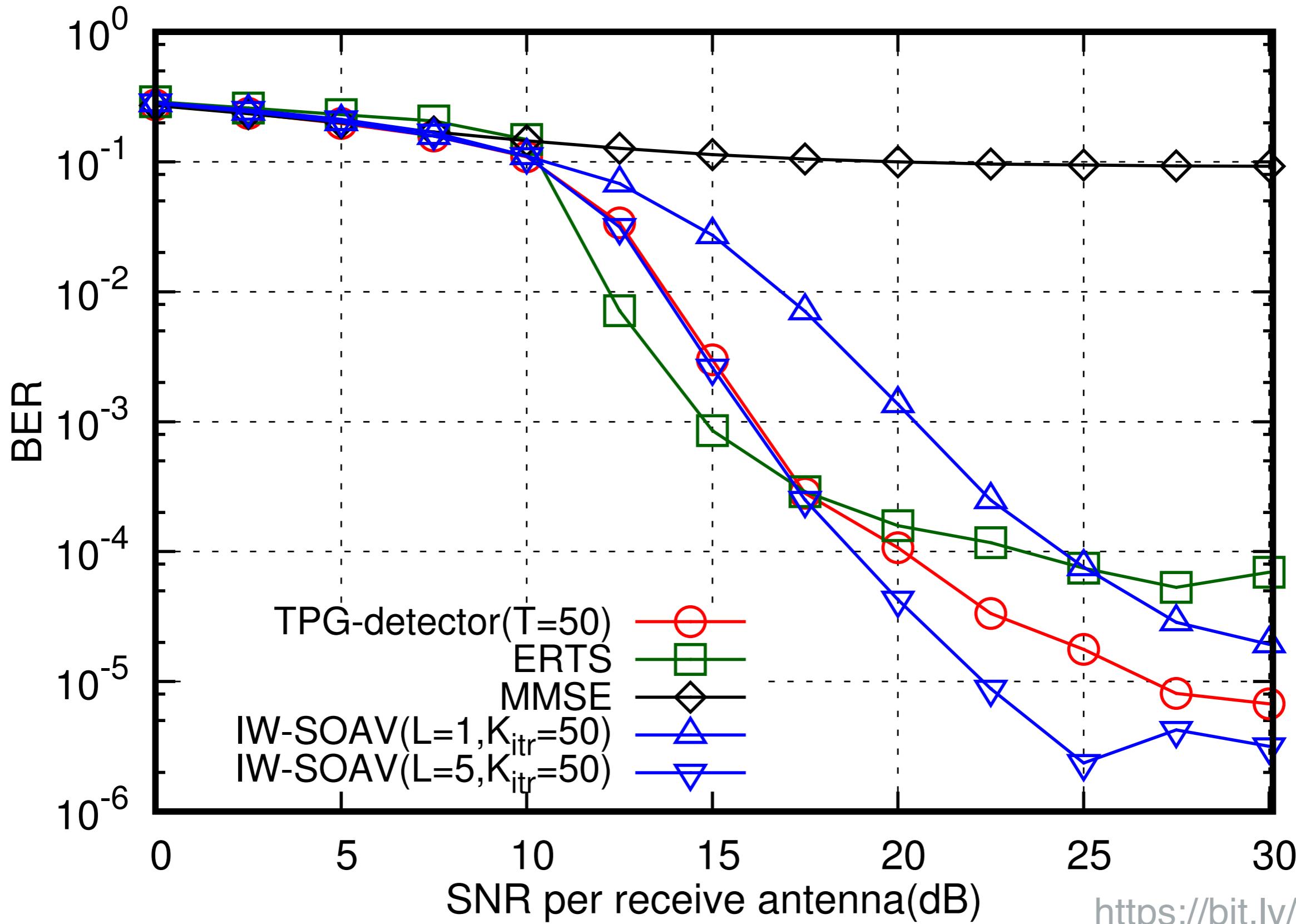
# Wの選択について



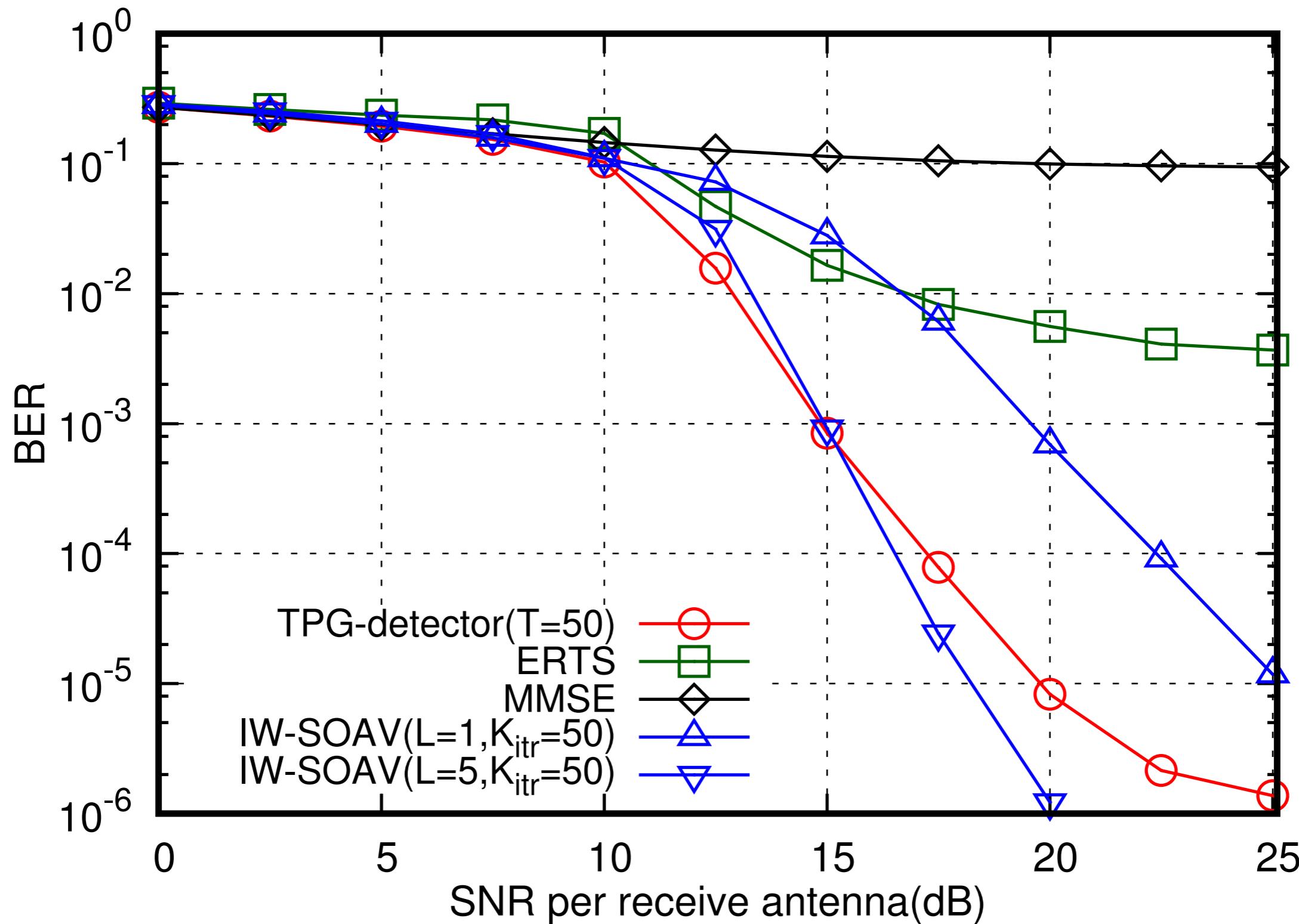
LMMSE型行列の性能が最も良好

<https://bit.ly/2sdNEJ4>

