

# 無線通信のための深層学習の基礎

和田山 正

wadayama@nitech.ac.jp

名古屋工業大学

# 自己紹介

和田山 正

名古屋工業大学大学院工学研究科 教授  
(情報工学専攻, 情報工学科担当)

**研究上の興味:** 誤り訂正符号(特にLDPC符号),  
無線通信+深層学習,  
信号処理(圧縮センシング)

# 無線物理層におけるDL+Comの動き

- ICC2018, Globecom2018, ICC2019 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
- 各論文誌の特集号(IEEE Access など)
- COMSOC Emerging Technology Initiative  
``Machine Learning for Communications''

# 機械学習技術を利用した無線通信技術

- ✓ 無線通信技術 + 機械学習(ML)を組み合わせた研究が急増中→なぜ？

## 深層学習(ディープラーニング)の登場

深層学習は、画像認識・音声認識・自然言語処理などの分野ですでに基盤技術となっている。他分野への応用が広がり始めている  
→深層学習技術の持つ普遍性

## より高度な信号処理への要求 (5G/6Gに向けて)

mmWave massive MIMO, ブラインド通信路推定, 誤り訂正,  
端末位置推定, ビームフォーミング, NOMA(過負荷検出), 干渉制御

## 近未来におけるAI/MLと無線との融合

AI/MLが主たる無線通信通信のタスクになるのでは？

(分散学習のためのインフラ) また、高度無線ネットワークをAI/MLが支える

## 深層学習

- 画像認識・音声認識・自然言語処理で圧倒的な強みを見せている
- 機械学習と関連が薄いと思われていた分野にも急速に波及中→医療診断、材料科学、無線通信
- 通信系国際会議においても、機械学習関連の研究が激増中

# 無線通信技術

- 5G→6Gにおいて、求められる技術はますます高度化の一途をたどっている
- ビーム形成・ユーザ位置推定・電力制御・スペクトルセンシング・NOMA・Massive MIMO検出・高度な誤り訂正処理。。。
- 従来技術の延長線上にない新しいアプローチへの期待

# ひとつの領域だけでも大変なのに、二つも？

- チャンスは境界分野にあり！
- よいフレームワーク(PyTorch, TensorFlowなど)の登場で深層学習に関する研究の敷居はさがってきた→通信研究者が自分で十分にできる
- 深層学習が強い従来ドメイン(画像・音声など)では比較的技術が成熟してきているが、無線通信・信号処理ではまだまだこれから！

# 無線通信技術 × 深層学習: 研究スタートガイド

- 両分野のクロスする部分に興味をお持ちの方が研究をスムースにスタートするために役立つ情報を提供したいと思います
- 初学者の方を意識して、あまりテクニカルになりすぎないように(後半は少しテクニカル)
- 具体的なコード(PyTorch)を提示しつつ (ハンズオン的) ご説明します

# 本チュートリアル関連の参考コード類

<https://github.com/wadayama/MIKA2019>

wadayama / MIKA2019

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

repository for MIKA2019 Edit

Manage topics

37 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find File Clone or download

Tadashi Wadayama and Tadashi Wadayama notes revised Latest commit d2f32bd 17 days ago

File	Description	Time
.ipynb_checkpoints	notes revised	17 days ago
ANDfunction.ipynb	notes revised	17 days ago
Egg.ipynb	notes revised	17 days ago
ISTA.ipynb	notes revised	17 days ago
MIKA2019.pdf	text updated	20 days ago
MIMO.ipynb	notes revised	17 days ago
NoisyGDBF.ipynb	text revised	24 days ago
PEGReg504x1008.dec	codes added	25 days ago
ProjectedGradient.ipynb	notes revised	17 days ago
README.md	text revised	24 days ago
autoencoder.ipynb	notes revised	17 days ago
complexlib.py	codes added	25 days ago
distributed.ipynb	notes revised	17 days ago
fftista.ipynb	text revised	24 days ago

# 本チュートリアルの構成

- 第一部：

- (a) 研究環境を整える
- (b) AND関数を学習する
- (c) MIMO検出器を作る

Q&A

- 第二部：

- (a) 深層展開
- (b) スパース信号復元とISTA(近接勾配法)
- (c) 深層展開に関する関連研究

Q&A

# 本チュートリアルの構成

- 第一部：

- (a) 研究環境を整える
- (b) AND関数を学習する
- (c) MIMO検出器を作る

Q&A

- 第二部：

- (a) 深層展開
- (b) スパース信号復元とISTA(近接勾配法)
- (c) 深層展開に関する関連研究

Q&A

# 本チュートリアルの構成

- 第一部：

- (a) 研究環境を整える

- (b) AND関数を学習する

- (c) MIMO検出器を作る

- Q&A

- 第二部：

- (a) 深層展開

- (b) スパース信号復元とISTA(近接勾配法)

- (c) 深層展開に関する関連研究

- Q&A

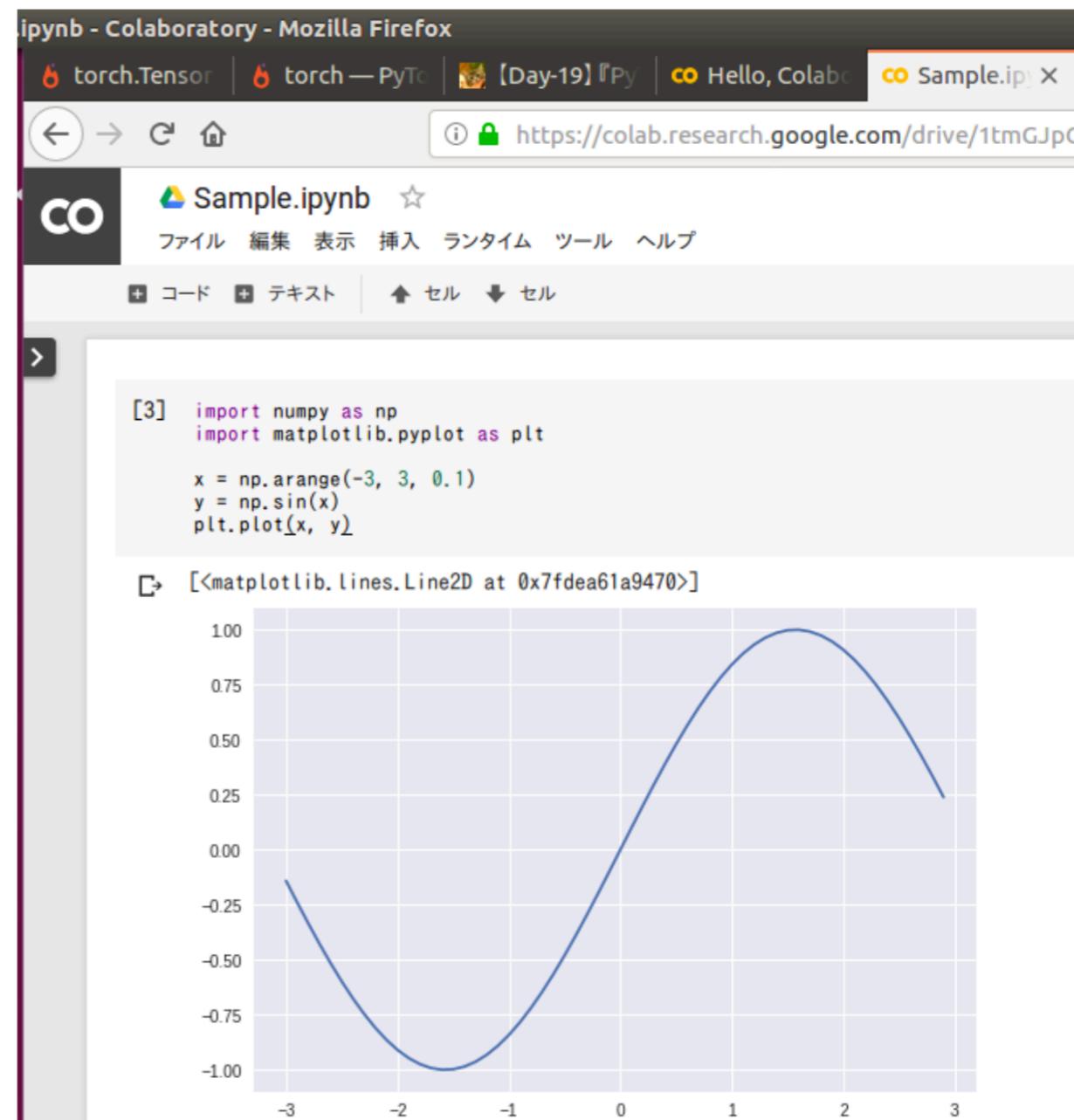
# 深層学習の実行環境

	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)

\* 手元にインストールをして利用したい場合は、Anacondaが便利

# Google Colaboratory

- ✓ 機械学習の教育・研究促進のため、Google の開発したPython 実行環境(Jupyter notebook ベース)
- ✓ 無償でクラウド上のマシンを利用可能(GPUも利用できる)
- ✓ ブラウザベースで環境構築不要でフレームワークが利用できる
- ✓ 主だったパッケージはインストール済



A screenshot of a Google Colaboratory notebook titled "Sample.ipynb". The code cell contains the following Python code:

```
[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)
```

The output of the cell is a line plot of the sine function, showing a wave from x = -3 to x = 3.

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

# 深層学習フレームワーク

- 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

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

# PyTorchへのアプローチ

本家サイトのチュートリアルがわかりやすい

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

# 本チュートリアルの構成

- 第一部：

- (a) 研究環境を整える
- (b) **AND関数を学習する**
- (c) MIMO検出器を作る

Q&A

- 第二部：

- (a) 深層展開
- (b) スパース信号復元とISTA(近接勾配法)
- (c) 深層展開に関する関連研究

Q&A

# AND関数を学習する



AND関数の真理値表

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

ここで考える学習タスク:  
ニューラルネットでAND関数を模擬



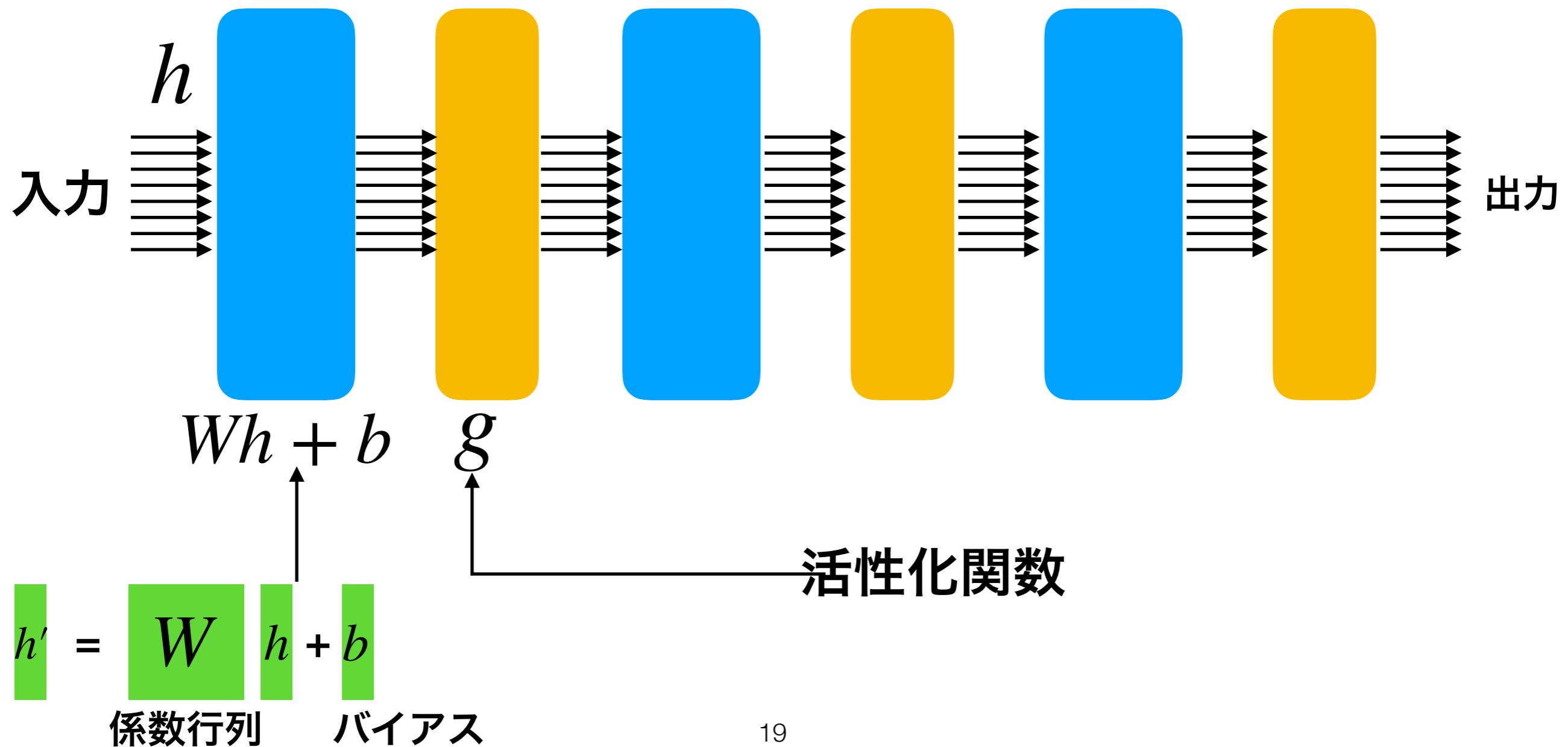
データセット:  $\{(0,0), 0\}, \{(0,1), 0\}, \{(1,1), 1\}, \dots$