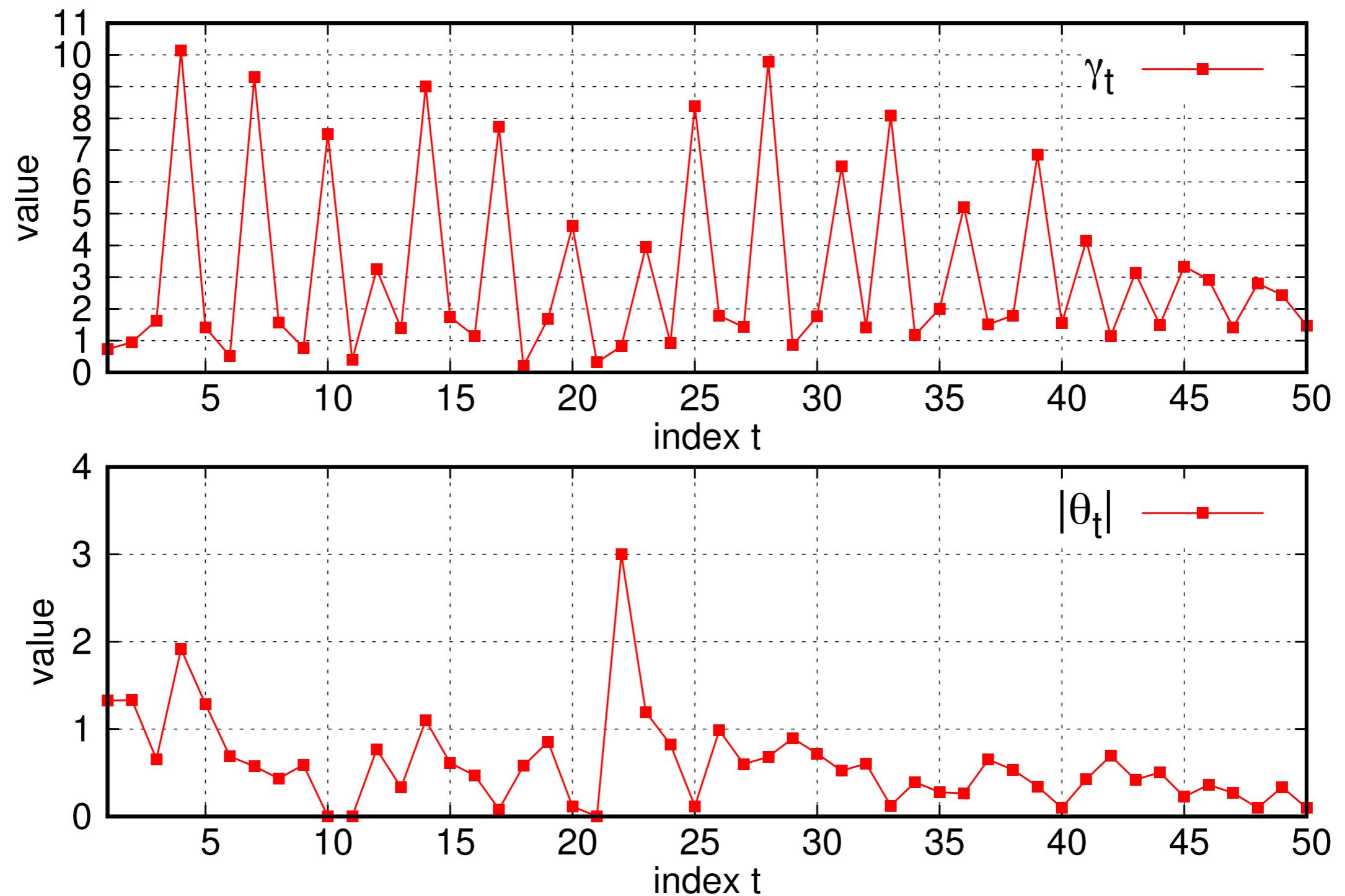
# BER比較 : (n,m)=(100,64)



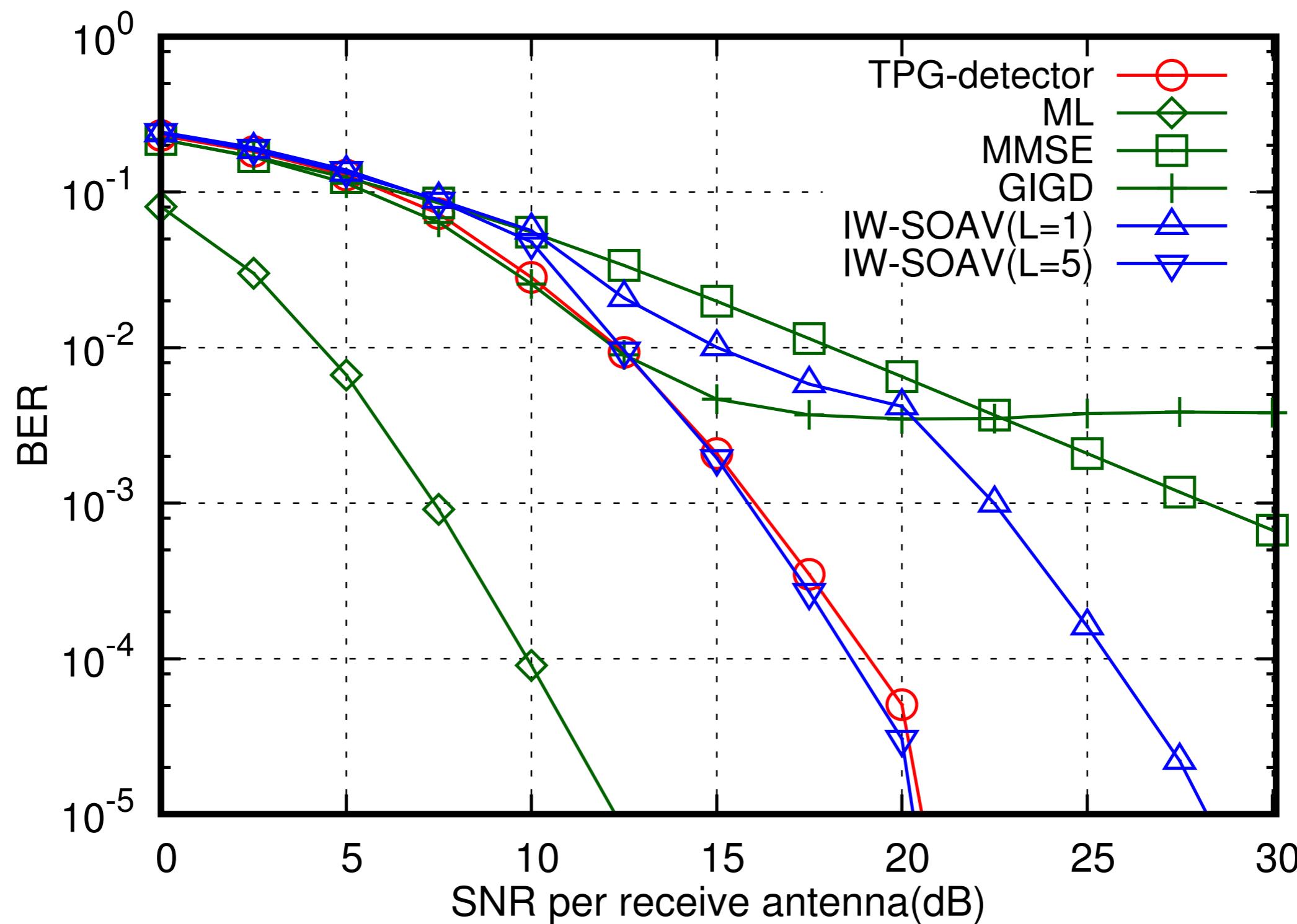
# BER比較 : (n,m)=(150,96)



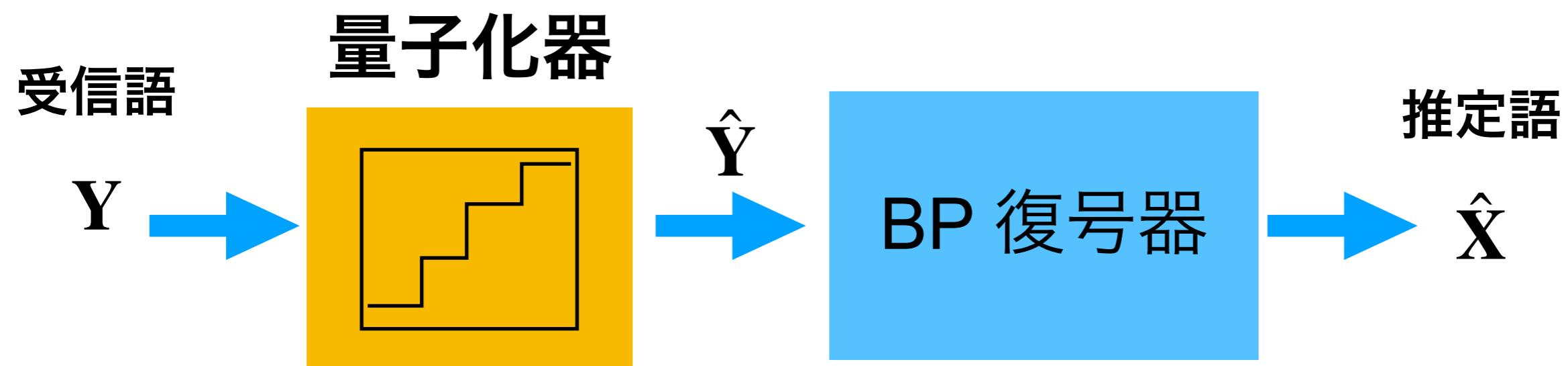
# 学習の結果(学習可能パラメータの値)



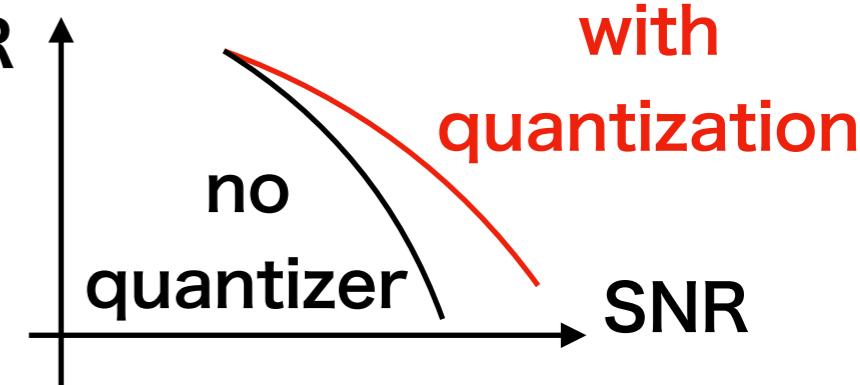
# Non-massiveなケース: (n,m)=(5,5)