# 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

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



# アフィン変換

$$h' = W h + b$$

シグモイド関数

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

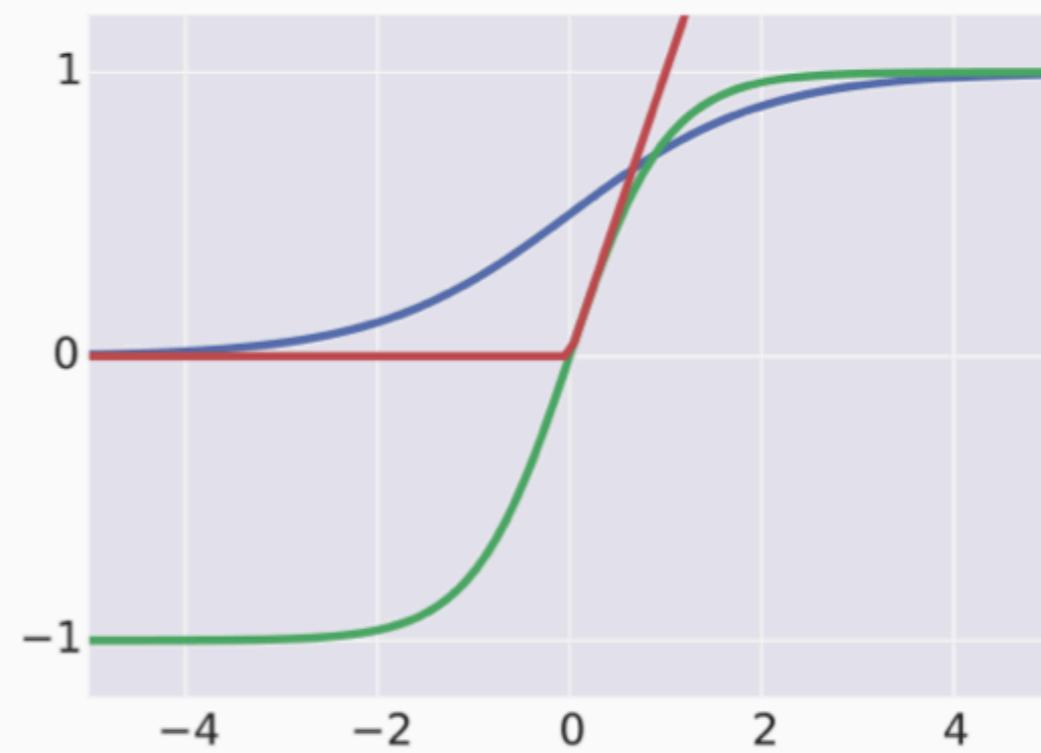
双曲線正接関数

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

ランプ関数

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

## 活性化関数



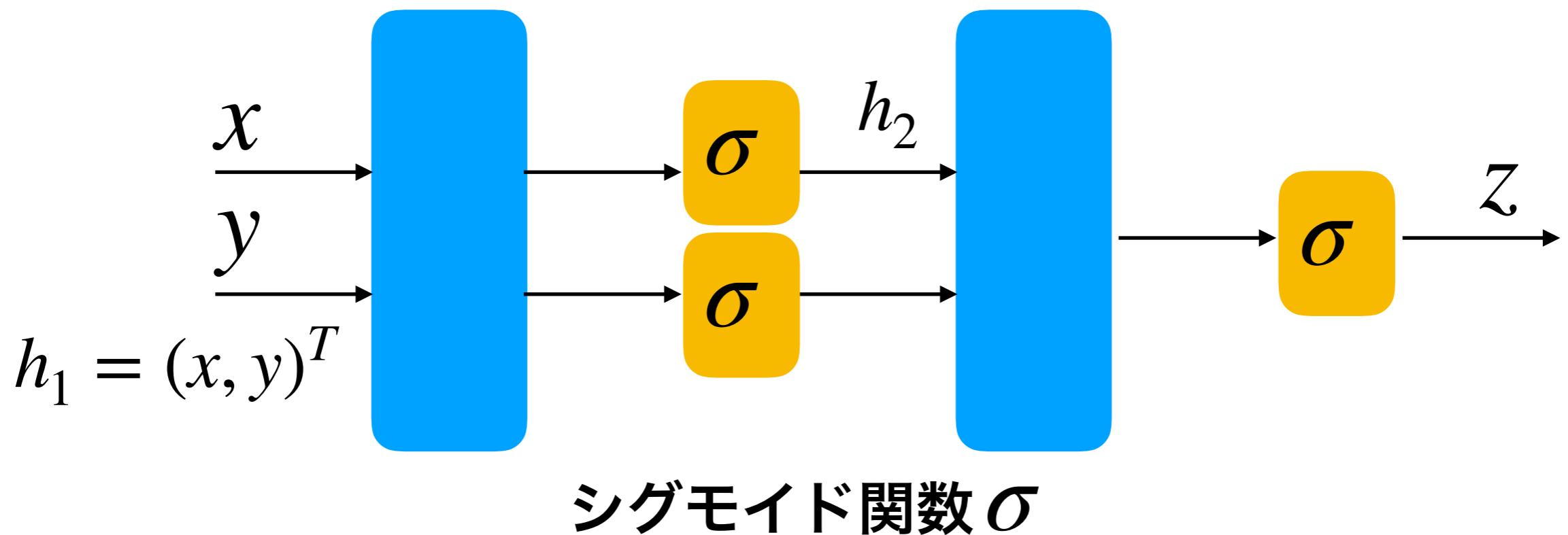
# AND関数のモデル

$$W_1 \in \mathbb{R}^{2 \times 2}$$

$$W_1 h_1 + b_1$$

$$W_2 \in \mathbb{R}^{1 \times 2}$$

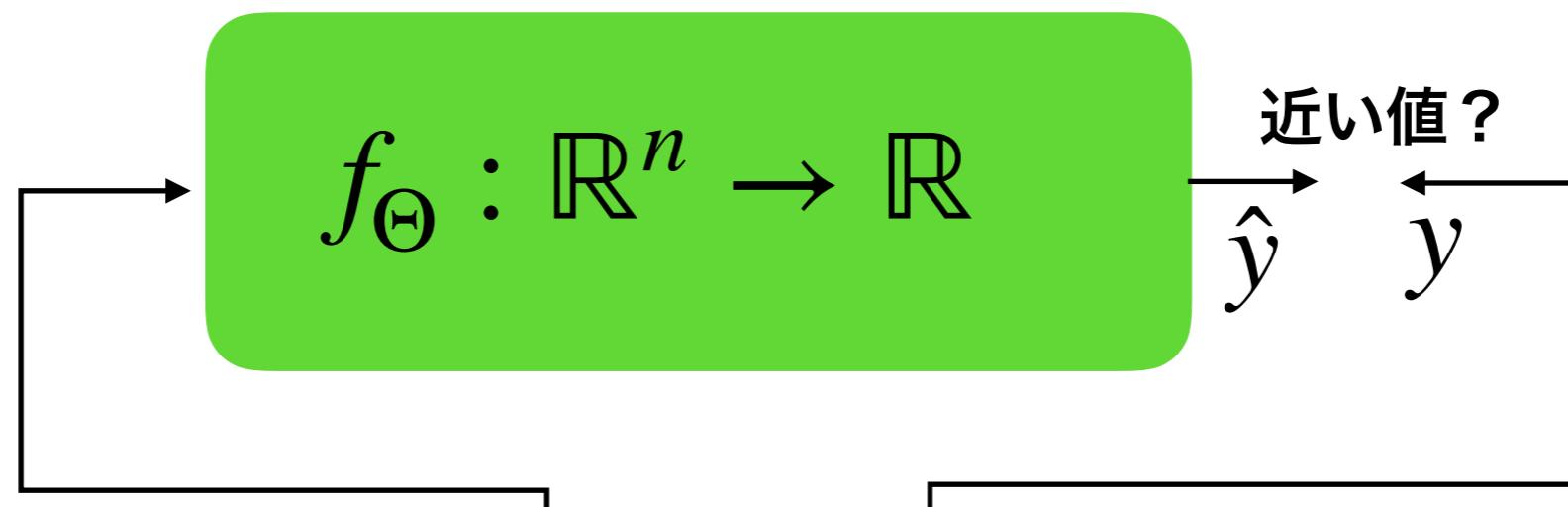
$$W_2 h_2 + b_2$$



$\{W_1, W_2, b_1, b_2\}$  は調整可能 → 「AND関数」に近づくように調整

# 訓練プロセス

出力と教師ラベルとの間の**食い違い(損失関数値)**がなるべく  
小さくなるように学習可能パラメータを更新する



$\{(0,0), 0\}$   
 $\{(0,1), 0\}$   
 $\{(1,1), 1\}$   
... 教師信号

# 確率的勾配法

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

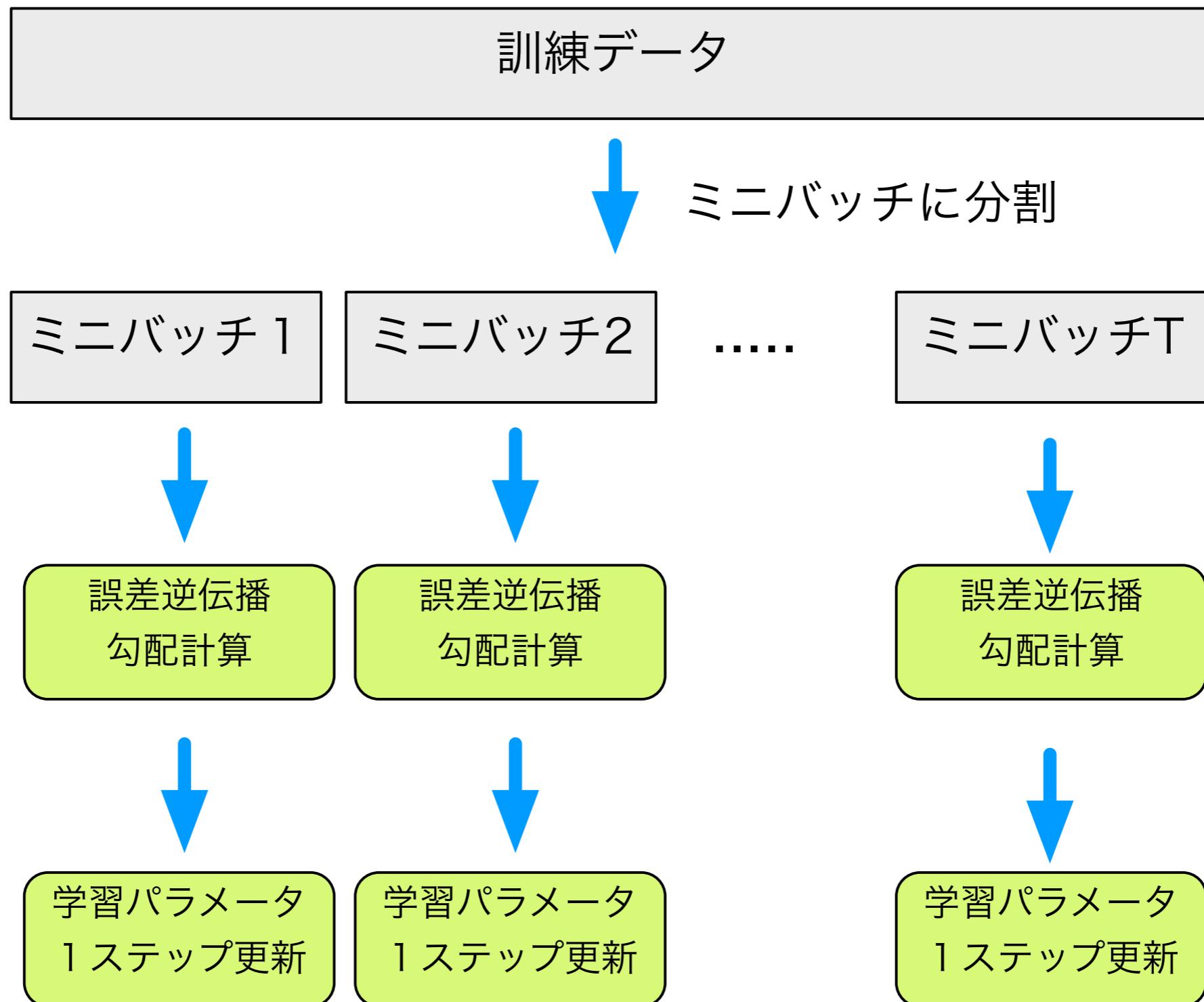
Step 3 (勾配ベクトルの計算)  $g := \nabla G_B(\Theta)$

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**

# 訓練プロセス

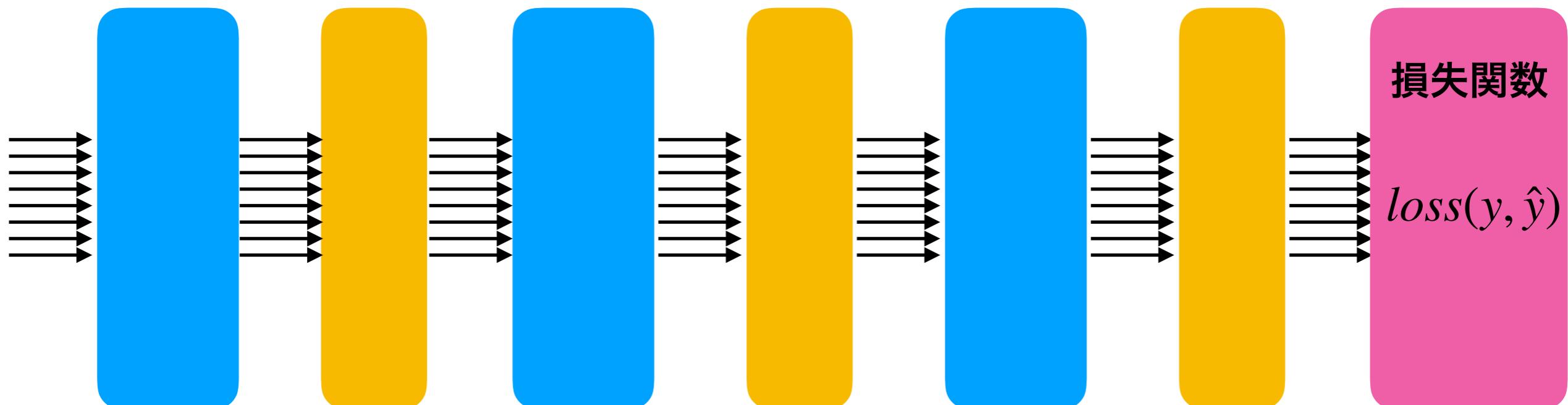


# 誤差逆伝播法

- ・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)

前向き計算フェーズ（単なる値の評価）

注：ただし途中の計算結果を残す



後ろ向き計算フェーズ

ヤコビ行列とベクトルの積を順次計算

# AND学習のコードを見てみる

## AND関数の学習

Le display

 Open in Colab

本ノートブックでは、ニューラルネットワークによりAND関数 AND(a,b)の学習を行う。

### 必要なパッケージのインポート

```
In [ ]: import torch # テンソル計算など  
import torch.nn as nn # ネットワーク構築用  
import torch.optim as optim # 最適化関数
```

PyTorch利用の場合  
にはインポート

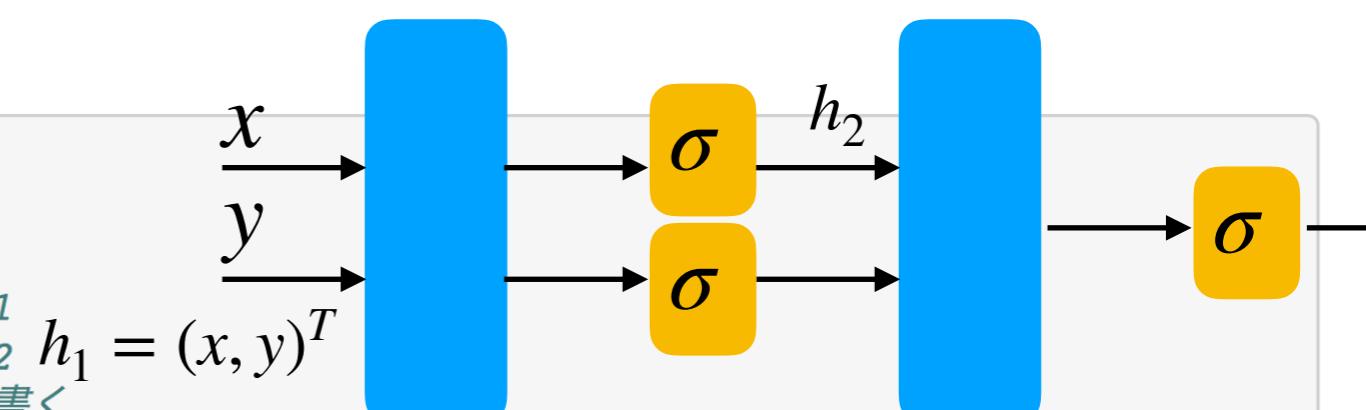
### グローバル定数の設定

```
In [ ]: mbs = 5 # ミニバッチサイズ
```

ミニバッチ学習を利用するので  
バッチサイズを5と設定

### ネットワークの定義

```
In [ ]: class Net(nn.Module): # nn.Module を継承  
    def __init__(self): # コンストラクタ  
        super(Net, self).__init__()  
        self.fc1 = nn.Linear(2, 2) # w_1, b_1  
        self.fc2 = nn.Linear(2, 1) # w_2, b_2  
        h1 = (x, y)T  
    def forward(self, x): # 推論計算をforwardに書く  
        x = torch.sigmoid(self.fc1(x)) # 活性化関数としてシグモイド関数を利用  
        x = torch.sigmoid(self.fc2(x))  
        return x
```



前向き計算をforwardに書く

# ミニバッチ生成部分

## ミニバッチ生成関数

ミニバッチのサイズがmbs × 2 である点に注意

```
In [ ]: def gen_minibatch():
    inputs = torch.bernoulli(0.5 * torch.ones(mbs, 2)) # (0,1)ランダム乱数テンソル(サイズ mbs x 2)
    result = torch.Tensor(mbs, 1)
    for j in range(mbs):
        if (inputs[j, 0] == 1.0) and (inputs[j, 1] == 1.0): # AND関数
            result[j] = 1.0
        else:
            result[j] = 0.0
    return inputs, result
```

```
In [ ]: inputs, result = gen_minibatch() # ミニバッチ生成の実行例
print('inputs = ', inputs)
print('result = ', result)
```

ランダムに生成したデータセットを利用するので、この部分の実装はカンタン  
実データを利用する場合は適切なデータローダを準備する必要がある

# 訓練ループ

## 訓練ループ

```
model= Net() # ネットワークインスタンス生成
loss_func = nn.MSELoss() # 損失関数の生成(二乗損失関数)
optimizer = optim.Adam(model.parameters(), lr=0.1) # オプティマイザの生成(Adamを利用)
for i in range(1000):
    inputs, result = gen_minibatch() # ミニバッチの生成
    optimizer.zero_grad() # オプティマイザの勾配情報初期化
    outputs = model(inputs) # 推論計算
    loss = loss_func(outputs, result) # 損失値の計算
    loss.backward() # 誤差逆伝播法(後ろ向き計算の実行)
    optimizer.step() # 学習可能パラメータの更新
    if i % 100 == 0:
        print('i =', i, 'loss =', loss.item())
```

訓練ループの構造はどのプログラムでもほとんど同じ  
学習率の設定は、学習の成否に大きく影響を与える

# MNIST数字認識

DLフレームワークの``Hello World'' ?