# LDPC符号化システムにおける量子化器設計問題

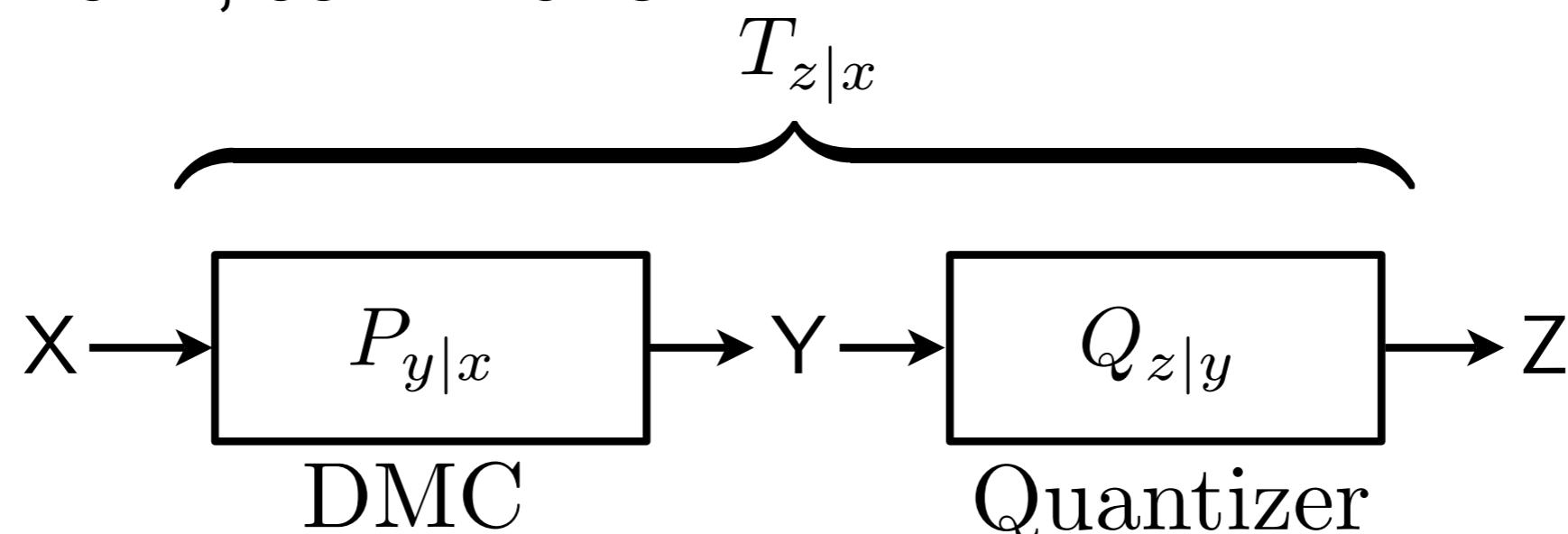


- 低ビットレートADCが省電力の点で 望ましい
- 可能な限りビット誤り率特性の劣化量 が少ないことが望ましい



# Information bottleneck method

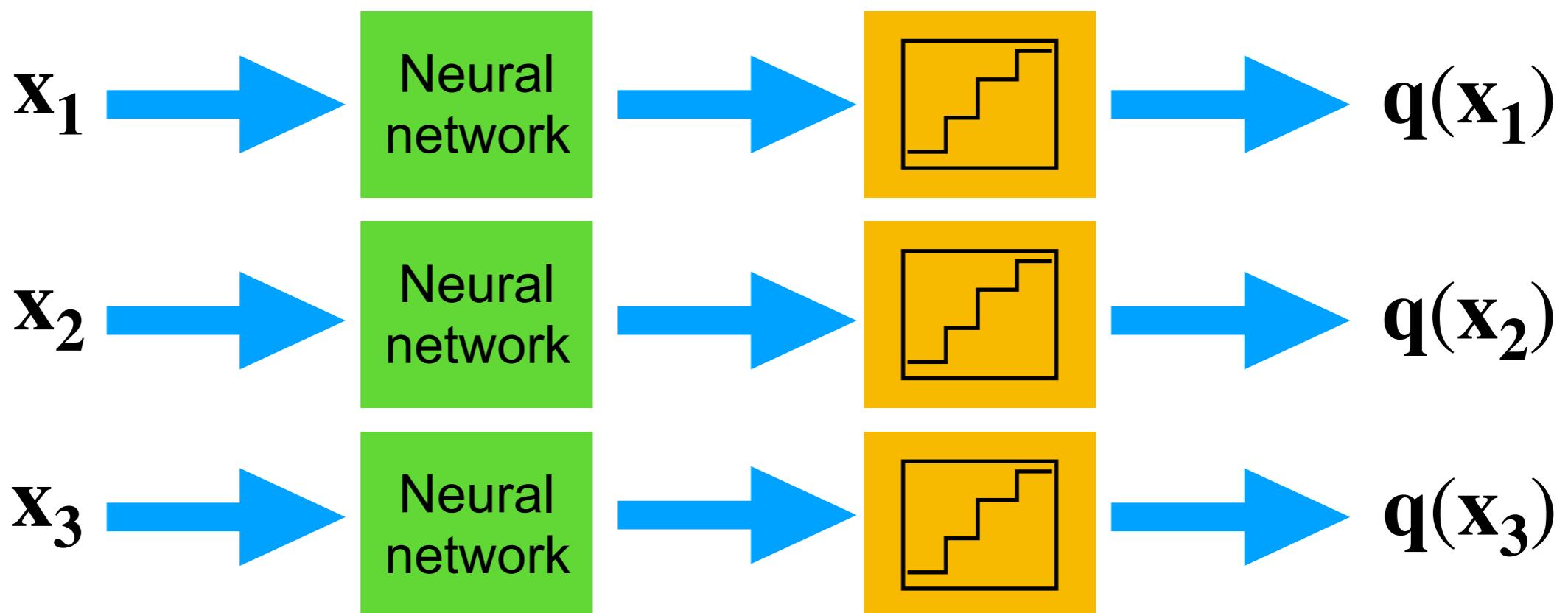
- B. M. Kurkoski and H. Yagi, ``Quantization of binary-input discrete memoryless channels, "IEEE Trans. Inf. Theory, vol. 60, no. 8, pp. 4544-4552, 2014.
- J. Lewandowsky and G. Bauch, ``Information-optimum LDPC decoders based on the information bottleneck method, " IEEE Access, vol. 6, pp. 4054-4071, Jan. 2018.