MNISTデータセット

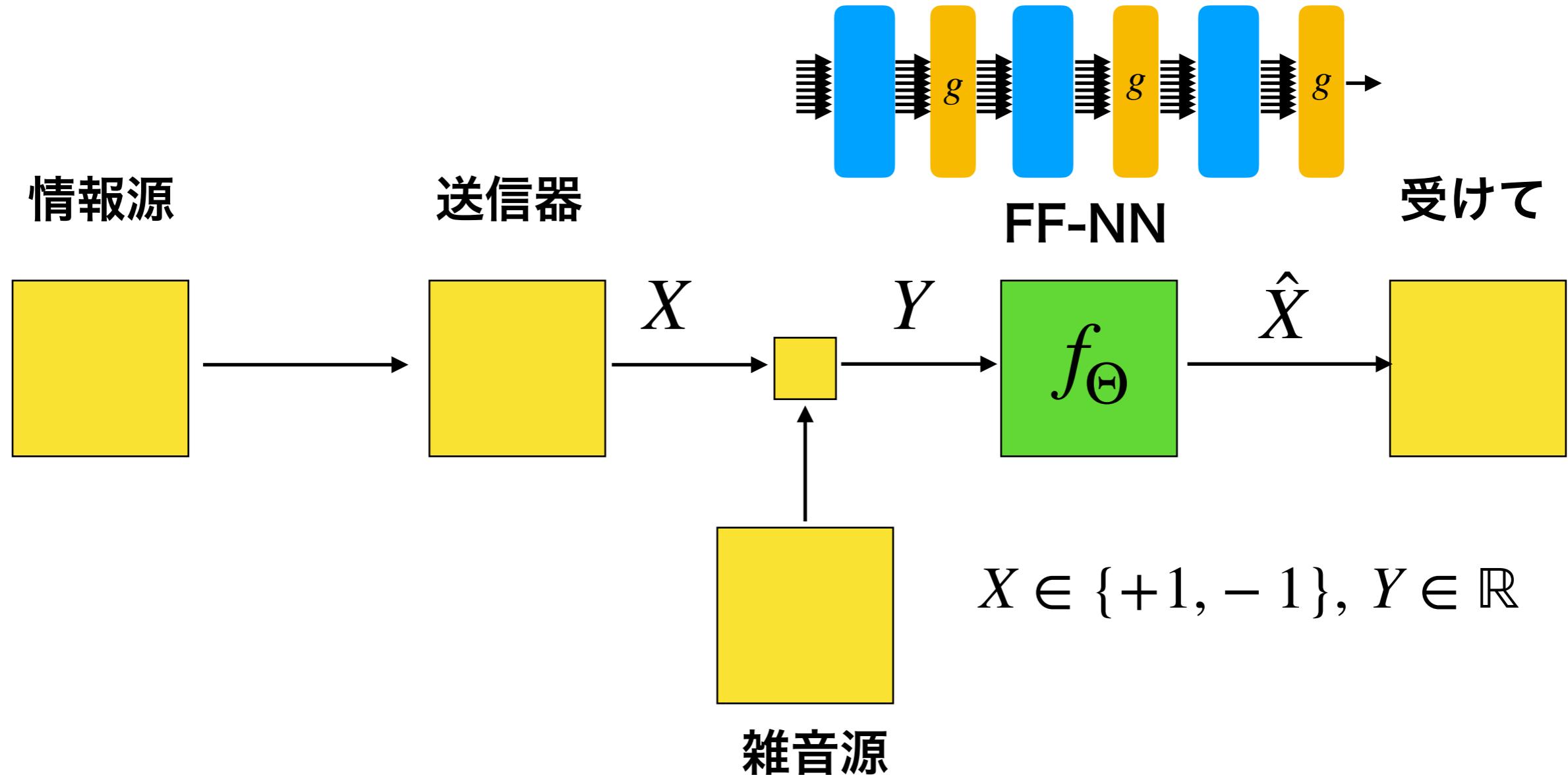
# 本チュートリアルの構成

- 第一部：
  - (a) 研究環境を整える
  - (b) AND関数を学習する
  - (c) MIMO検出器を作る**
  - (d) 自己符号化器による通信路模擬
- 第二部：
  - (a) 深層展開
  - (b) スパース信号復元とISTA(近接勾配法)
  - (c) 「深層展開」に関する関連研究

# 通信関連の深層学習研究の分類

タイプ	アイデア
ブラックボックスモデル	通信路全体・検出アルゴリズム・復号アルゴリズムを深層ニューラルネットワークを利用して表現。データ駆動型設計。事前知識・過去の知見必要なし。
深層展開	既知の反復型型アルゴリズムに学習可能パラメータを埋め込み深層学習技術を利用してパラメータを学習。
最適化模擬	凸最適化・非凸最適化アルゴリズムのニューラルネットワークによる模擬。
分散学習	各端末における学習とその学習結果の統合
強化学習	報酬に基づく最適戦略の決定、マルコフ決定プロセス。

# 深層ネットワークに基づく信号検出器



この形の検出器の論文はかなり数が出ています。その一例：

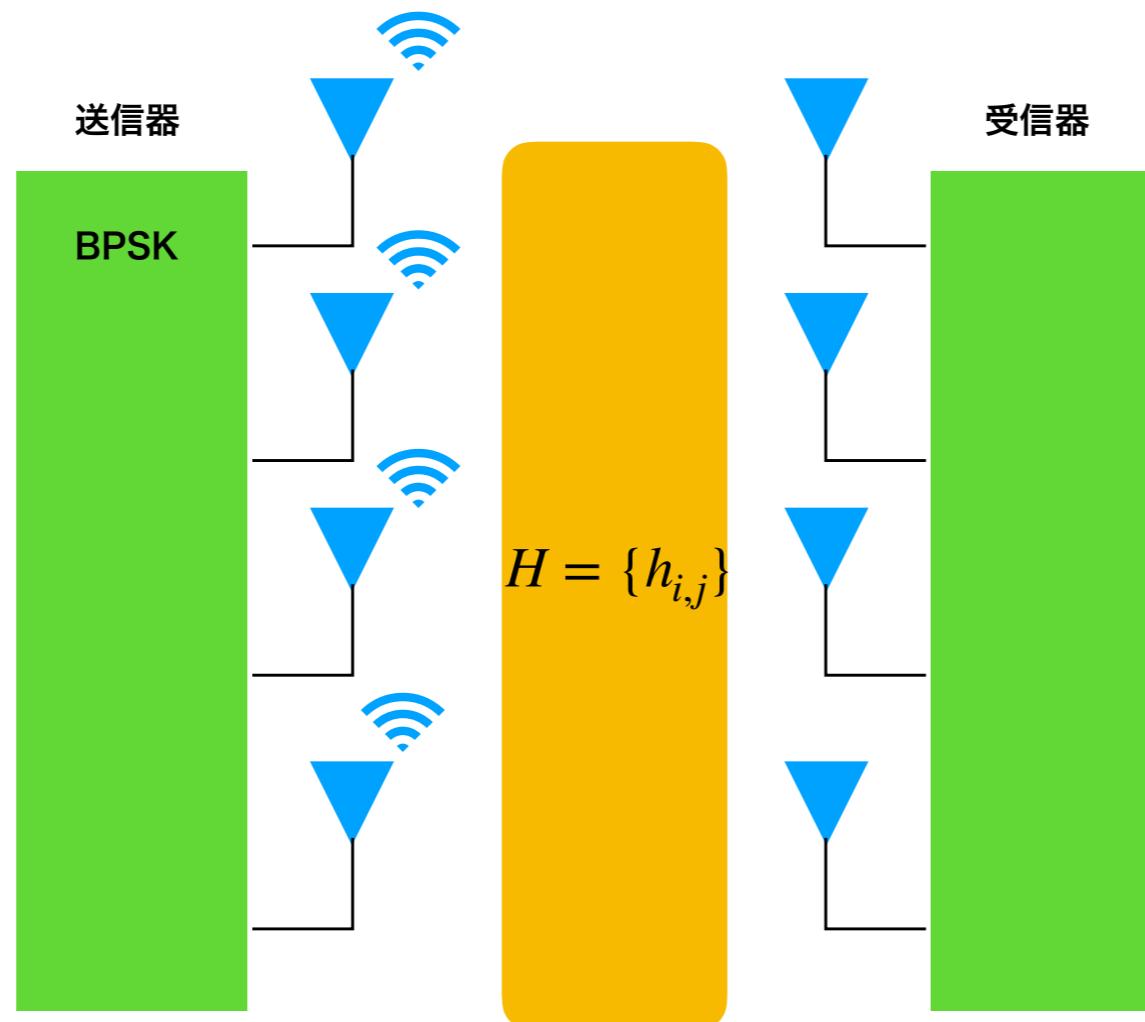
N.Samuel et al. ``Learning to Detect," IEEE Trans. Signal Processing, May, 2019

# MIMO検出問題

送信信号:  $\{+1, -1\}^n$

$$H \in \mathbb{R}^{n \times n}$$

通信路モデル:  $y = Hx + w$  (加法的白色ガウス雑音を仮定)



信号検出問題:  $y$  から  $x$  をなるべく正確に推定

# 最尤推定問題

尤度関数(受信ベクトル $y$ に基づく)

$$P(y|x) = \beta \exp(-\alpha ||y - Hx||^2)$$

最尤推定則(誤り率を最小にする)

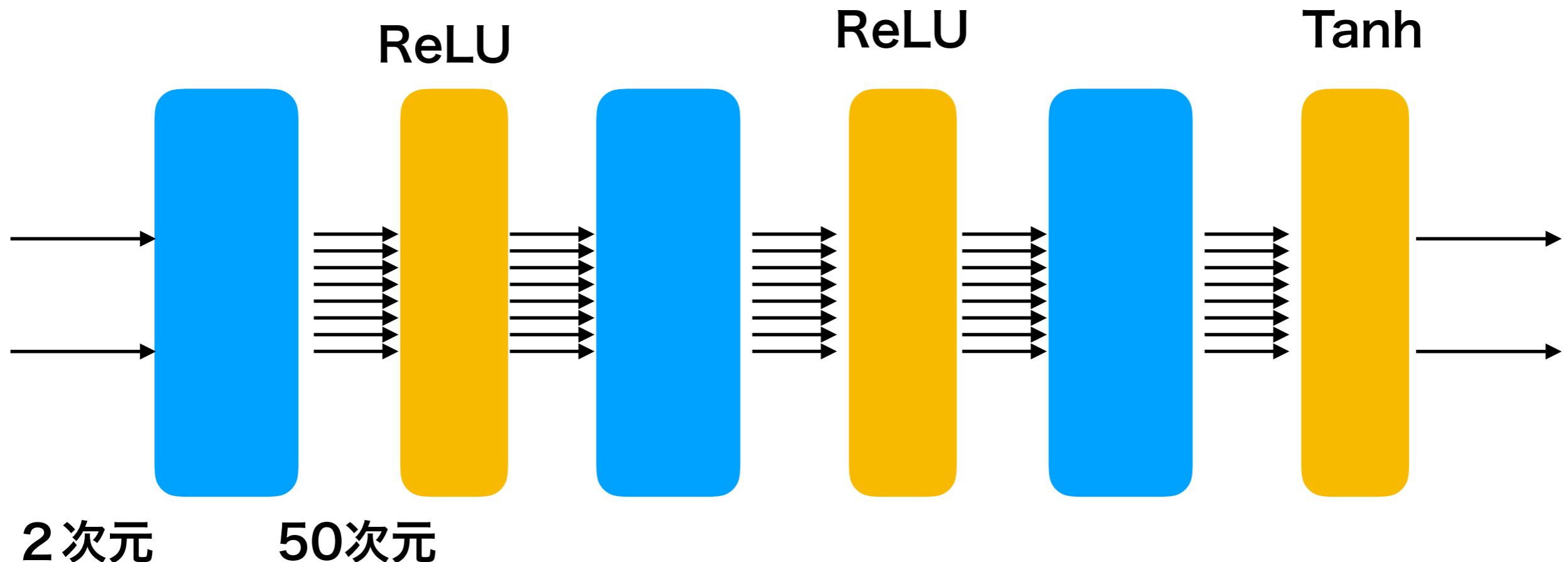
$$\hat{x} = \arg \max_{x \in \{+1, -1\}^n} P(y|x) = \arg \min_{x \in \{+1, -1\}^n} ||y - Hx||^2$$

## リマーク

- 最尤推定(ML)は $n$ が大きいときには計算量的に困難な問題になる
- 逆行列を $y$ に乘じるZF推定は計算量的に楽だが、復元性能はMLに劣る
- MMSE推定もZFよりは良いが、MLには追いつかない
- MLよりも計算量が少なく、ZF/MMSEよりも推定性能がよい検出手法は？

# MIMO信号推定

$n = 4$  のケース(このパラメータだとMLが簡単にできますが。。。)



# MIMO.ipynbのコードを見てみる

## グローバル定数の設定 ¶

```
: mbs = 50 # ミニバッチサイズ  
noise_std = 0.5 # 通信路において重畠される加法的白色ガウス雑音の標準偏差 (sigma)  
n = 4 # アンテナ数  
h = 50 # 隠れ層のユニット数  
H = torch.normal(mean=torch.zeros(n, n), std=1.0) # 干渉行列  
adam_lr = 0.001 # Adamの学習率
```

ガウス乱数の生成

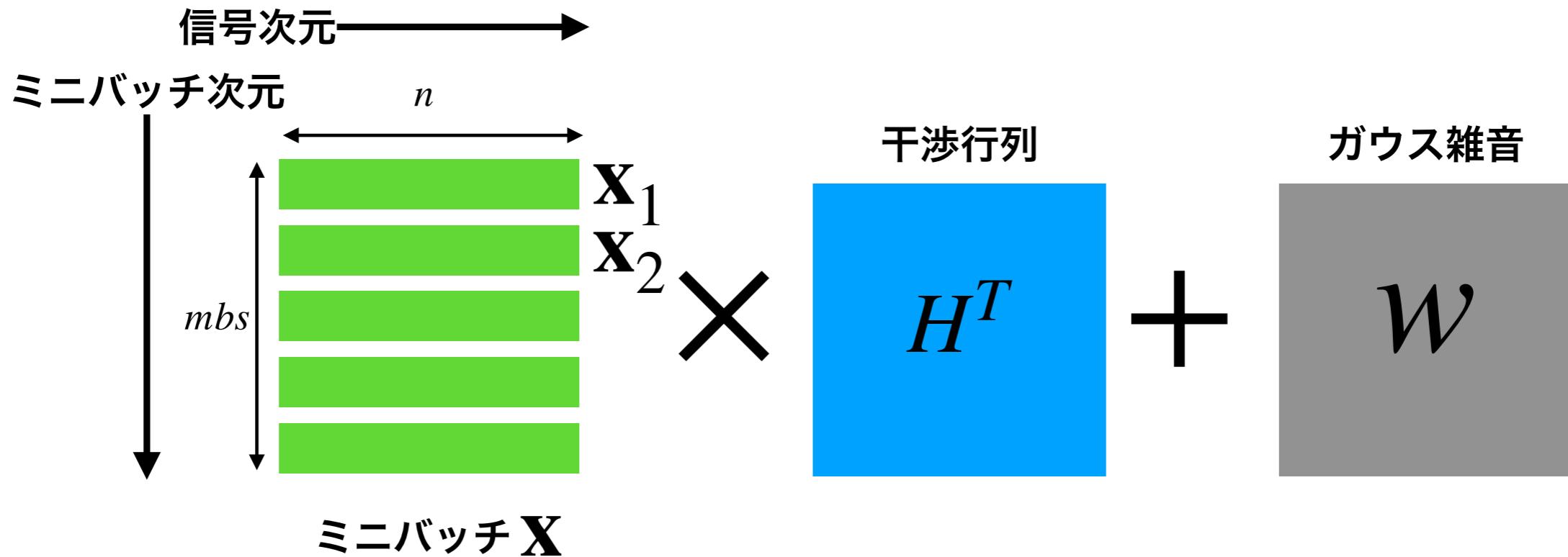
## ミニバッチ生成関数

```
: def gen_minibatch():  
    x = 1.0 - 2.0 * torch.randint(0, 2, (mbs, n)) # 送信ベクトル x をランダムに生成  
    x = x.float()  
    w = torch.normal(mean=torch.zeros(mbs, n), std = noise_std) # 加法的白色ガウス雑音の生成  
    y = x @ H.t() + w # @は行列ベクトルの積  
    return x, y
```



行ベクトル形式

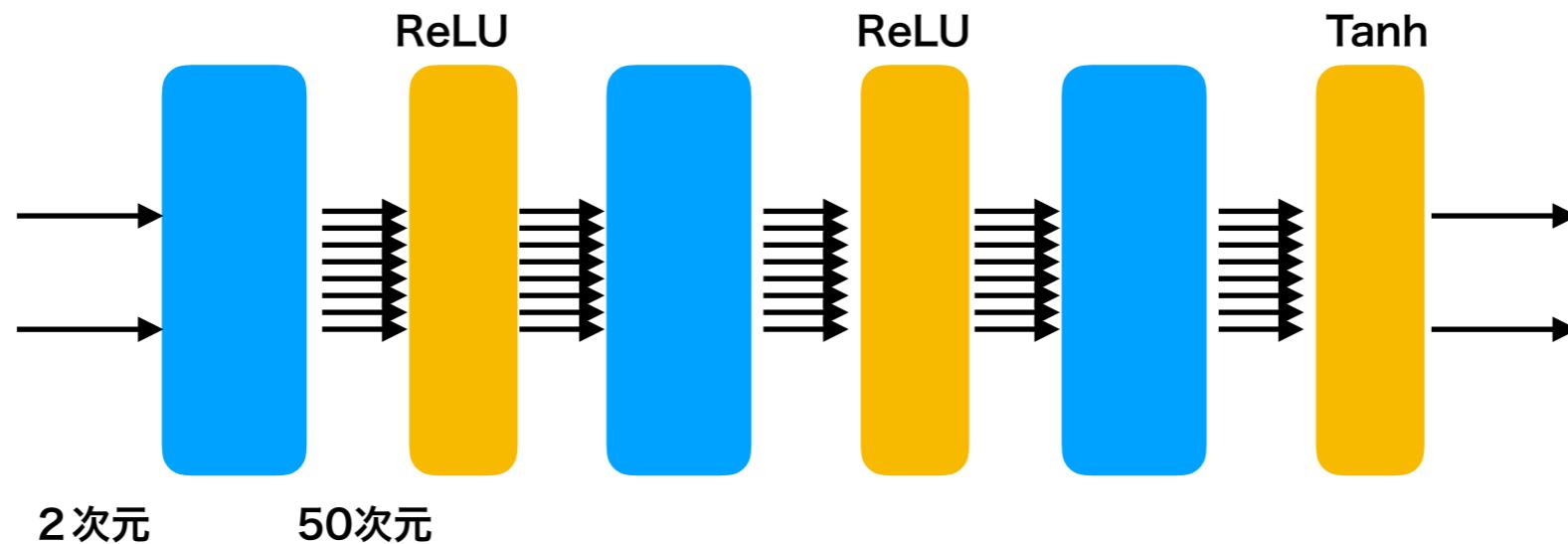
# 次元を確認する



= リマーク  
ミニバッチの第0次元はミニバッチ  
次元として利用されることが慣例

ミニバッチ  $\mathbf{y}$

# ニューラル検出器



## ネットワークの定義

```
] : class Net(nn.Module): # nn.Module を継承
    def __init__(self): # コンストラクタ
        super(Net, self).__init__()
        self.detector = nn.Sequential(
            nn.Linear(n, h), # W_1, b_1,
            nn.ReLU(), # 活性化関数としてReLUを利用
            nn.Linear(h, h), # W_2, b_2
            nn.ReLU(),
            nn.Linear(h, n) # W_3, b_3
        )
    def forward(self, x): # 推論計算をforwardに書く
        x = self.detector(x)
        x = torch.tanh(x) # x \in {+1, -1}^4 なので、最終層はtanhを利用
        return x
```

ユニットを並べます

## 考察

- 通信路に関する事前知識が不要ない→高い柔軟性・設計の容易さ
- ここではAWGN・線形干渉通信路を仮定したが、より複雑な通信路にも対応できる
- ここではランダムデータで訓練を行ったが、実観測データを利用した訓練も可能
- 通信路の変動に対応するには、オンラインで学習する仕組みが必要

# モデルベース 対 ブラックボックス

## モデルベース アプローチ

## ブラックボックス アプローチ

Pros:

ドメイン知識の有効活用  
原理からアルゴリズムの演繹的導出  
(例: 尤度関数最大化 → 最尤推定則 →  
近似 → ZF/MMSE)

Cons:

既存のアルゴリズムをよりパワー  
アップ(性能・柔軟性)するにはどうす  
れば。。。?

ドメイン知識不要  
データ駆動型アルゴリズム設計  
(例: NNモデル作成 → 訓練プロセス)

解釈可能性問題  
スケーラビリティ

# 本チュートリアルの構成

- 第一部：

- (a) 研究環境を整える
- (b) AND関数を学習する
- (c) MIMO検出器を作る

## Q&A

- 第二部：

- (a) 深層展開
- (b) スパース信号復元とISTA(近接勾配法)
- (c) 深層展開に関する関連研究

## Q&A

# 本チュートリアルの構成

- **第一部：**

- (a) 研究環境を整える
- (b) AND関数を学習する
- (c) MIMO検出器を作る

Q&A

- **第二部：**

- (a) 深層展開**

- (b) スパース信号復元とISTA(近接勾配法)
- (c) 「深層展開」に関する関連研究

Q&A

# 通信関連の深層学習研究の分類

タイプ	アイデア
ブラックボックスモデル	通信路全体・検出アルゴリズム・復号アルゴリズムを深層ニューラルネットワークを利用して表現。データ駆動型設計。事前知識・過去の知見必要なし。
深層展開	既知の反復型型アルゴリズムに学習可能パラメータを埋め込み深層学習技術を利用してパラメータを学習。
最適化模擬	凸最適化・非凸最適化アルゴリズムのニューラルネットワークによる模擬。
分散学習	各端末における学習とその学習結果の統合
強化学習	報酬に基づく最適戦略の決定、マルコフ決定プロセス。

## 深層展開

- ✓ 既存のアルゴリズムの処理を時間方向に展開  
(deep 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) が得られる

# 可微分プログラミング



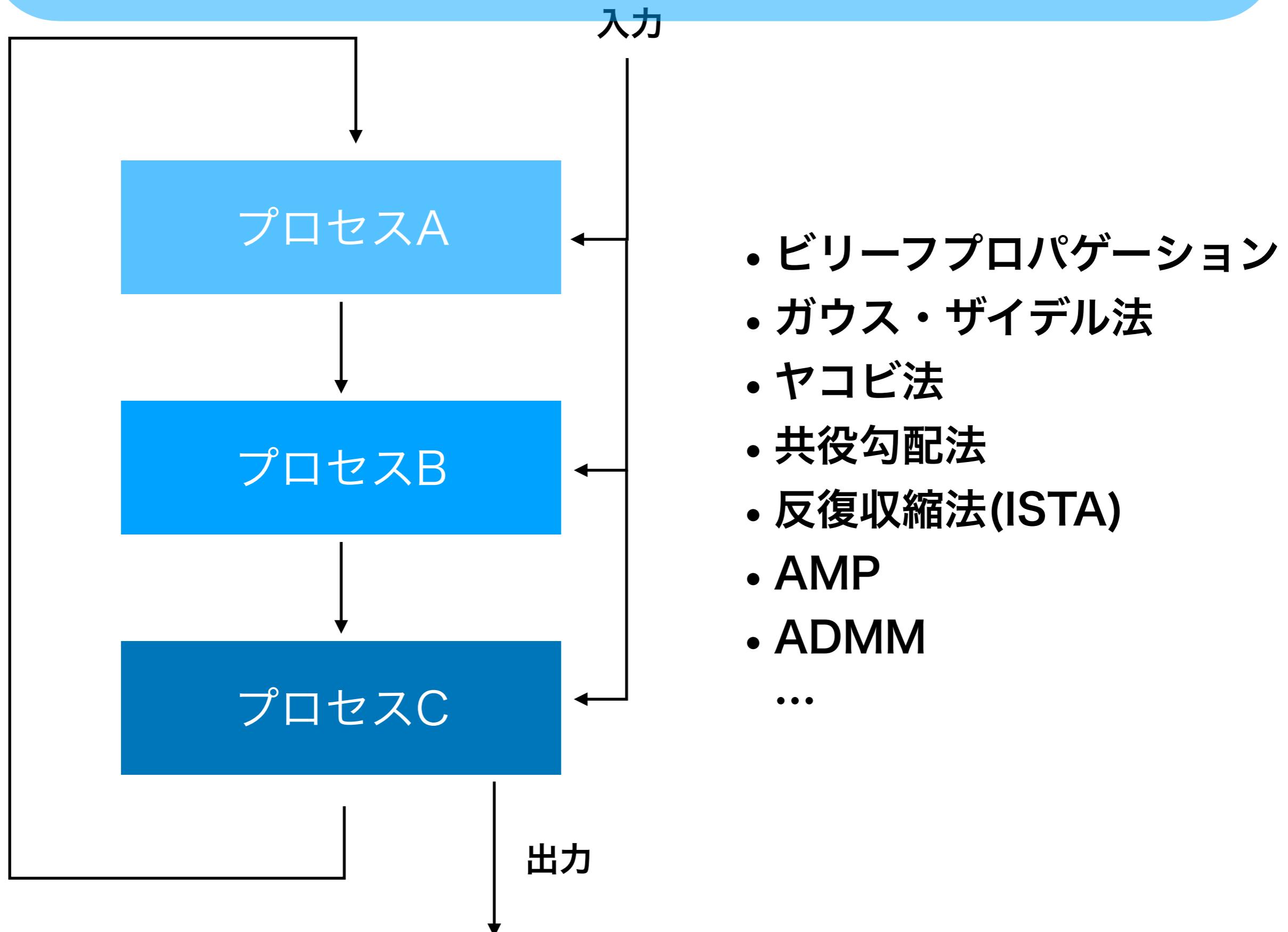
LeCunは、入出力を伴う処理(NNとは限らない)に対して、深層学習技術が適用可能であることを示唆している

… 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…

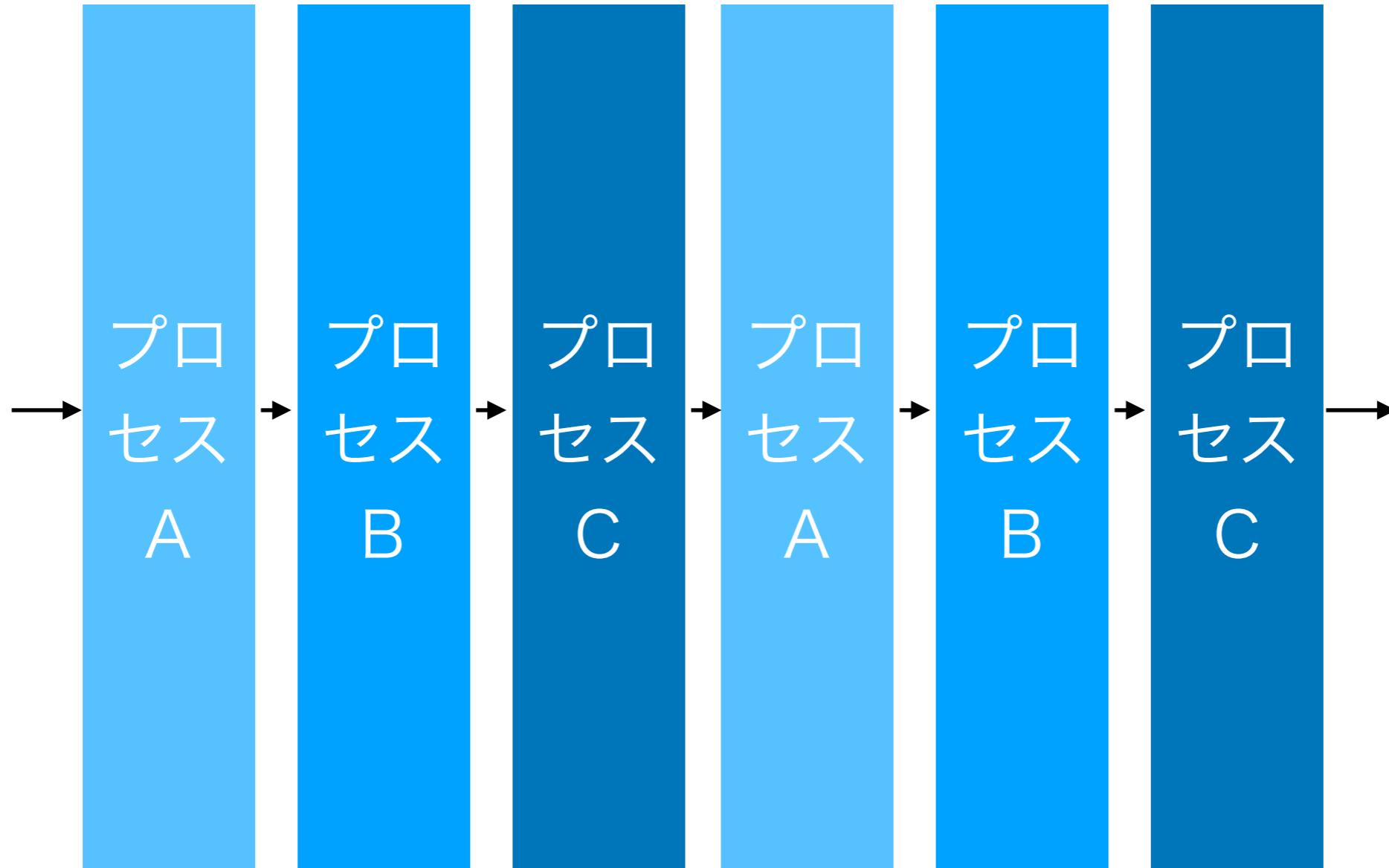


処理全体が微分可能であれば、内部に含まれるパラメータをbackprop+SGDで最適化できる  
→**可微分プログラミング**  
(differentiable programming )