$$Q^* = \arg \max_{Q \in \mathcal{Q}} I(X; Z),$$

# 基本的なアイデア

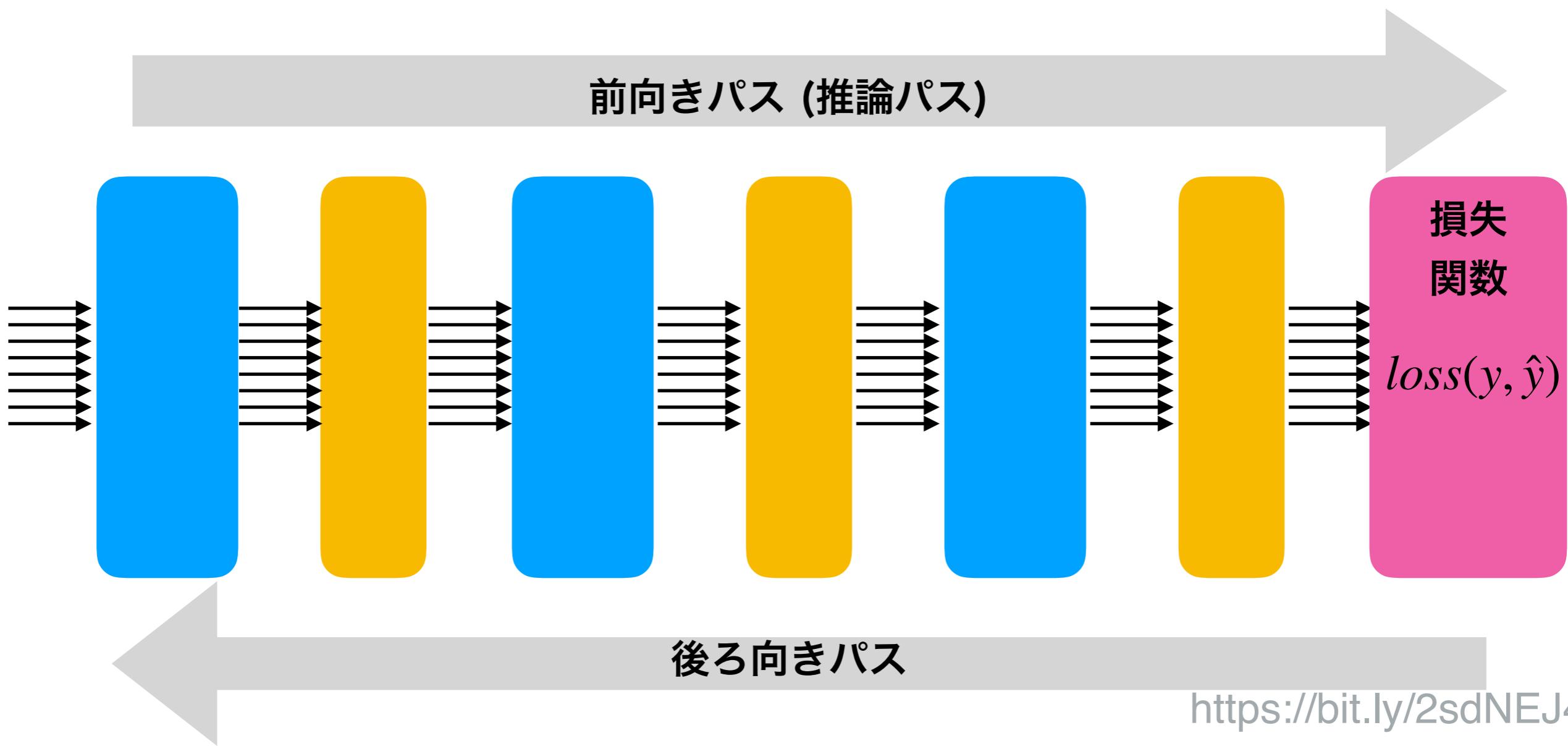
## 階段関数



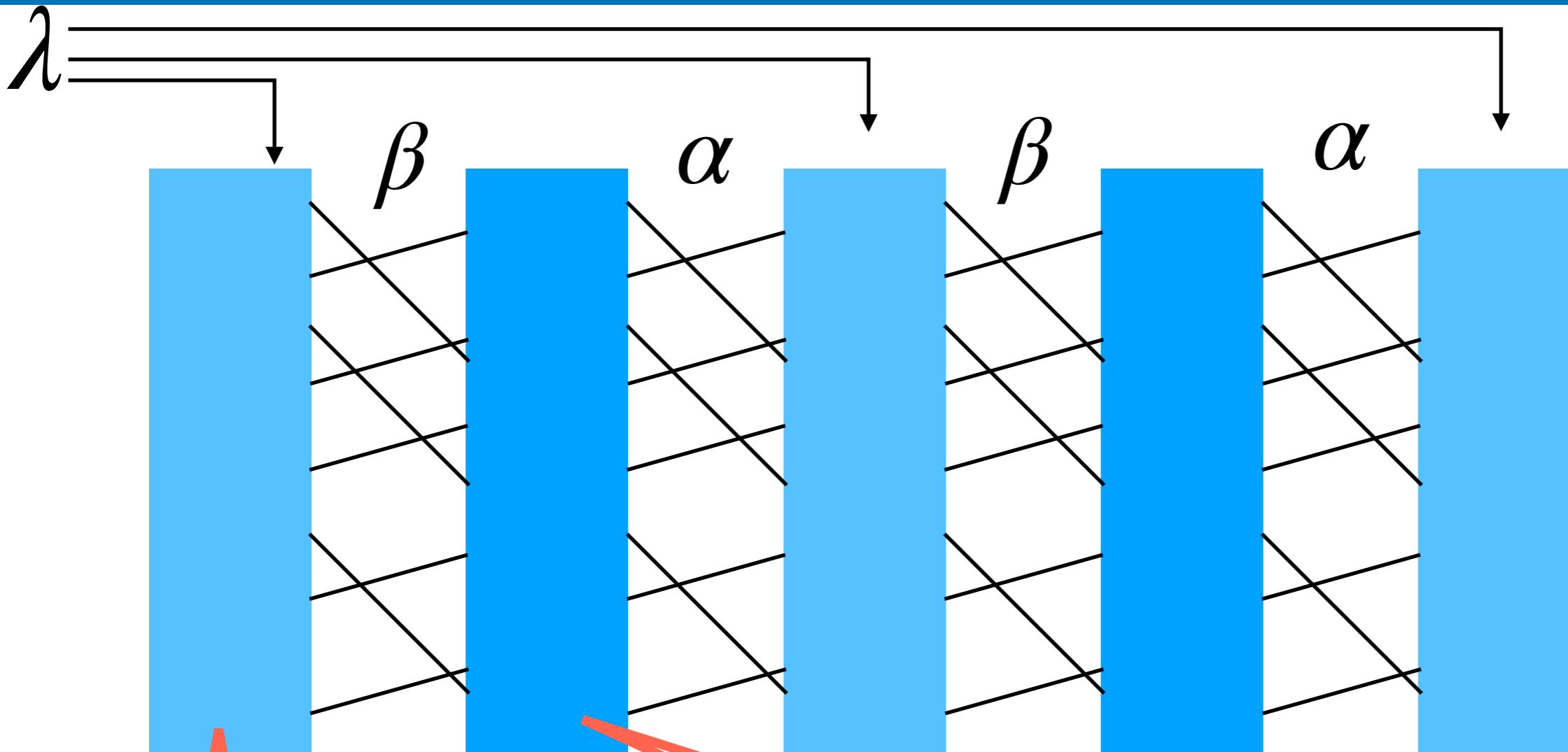
- フィードフォワード型ニューラルネットワークと階段関数の組み合わせ

# バックプロパゲーション

- 学習可能パラメータに関する偏微分係数を計算するための効率良い技法



# BP部分は、backprop可能



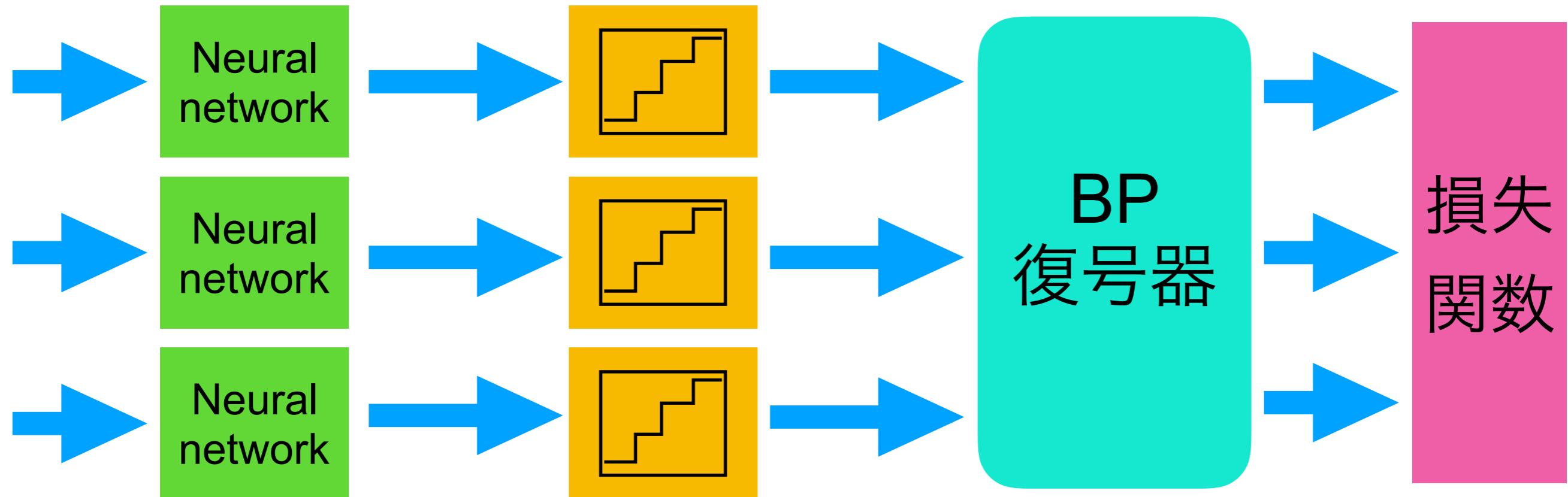
Variable node operation

$$\beta_{j \rightarrow i} = \lambda_j + \sum_{k \in B(j) \setminus i} \alpha_{k \rightarrow j}$$

Check node operation

$$\alpha_{i \rightarrow j} = 2 \tanh^{-1} \left( \prod_{k \in A(i) \setminus j} \tanh \left( \frac{1}{2} \beta_{k \rightarrow i} \right) \right)$$