# 反復アルゴリズムの構造

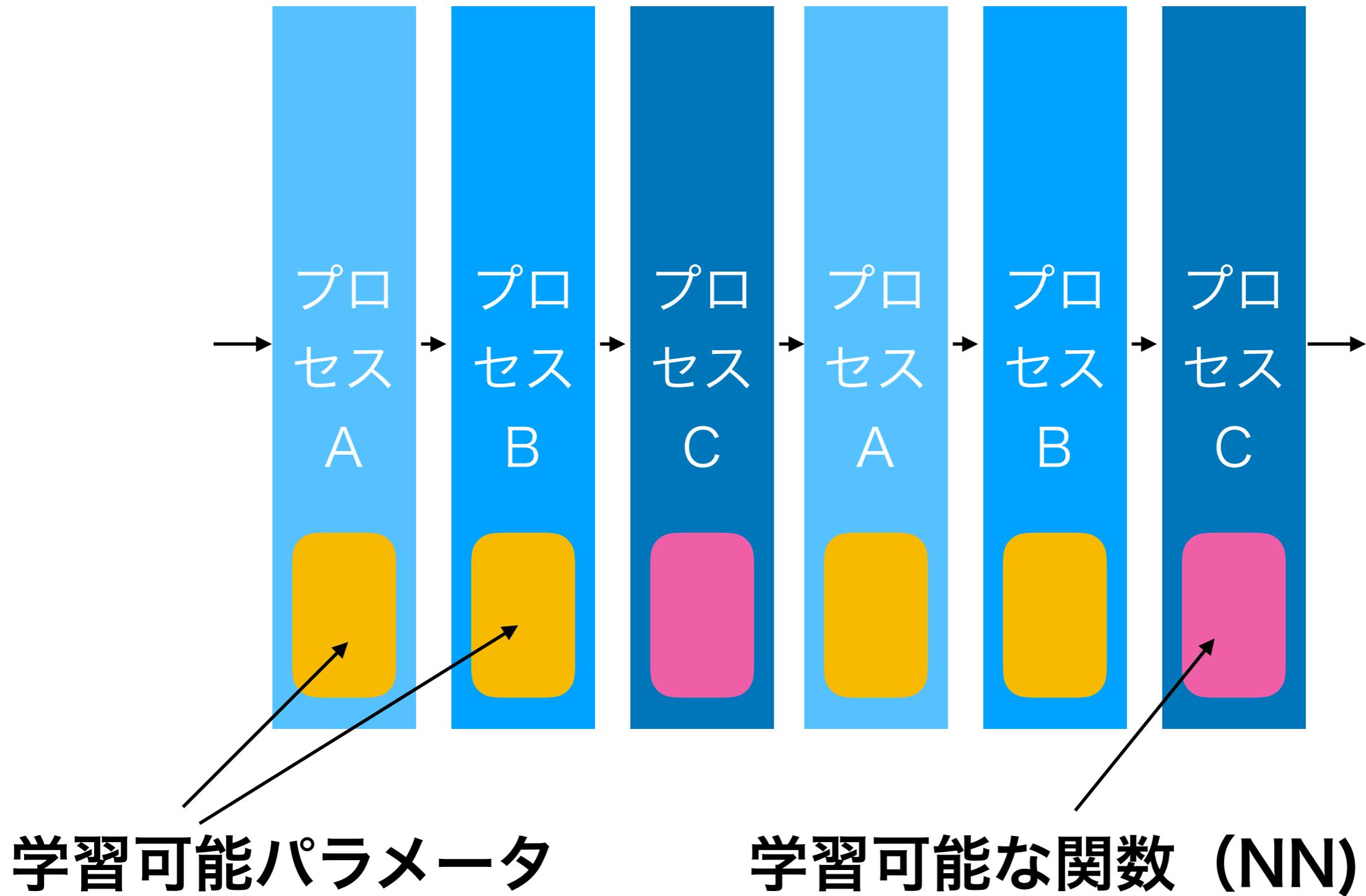


# 信号フローを時間方向に展開

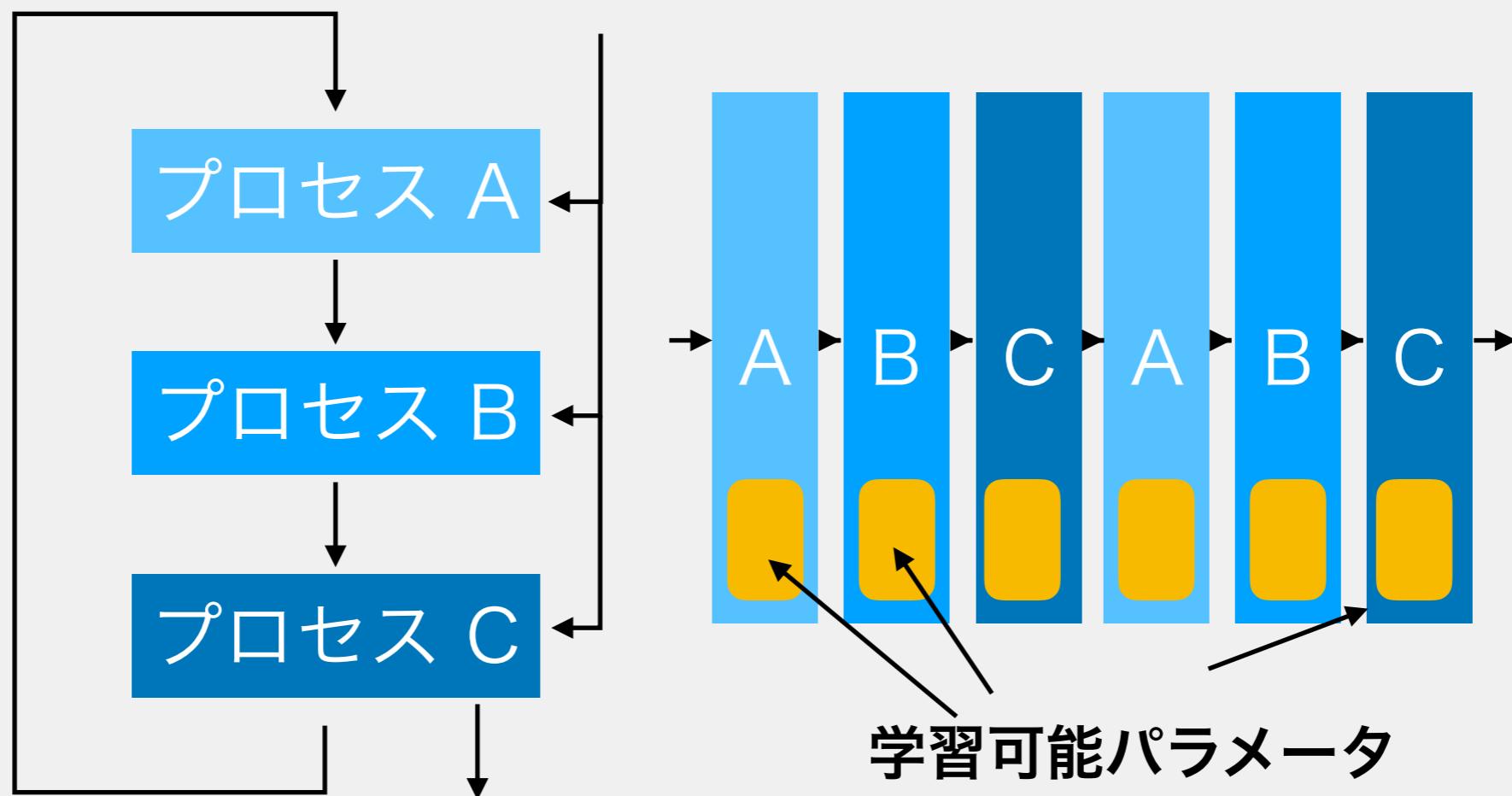


各プロセスが微分可能であり, かつ, その導関数が  
ほぼ至るところでゼロでなければ, **backprop可能**

# 学習パラメータを埋め込む



# 深層展開 (Deep unfolding)



既知の反復型型アルゴリズムに学習可能パラメータを埋め込み深層学習技術を利用してパラメータを学習

## 深層展開のメリット

- 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

## 深層展開の例：2次関数

- ✓ 2次関数最小化問題(凸関数)を例に挙げる
- ✓ 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

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

# TGD クラス (Trainable Gradient Descent)

```
] 1 class TGD(nn.Module):
2     def __init__(self, num_itr):
3         super(TGD, self).__init__()
4         self.beta = nn.Parameter(init_val*torch.ones(num_itr)) #学習可能ステップサイズ
5     def forward(self, num_itr, bs):
6         s = (torch.rand(bs, 2)*20.0 - 10.0) # ランダムな初期探索点
7         for i in range(num_itr):
8             s = s - self.beta[i] * grad_numerical_f(s, bs) ← 勾配法のステップ
9     return s
```

self.betaを学習可能  
パラメータにする

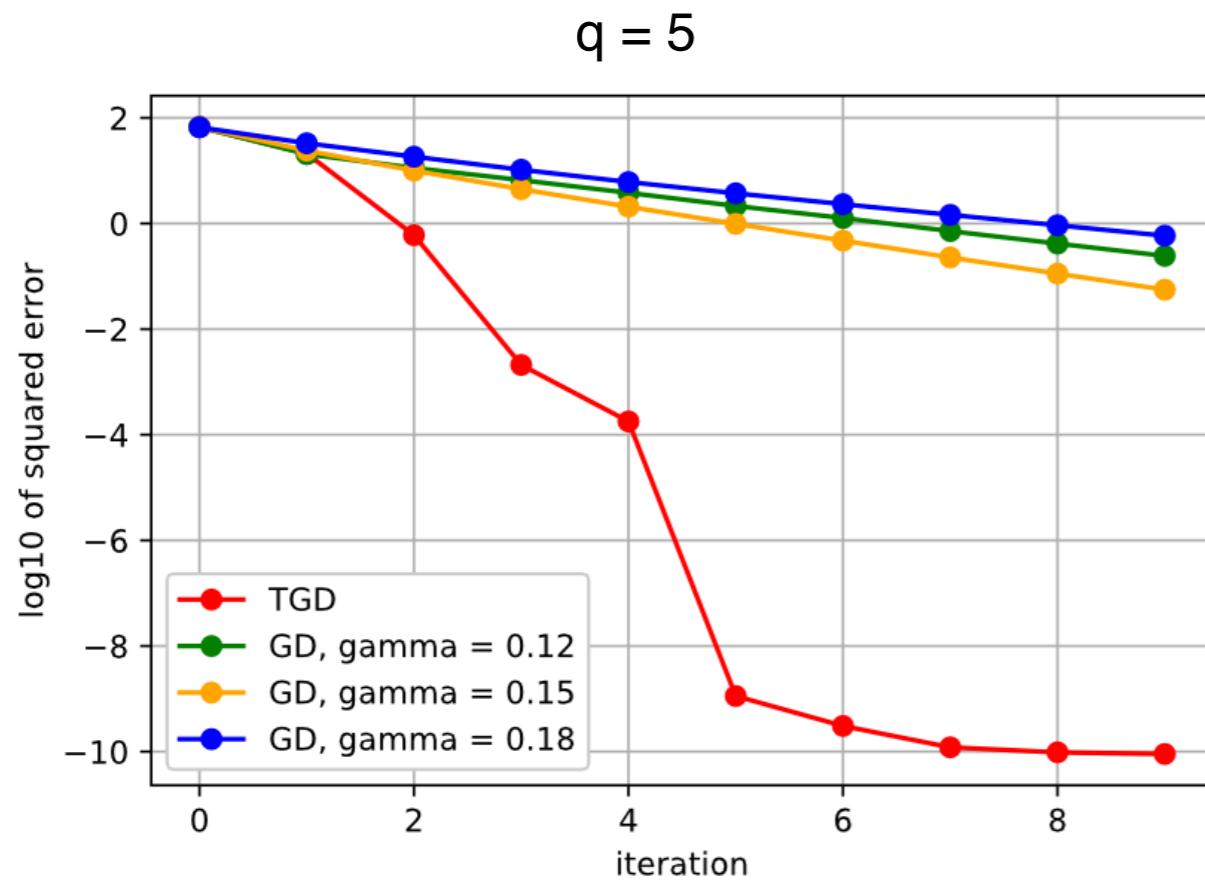
← 勾配法のステップ

## 訓練ループ(インクリメンタルトレーニング)

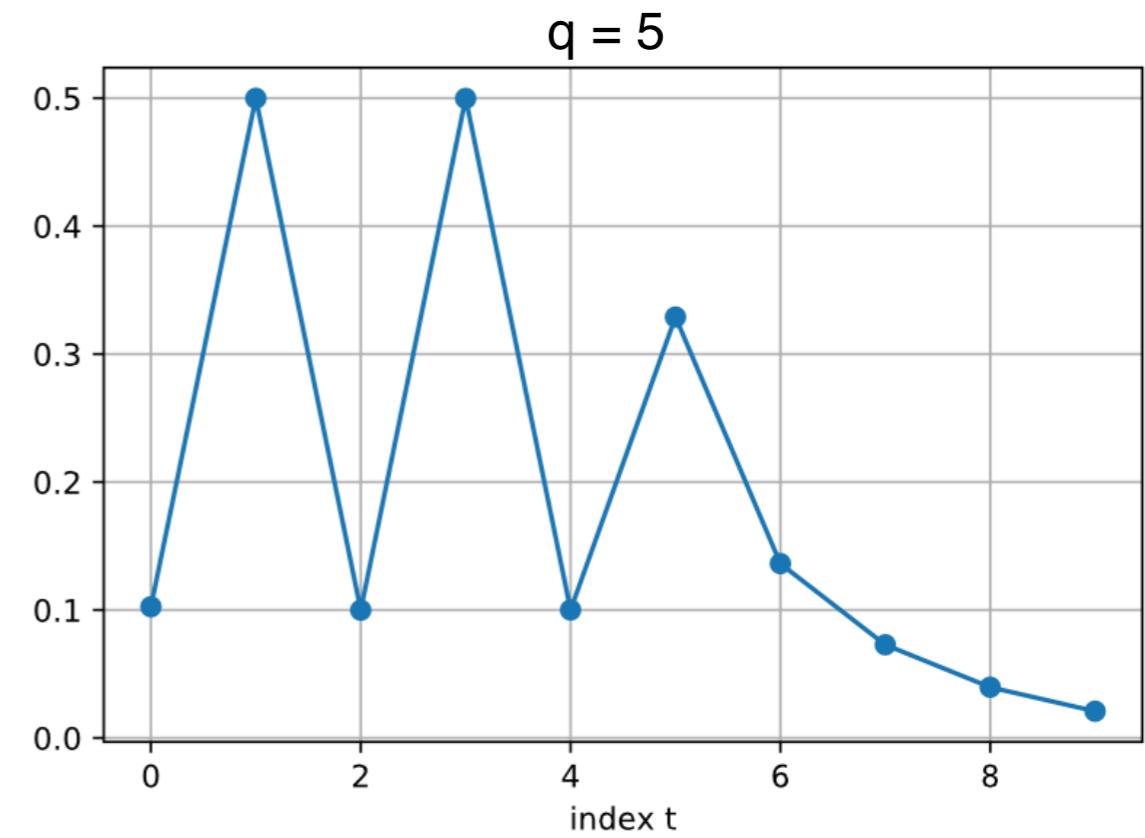
```
] 1 model = TGD(itr)
2 opt   = optim.Adam(model.parameters(), lr=0.001)
3 loss_func = nn.MSELoss()
4 solution = torch.tensor([[0.0, 0.0]]).repeat(bs,1) #解
5 for gen in range(itr):
6     for i in range(1000): ← インクリメンタルトレーニング
7         opt.zero_grad()          (二重ループになっている)
8         x_hat = model(gen + 1, bs)
9         loss  = loss_func(x_hat, solution)
10        loss.backward()
11        opt.step()
12        print(gen, loss.item())
```