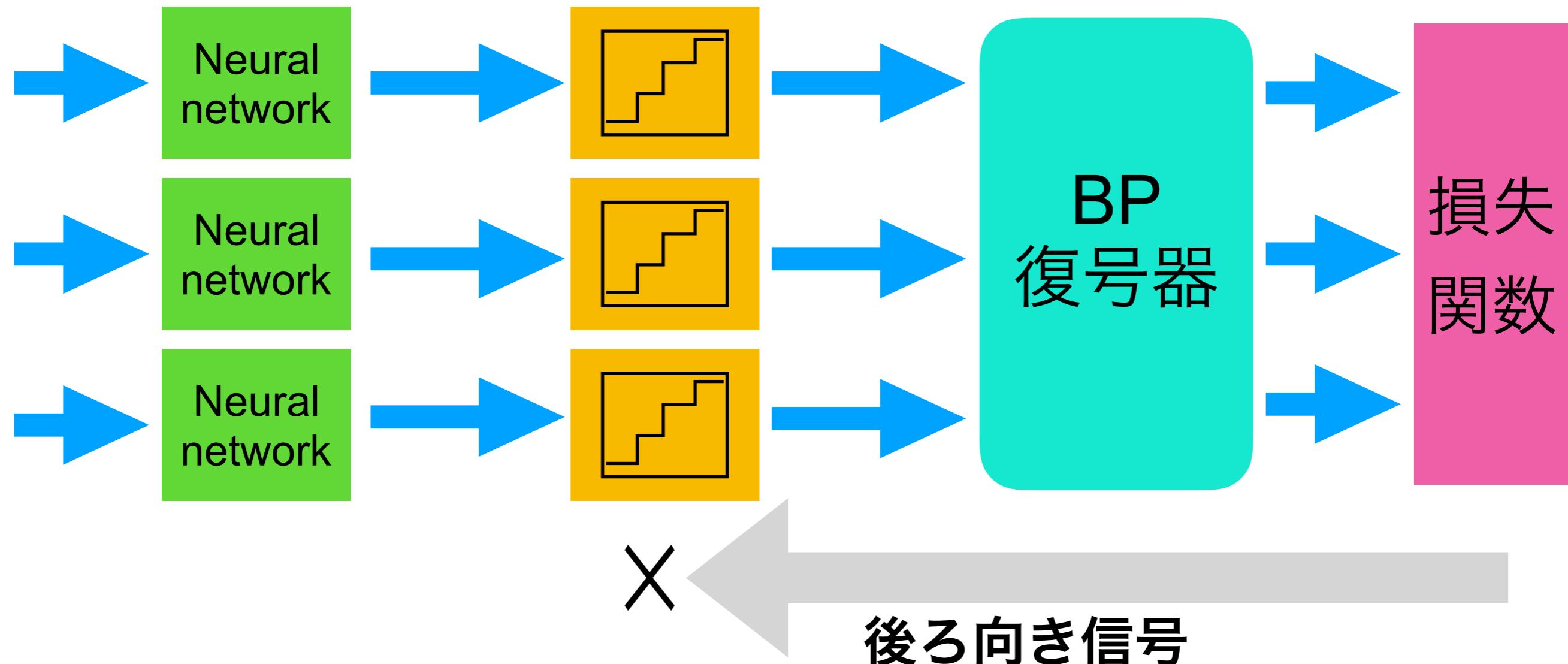
# 前向きパスは問題なし



$$h_1 = \text{relu}(W_1 y + b_1)$$

$$h_i = \text{relu}(W_i h_{i-1} + b_i),$$

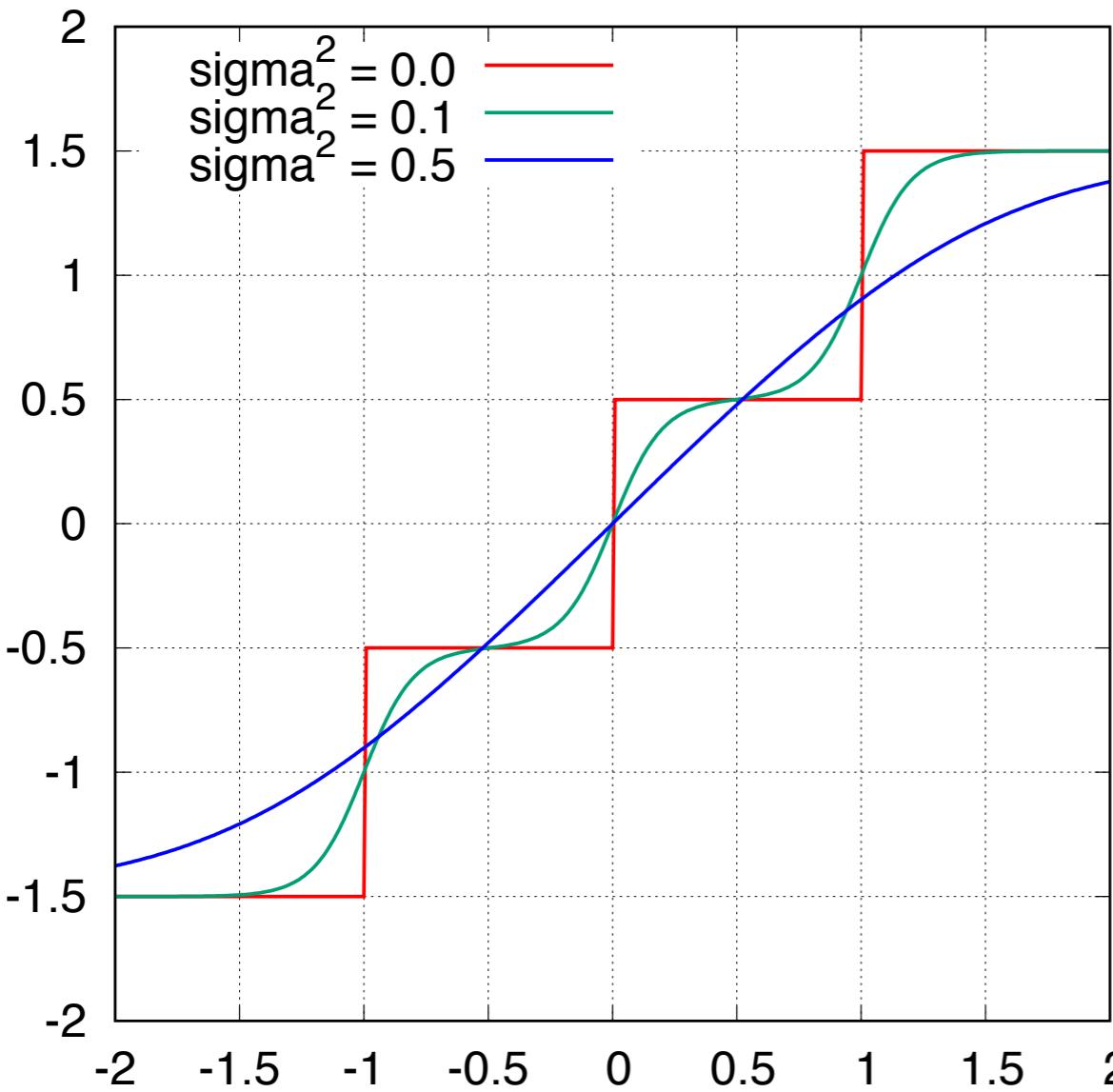
# 後ろ向きパスが失敗



階段関数は、ほとんど至る所で導関数値が  
ゼロ

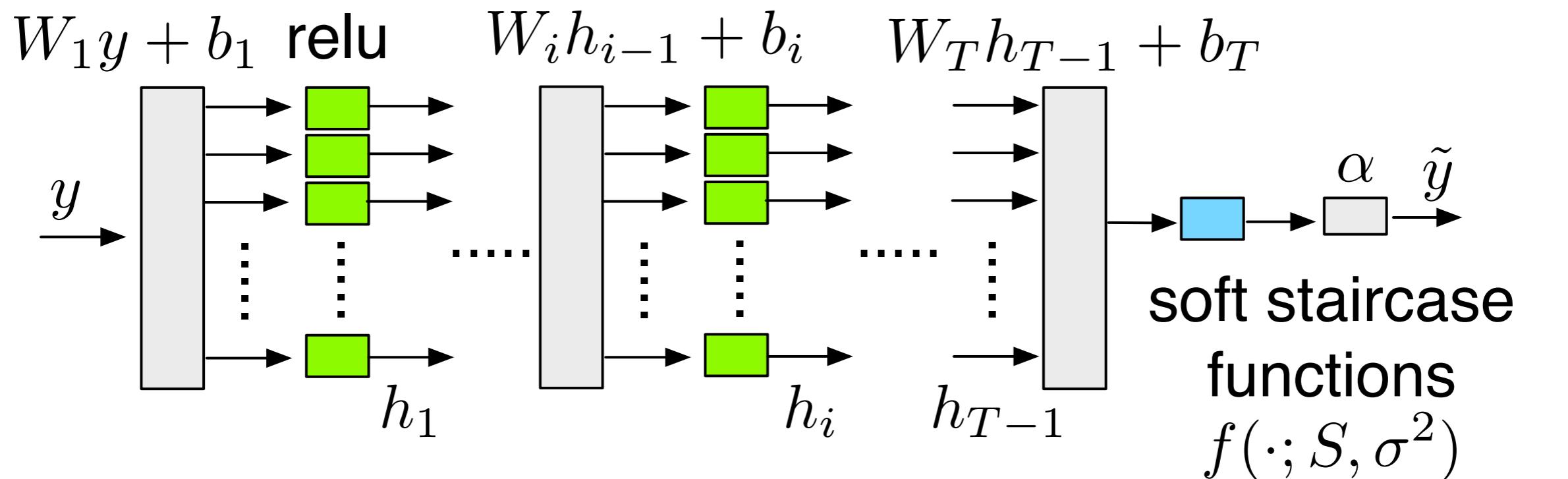
# ソフト階段関数

$$f(r; S, \sigma^2) \triangleq \frac{\sum_{s \in S} s \exp\left(-\frac{(r-s)^2}{2\sigma^2}\right)}{\sum_{s \in S} \exp\left(-\frac{(r-s)^2}{2\sigma^2}\right)}, \quad \sigma^2 > 0,$$