# 2次関数最小化における深層展開

真値からの誤差



学習可能パラメータの値

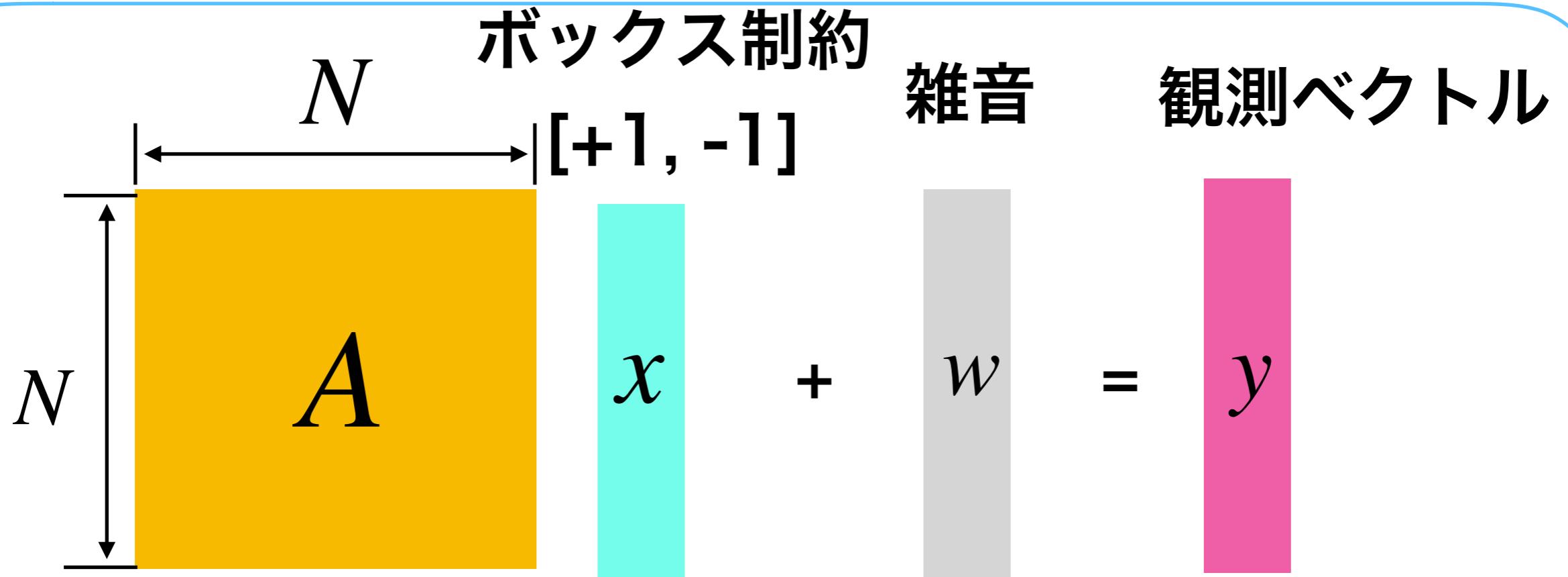


- ✓ ステップサイズ固定よりも速い収束
- ✓ ジグザグ型のパラメータ値

## 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

# 深層展開の例: 射影勾配法



問題： $y$ から  $x$  を再現せよ。

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

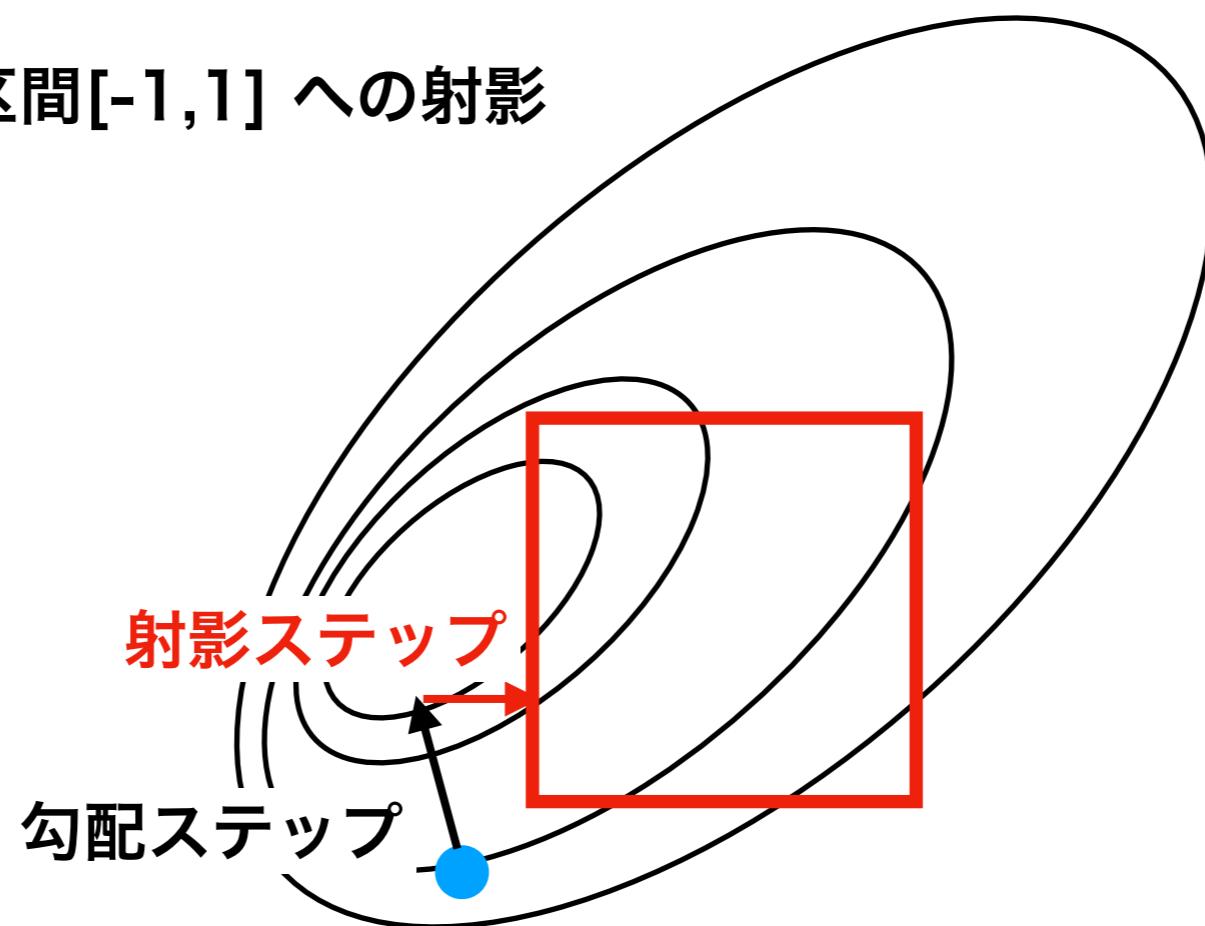
# 深層展開の例: 射影勾配法

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

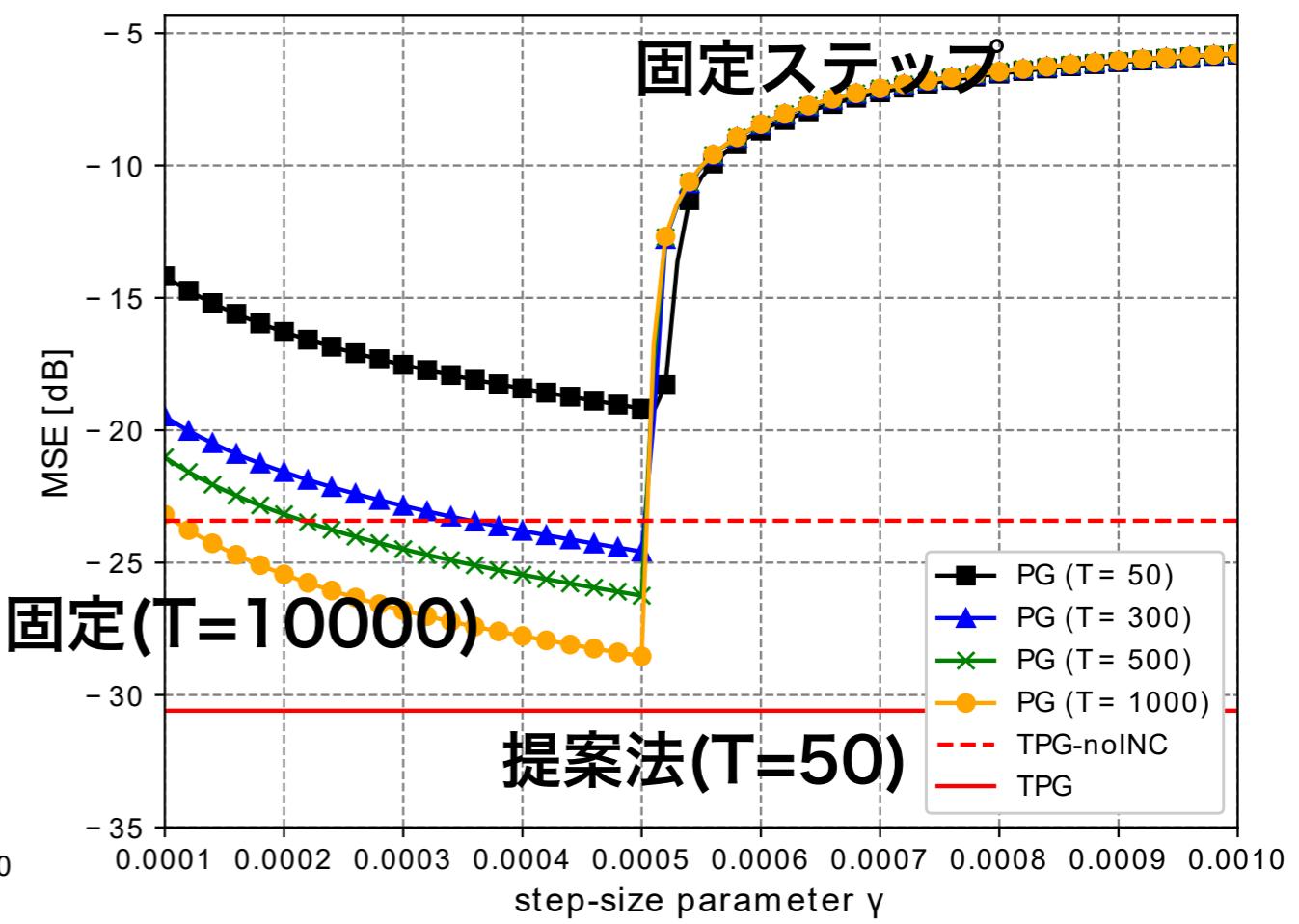
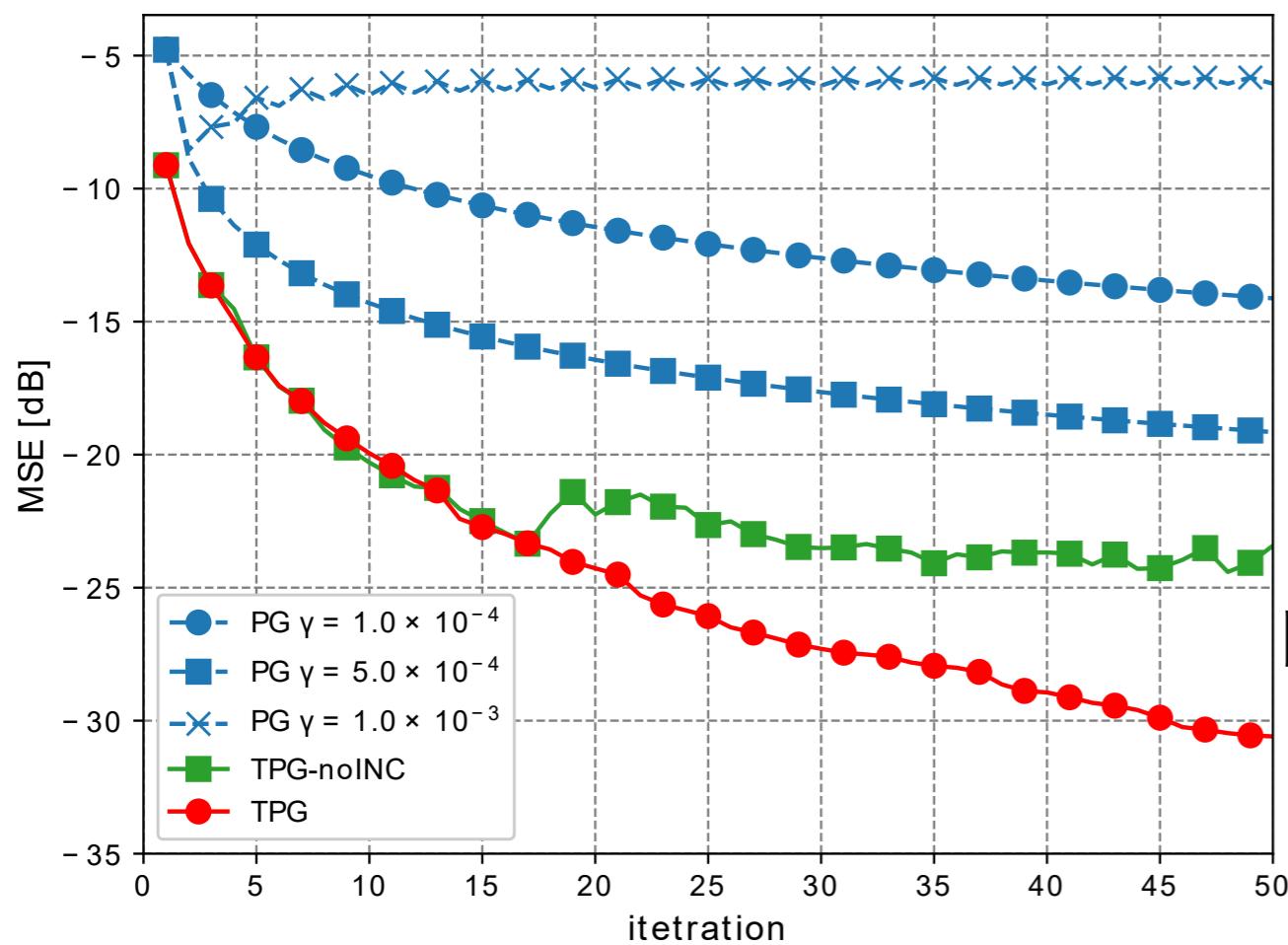
$$r_t = s_t + \underline{\gamma_t} A^T (y - As_t), \quad (\text{勾配ステップ})$$

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

真値からの誤差

$n=1000$

$\gamma$  依存性



# 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

# 本チュートリアルの構成

- 第一部：

- (a) 研究環境を整える
- (b) AND関数を学習する
- (c) MIMO検出器を作る

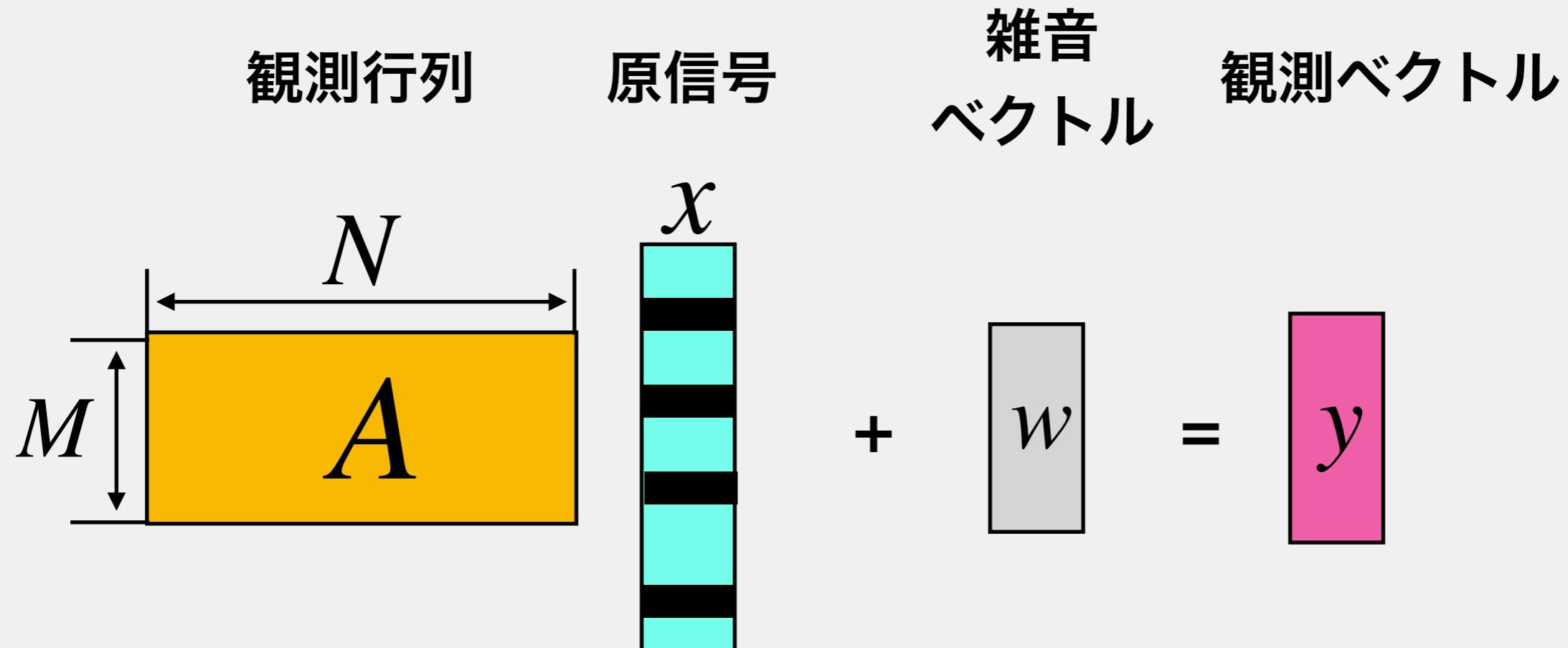
Q&A

- 第二部：

- (a) 深層展開
- (b) スパース信号復元とISTA(近接勾配法)**
- (c) 深層展開に関する関連研究

Q&A

# スペースモデリング(圧縮センシング)



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**

# 無線物理層における劣決定性問題

劣決定系の推定問題の重要性がますます高まりつつある

フルランク系・直交系→劣決定系

- ✓ 通信路推定 (スペースなマルチパス)
- ✓ 到来波方向推定
- ✓ スペクトルセンシング
- ✓ NOMA

劣決定系の推定問題(逆問題)への強力なアプローチ

→正則化

# スパース信号再現問題のLASSO定式化

LASSO (凸計画問題)

$$\hat{x} = \arg \min_{x \in \mathbb{R}^N} \frac{1}{2} \|y - Ax\|_2^2 + \lambda \|x\|_1$$

「近さ」を表す

原信号の事前情報に基づく

二次項

- (1)  $y$  と  $Ax$  の誤差の小さい  $x$  を選好
- (2) 微分可能

L1-正規化項

- (1) 解の唯一性をもたらす
- (2) スパースベクトルを選好
- (3) 微分不可能 (原点にて)

微分不能な正規化項が入っているため、単純な勾配法はうまく動かない

- 線形計画問題として再定式化 → LPソルバで解く
- 近接勾配法の利用

# 近接(proximal)演算子

## 近接演算子

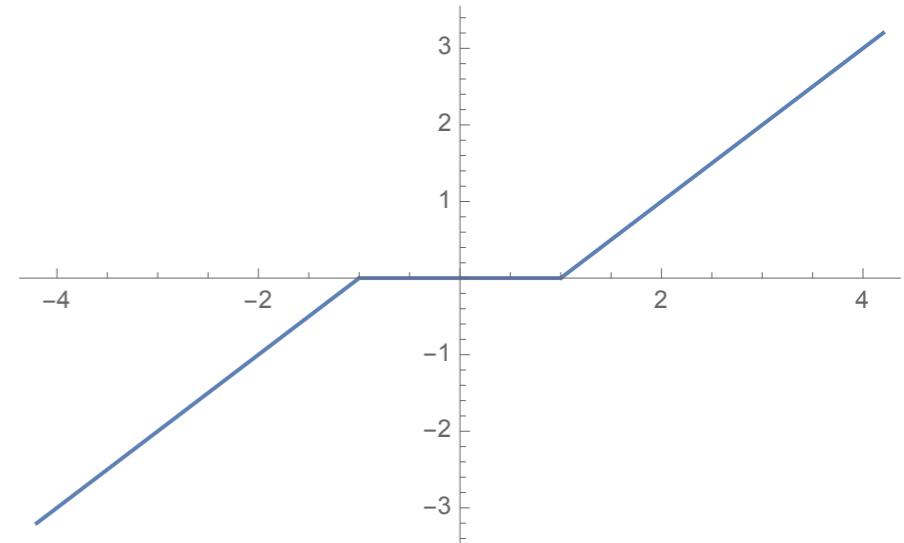
$$f: \mathbb{R}^n \rightarrow \mathbb{R}$$

$$\text{Prox}_{\gamma f}(x) := \arg \min_{u \in \mathbb{R}^n} f(u) + \frac{1}{2\gamma} \|x - u\|_2^2$$

- (1) 射影演算子に似ている
- (2) 微分不能関数を滑らかに

## 例: L1 正規化項に対応する近接演算子

$$f(x) := |x| \quad (f: \mathbb{R} \rightarrow \mathbb{R})$$



$$\text{Prox}_{\gamma f}(x) := \text{sign}(x) \max\{|x| - \gamma, 0\} =: \eta(x; \gamma)$$

ソフトしきい値関数

# 近接勾配法(proximal gradient method)

ゴール

$$\text{minimize } f(x) + g(x) \text{ s.t. } x \in \mathbb{R}^n$$

$f(x)$  : 微分可能

$g(x)$  : 簡潔な近接演算子を持つ

近接勾配法 (proximal gradient method)

$$x^{n+1} := \text{prox}_{\gamma g}(x^n - \gamma \nabla f(x^n))$$

いくつかの条件が満足されるならば、近接勾配法において探索点は最小点に収束する

# ISTA: LASSOに対する近接勾配法

LASSO

$$\hat{x} = \arg \min_{x \in \mathbb{R}^N} \frac{1}{2} \|y - Ax\|_2^2 + \lambda \|x\|_1$$

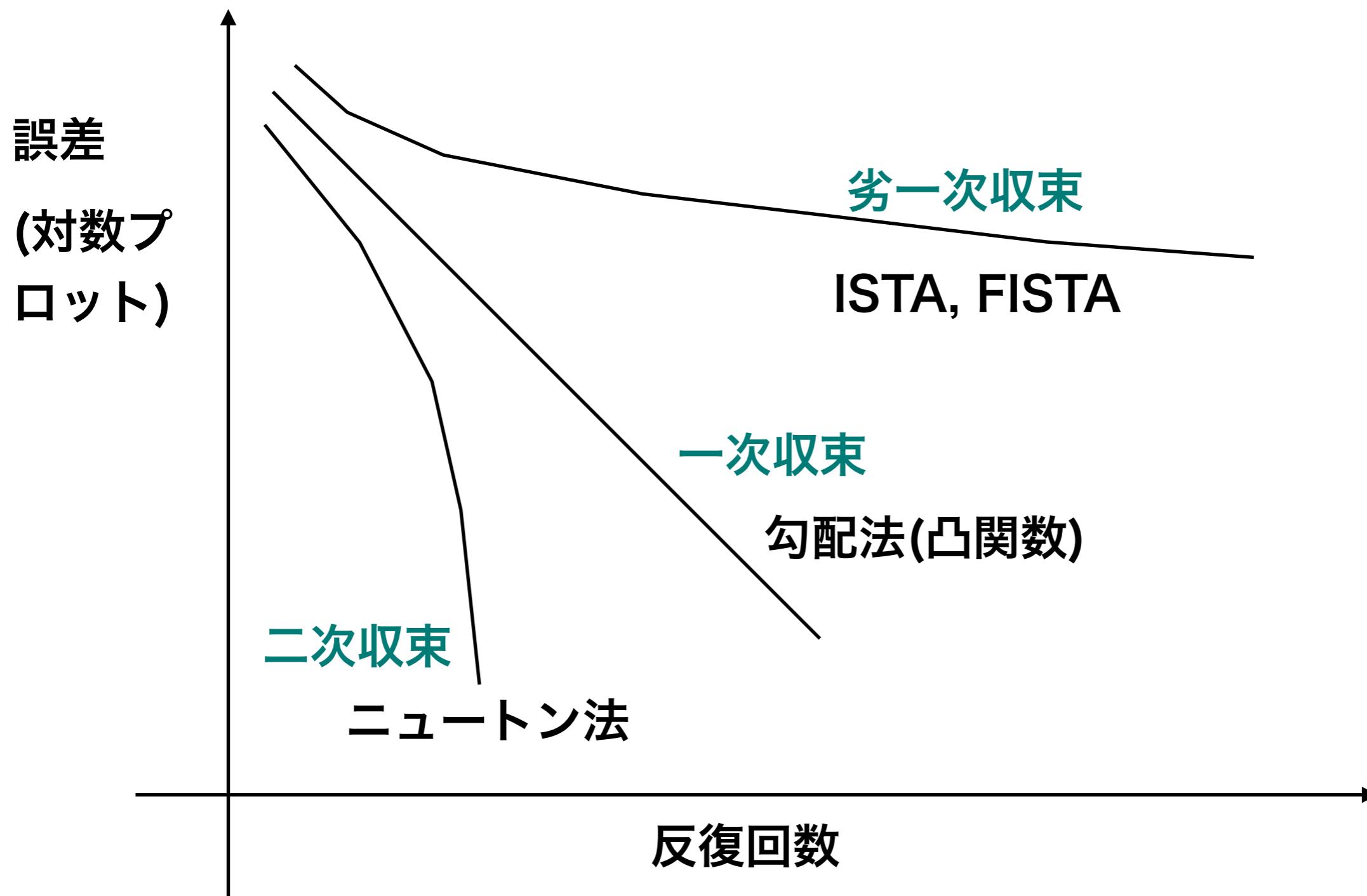
ISTA (Daubechies et.al, 2004)

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

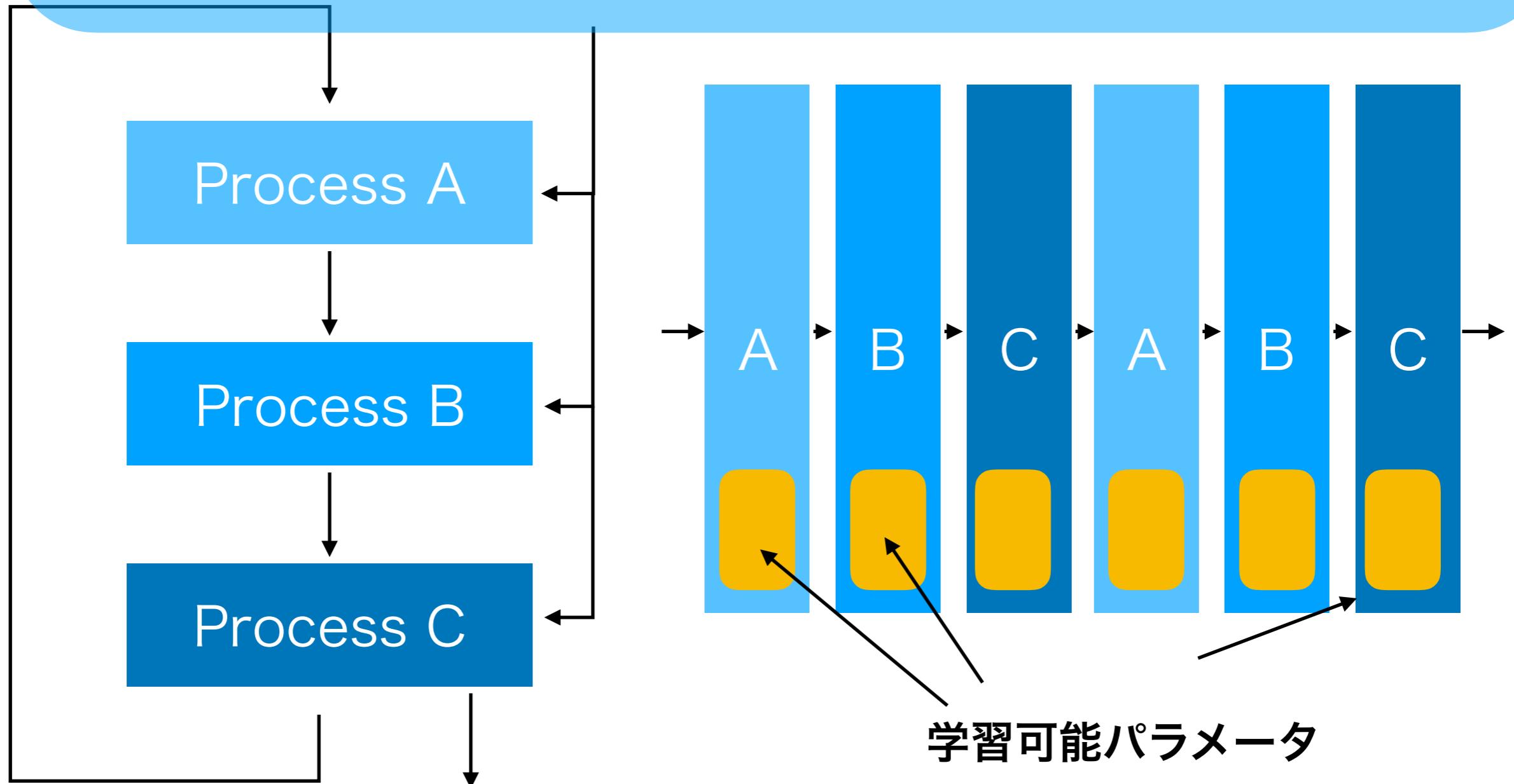
$$s_{t+1} := \eta(r_t; \tau) \quad : \text{近接写像ステップ}$$

- (1) LASSO解に収束する
- (2) 収束が遅い(劣一次収束)
- (3)  $\beta$ と $\tau$ は $A^T A$  の最大固有値に基づき設定

# 反復アルゴリズムの収束速度



# 「深層展開」の誕生



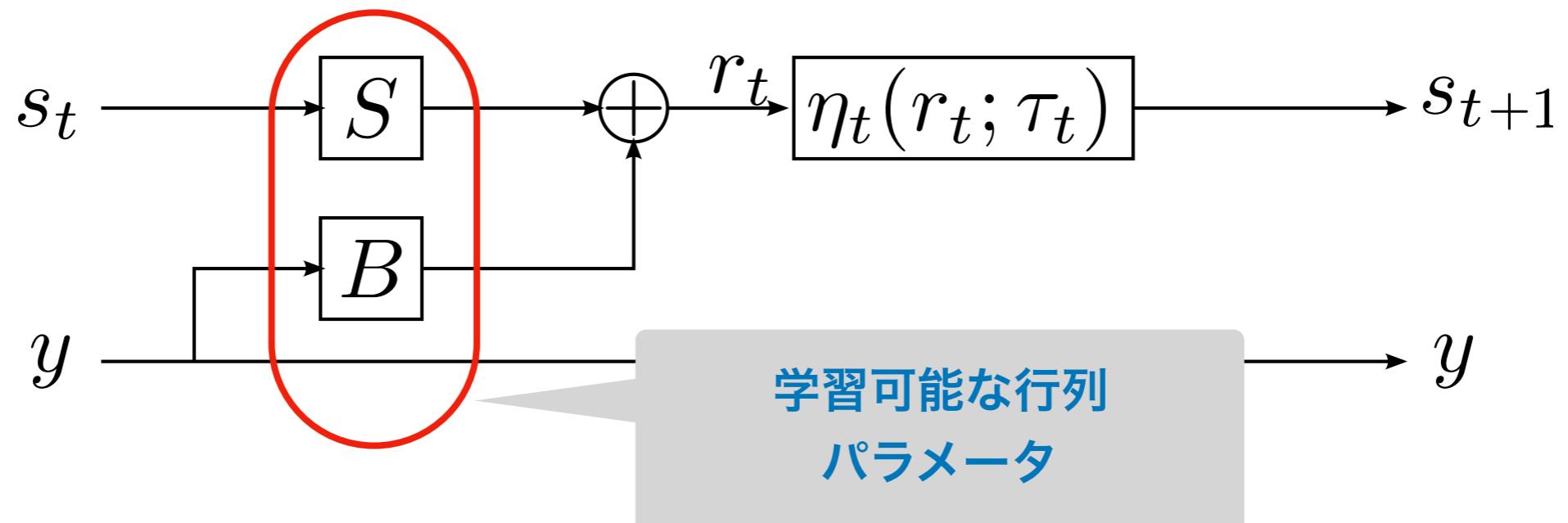
K. Gregor, and Y. LeCun,  
``Learning fast approximations of sparse coding,"  
Proc. 27th Int. Conf. Machine Learning, pp. 399--406, 2010.

# LISTAの構成

$$\text{Original ISTA: } r_t = S_t + \beta A^T (y - As_t)$$
$$S_{t+1} := \eta(r_t; \tau)$$

$$r_t = Bs_t + Sy$$

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



# ISTA.ipynbのネットワーク定義部

## 学習可能なISTA クラス

- アルゴリズムの詳細の説明は5章にて。
- 2種類の学習可能パラメータが導入されている
- 縮小関数として、ソフトしきい値関数を利用

```
: class ISTA(nn.Module):  
    def __init__(self, max_itr):  
        super(ISTA, self).__init__()  
        self.beta = nn.Parameter(0.001*torch.ones(max_itr)) # 学習可能ステップサイズ  
        self.lam = nn.Parameter(0.1*torch.ones(max_itr)) # 学習可能縮小パラメータ  
    def shrinkage(self, x, lam): # 縮小関数 (ソフトしきい値関数)  
        return (x-lam)*(x-lam > 0).float() + (x + lam)*(x+lam < 0).float()  
    def forward(self, num_itr):  
        s = torch.zeros(mbs, n) # 初期探索点  
        for i in range(num_itr):  
            r = s + self.beta[i] * (y - s @ A.t()) @ A # @は普通の行列・ベクトル積  
            s = self.shrinkage(r, self.lam[i])  
    return s
```

$$r_t = s_t + \beta A^T(y - As_t)$$

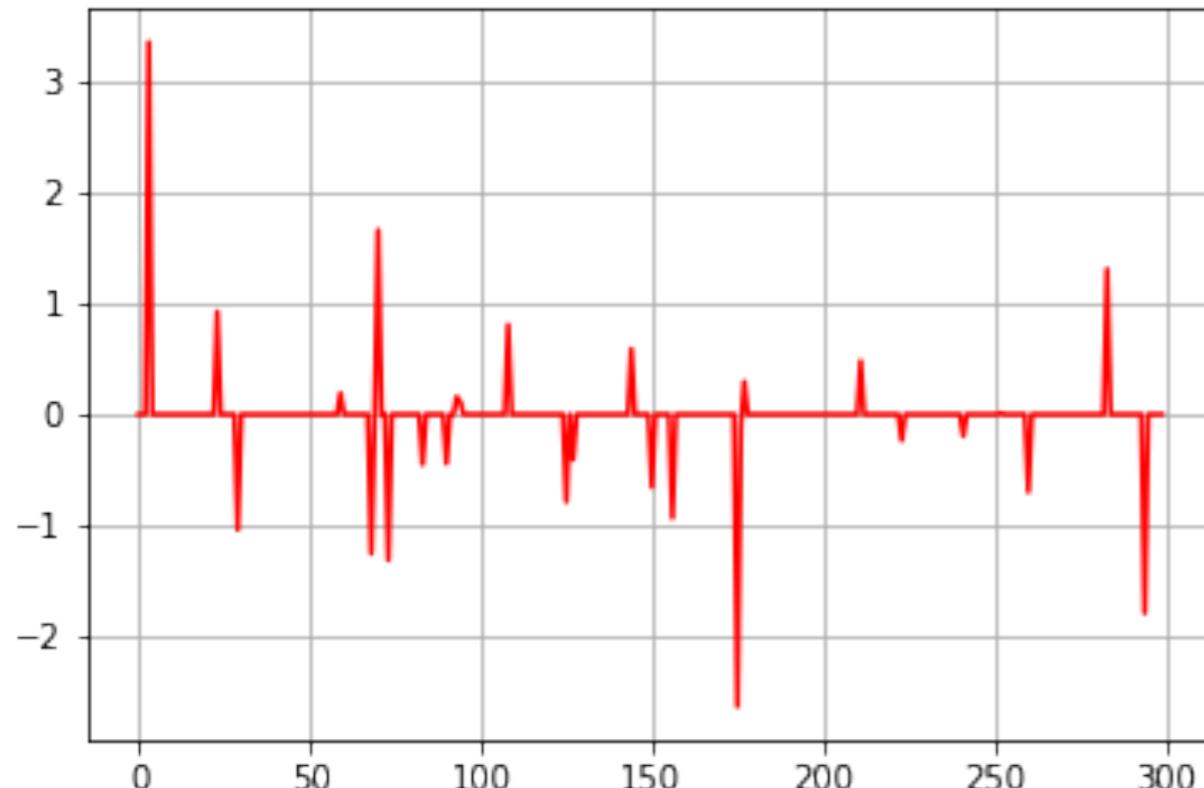
ISTA

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

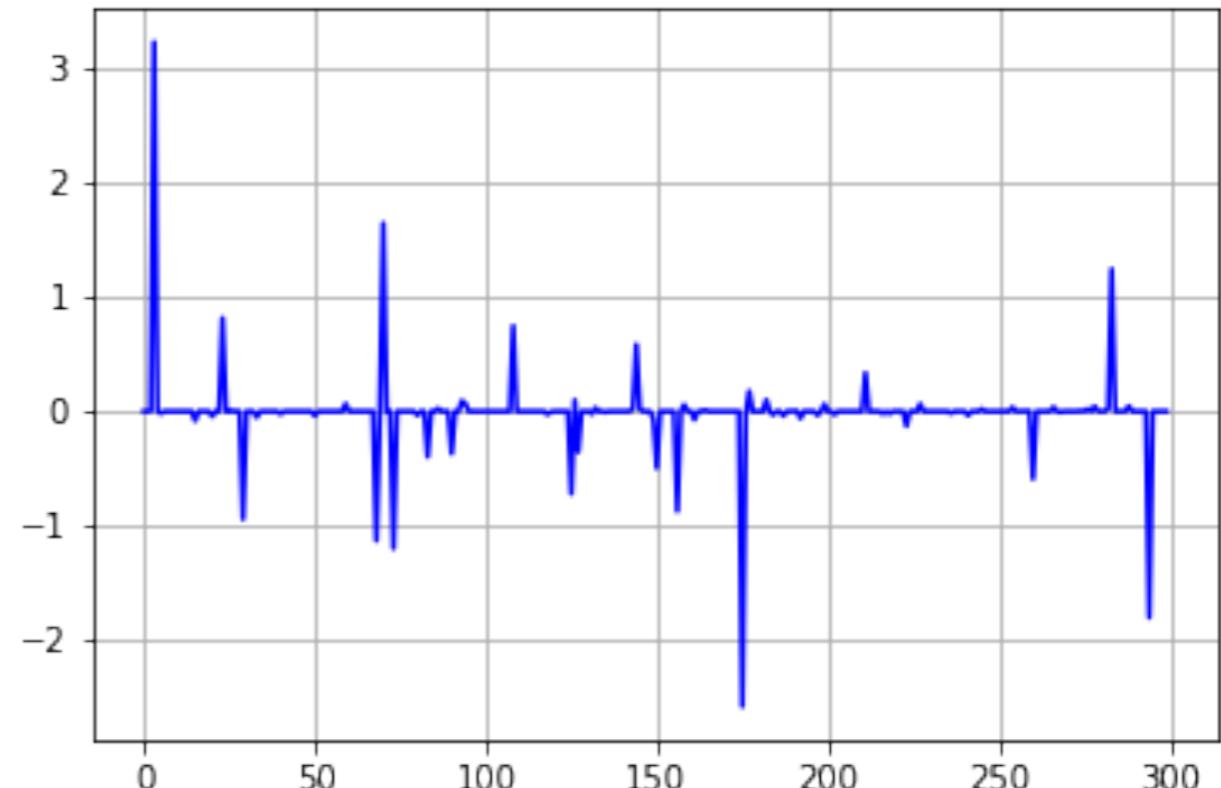
# スパース信号再現の実例

$N = 300, M = 150, p = 0.1$

元信号(スパースベクトル)