- 微分可能
- 導関数値が非ゼロ
- 多值信号に対する  
MMSE推定関数
- 柔らかさを制御できる

# ニューラル量子化器



フィードフォワードNN

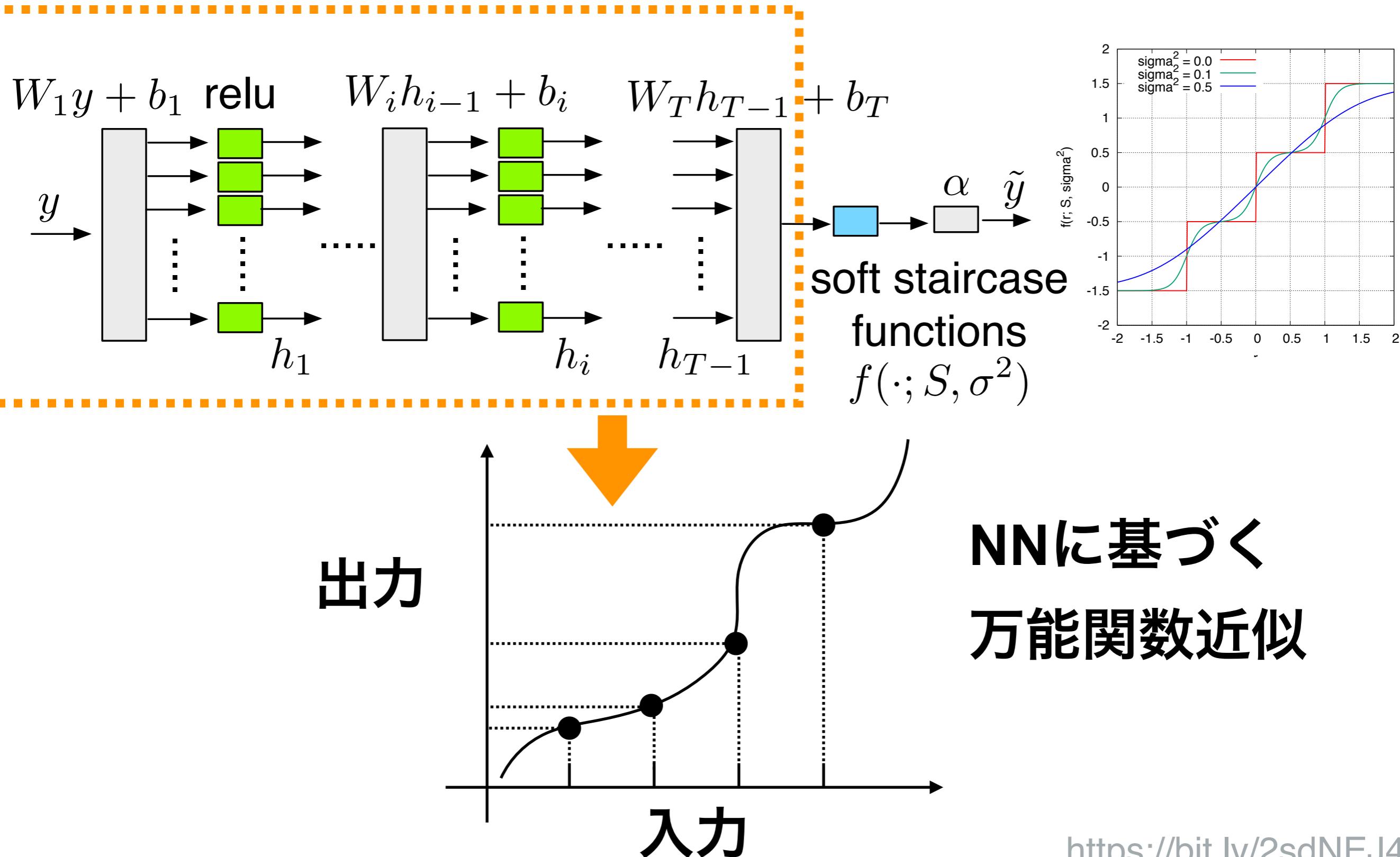
$$h_1 = \text{relu}(W_1y + b_1)$$

$$h_i = \text{relu}(W_i h_{i-1} + b_i),$$

ソフト階段

関数

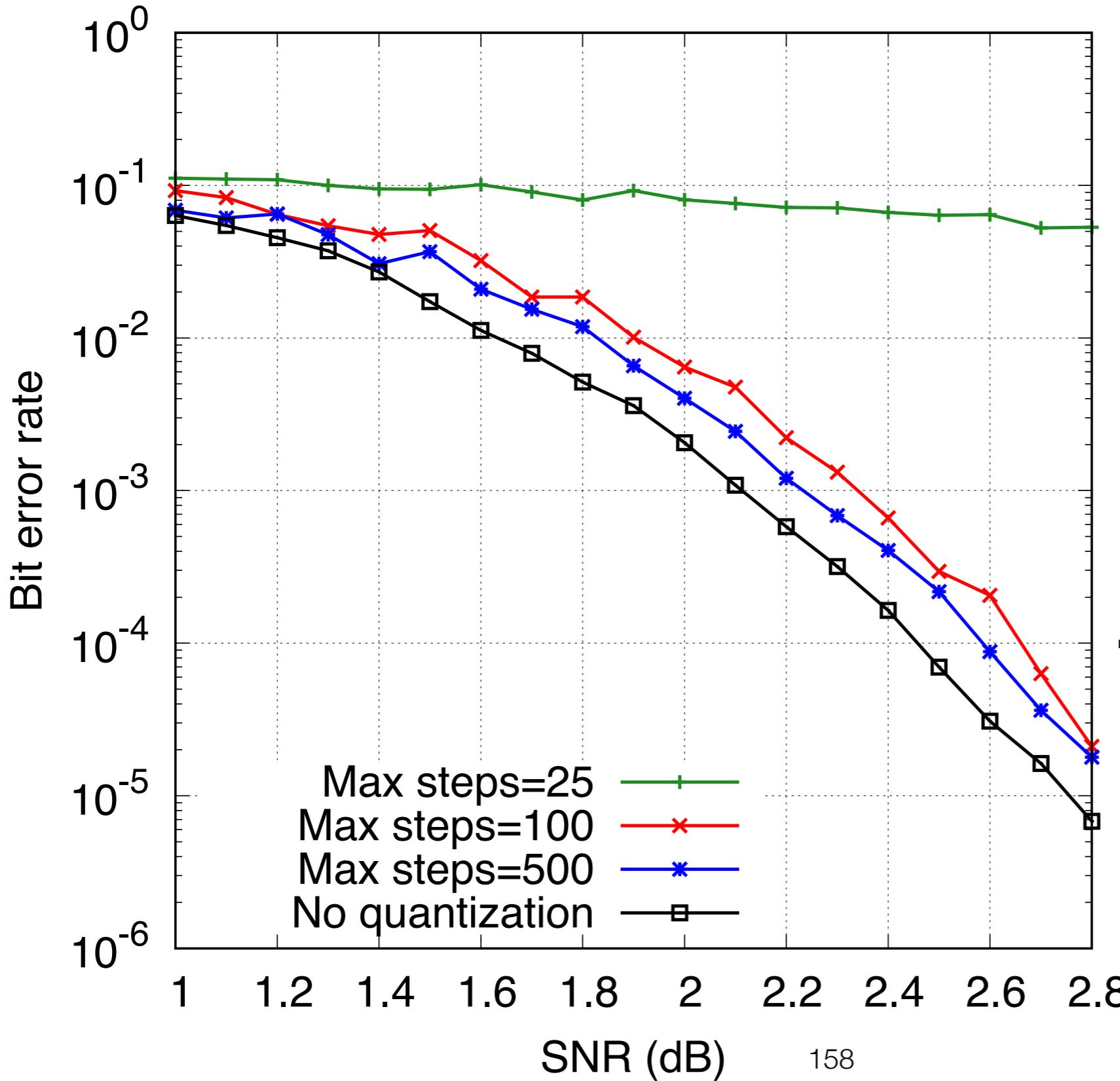
# 非線形座標変換器としてのNN



# アニーリングを含む学習プロセス

- ミニバッチ学習
- Adam オプティマイザ
- 二乗誤差
- AWGN通信路モデルに基づくランダムサンプル
- アニーリングスケジュール  $\sigma^2 = t^\eta$

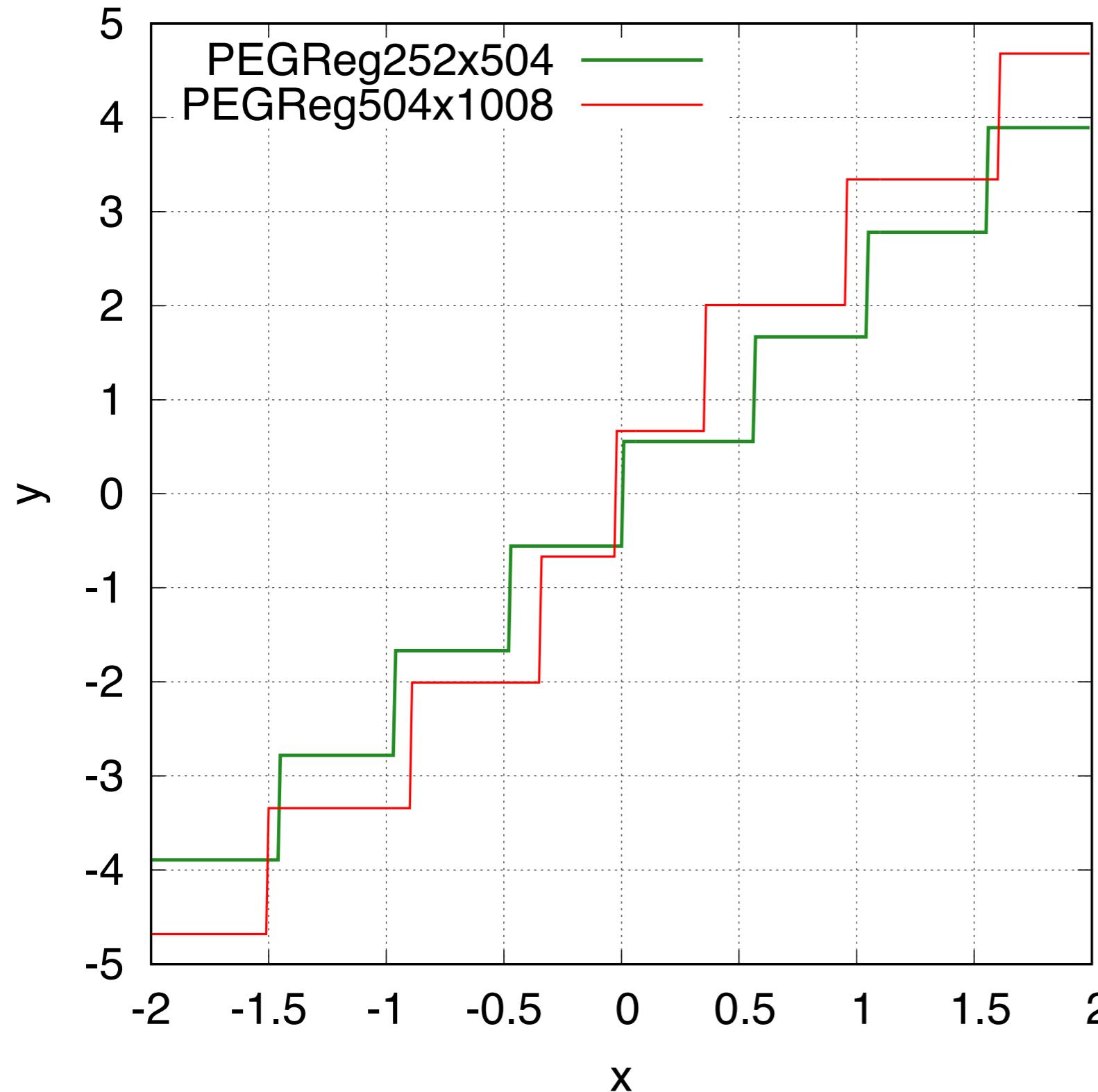
# BER性能の比較



8-level trained  
neural quantizer  
PEGReg504x1008  
 $n = 1008, m = 504$

$u = 8$  (# of hidden units)  
 $T = 2$  (# of NN layers)

# 訓練の結果として得られた量子化器



**8-level trained  
neural quantizer**

**CODE1**

**PEGReg504x1008**

**n = 1008, m = 504**

**CODE2**

**PEGReg252x504**

**n = 504, m = 252**

# 今後の課題

-  データ駆動加速に関する理解の深化
-  スパースNOMA(LDS, SCMA)(LDSについて検討中)
-  射影勾配法に基づくLDPC符号の復号法(3月 or 5月情報理論研究会にて発表予定)

# 全体のまとめ



通信工学においても、深層学習技術は使える  
→アルゴリズム設計の可能性が広がる



End-to-Endアプローチに基づく実環境に  
柔軟に適応する通信系アルゴリズムの実現



データ駆動アプローチによる反復アルゴリズム  
の改善

物理層無線通信技術へ深層学習は徐々に導入されていく  
可能性が高いと予想します  
簡単に試すことができますので、ぜひお試しください！