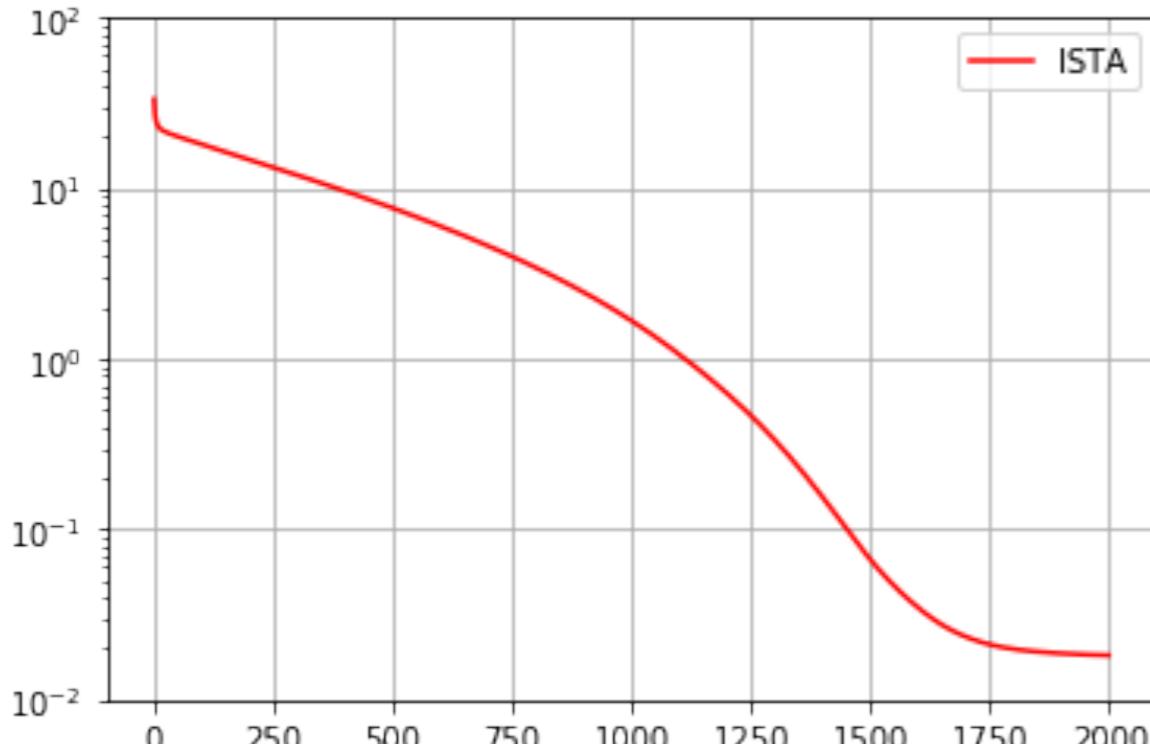
再現ベクトル(学習型ISTA)



# 二乗誤差の収束の様子

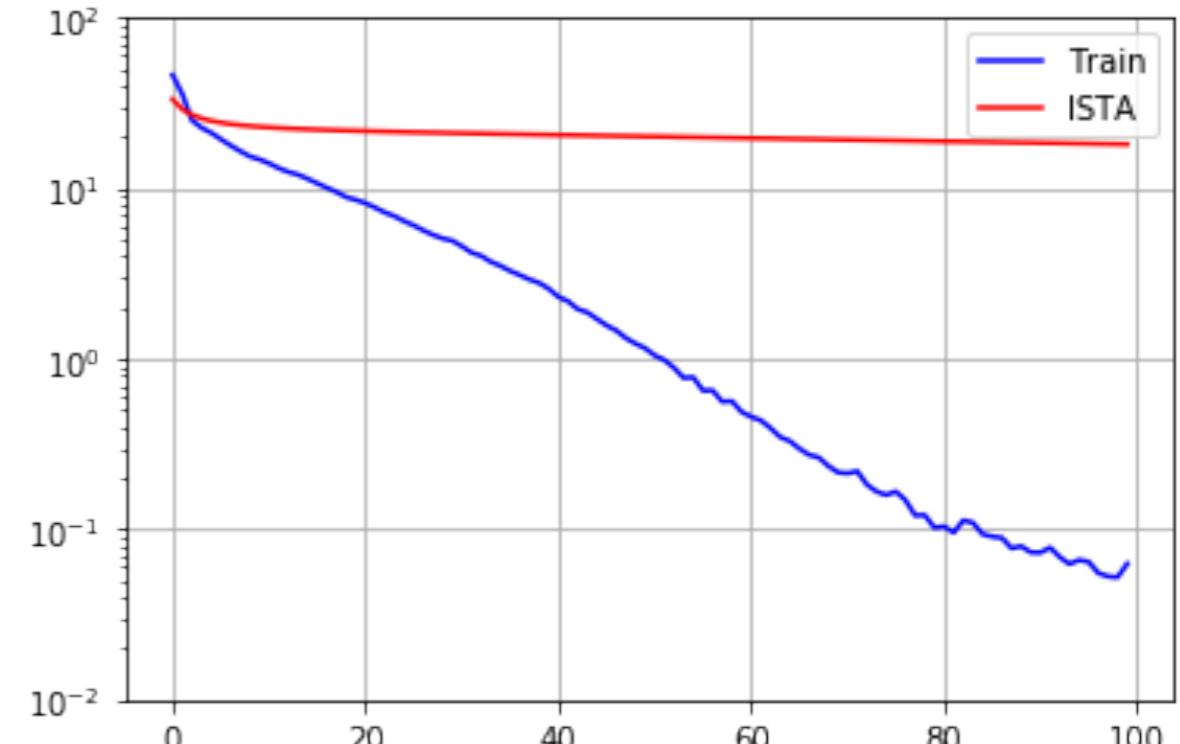
$N = 512, M = 256, p = 0.1$

オリジナルのISTA



反復回数(最大2000)

学習型ISTA

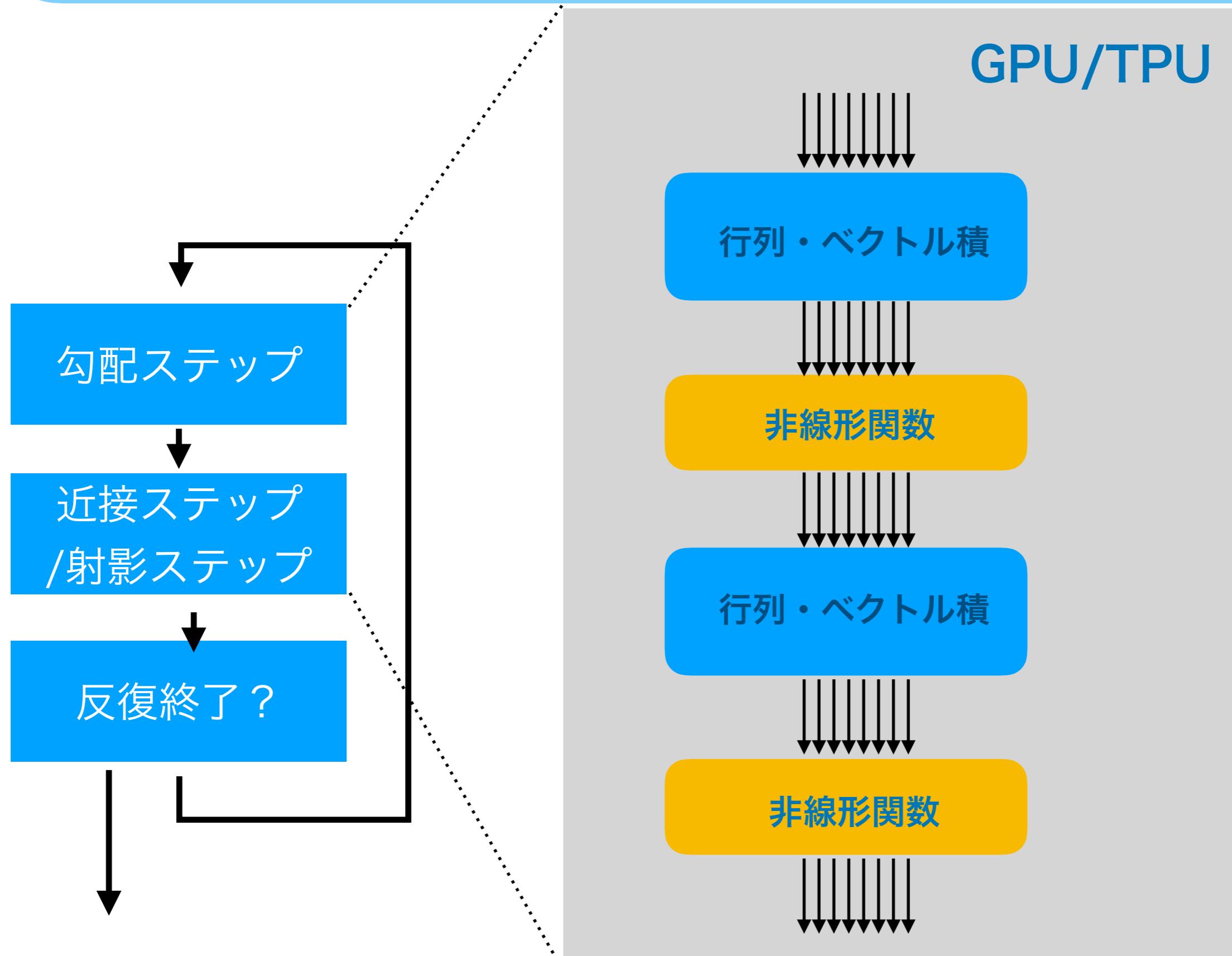


反復回数(最大100)

## 観察

- ISTA (近接勾配法)はやはりそのままでは収束が遅い(劣一次収束) → 高速性が要求される通信信号処理ではそれほど興味がもたれて来なかつたのでは?
- 学習により収束速度が大幅に向上 → おそらく一次収束
- 正則化項は事前情報に基づいて決めればよい。この部分を学習することもできる
- 劣決定性問題に対して自然に近接勾配法が定義できる → 学習型近接勾配法(ステップサイズ、正則化パラメータ、近接写像の形状)への期待

# 近接勾配法と深層ネットワークはとても似ている



# 本チュートリアルの構成

- 第一部：

- (a) 研究環境を整える
- (b) AND関数を学習する
- (c) MIMO検出器を作る

Q&A

- 第二部：

- (a) 深層展開
- (b) スパース信号復元とISTA(近接勾配法)
- (c) **深層展開に関する関連研究**

Q&A

# 深層展開に関する関連研究

## 関連論文

Trainable ISTA (TISTA), Ito-Takabe-Wadayama (スパース信号再現)

BP復号器に適したニューラル量子化器,  
Wadayama-Takabe

学習可能射影勾配法に基づく過負荷MIMO検出,  
アルゴリズム, Takabe-Imanishi-Wadayama

学習可能射影勾配型復号法 (LDPC符号)  
Wadayama-Takabe

複素TISTA(C-TISTA), Takabe-Wadayama  
非線形観測再現

## 発表先

IEEE ICC Workshop 2018

IEEE Trans. on Signal Processing, 2019

IEEE Globecom 2018

IEEE ICC 2019/ IEEE Access, 2019

IEEE ISIT 2019

A recent survey: A. Balatsoukas-Stimming and C. Studer,  
arXiv:1906.05774, 2019.

# TISTA(Trainable ISTA)

D. Ito, S. Takabe, and T. Wadayama, ``Trainable ISTA for sparse signal recovery," IEEE Trans. Signal Processing, vol. 67, no. 12, pp. 3113-3125, Jun., 2019.

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

W は A のムーア・ペンローズ擬似逆行列

# TISTA(Trainable ISTA)

訓練可能パラメータ

$$\begin{aligned} r_t &= s_t + \boxed{\gamma_t} W(y - As_t), \\ s_{t+1} &= \eta_{MMSE}(r_t; \tau_t^2), \end{aligned}$$

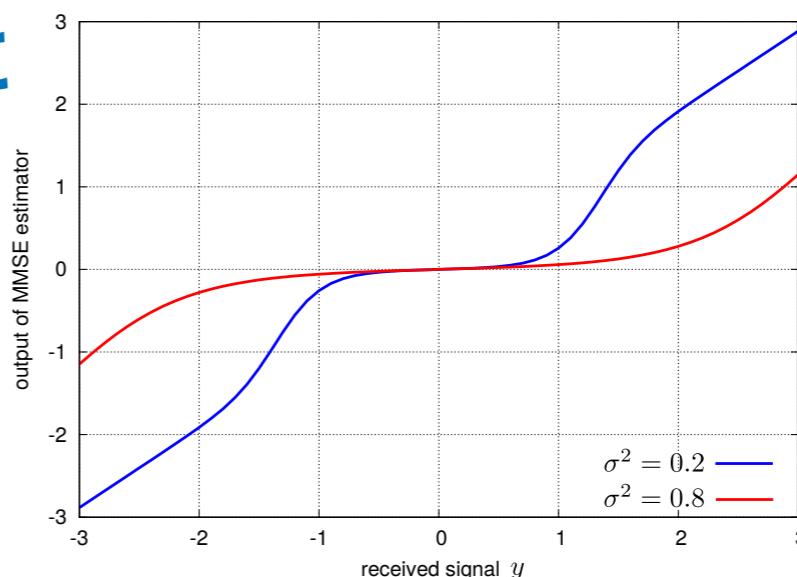
近接勾配法

$$\nu_t^2 = \max \left\{ \frac{\|y - As_t\|_2^2 - M\sigma^2}{\text{trace}(A^T A)}, \epsilon \right\},$$

$$\tau_t^2 = \frac{\nu_t^2}{N} (N + (\gamma_t^2 - 2\gamma_t)M) + \frac{\gamma_t^2 \sigma^2}{N} \text{trace}(WW^T)$$

誤差分散推定

MMSE推定関数に基づく近接写像



## 比較: 学習パラメータ数

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

- ✓ 学習パラメータ数が少ない → 学習時間が短くなる・学習の安定性が向上
- ✓ 解釈可能性・説明可能性の観点からも学習パラメータ数が少ないことが望ましい

# TISTAの復元性能 (10%が非ゼロとなる状況)

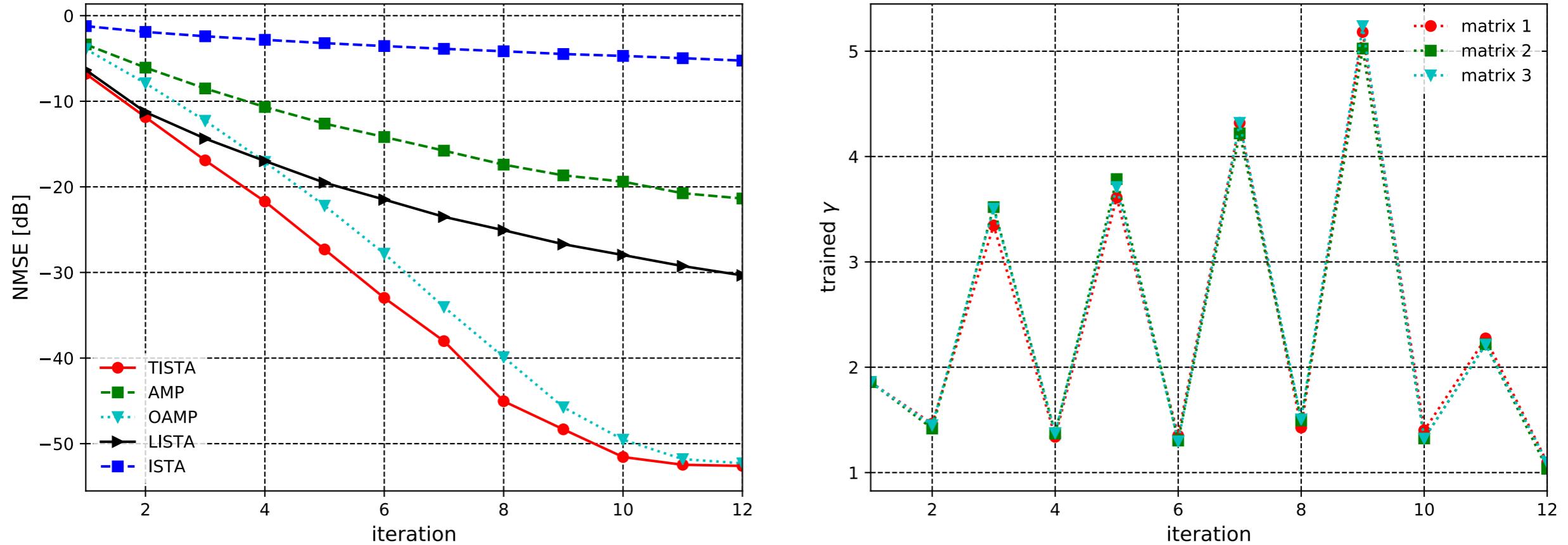


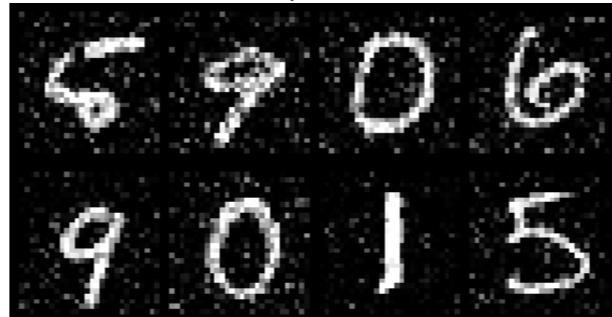
Figure 7: Sparse signal recovery for  $(n, m) = (300, 150)$ ,  $p = 0.1$  and SNR= 50 dB. (Left) MSE performances as a function of iteration steps. (Right) Trained values of  $\gamma_t$  under three different measurement matrices. The initial value is set to 1.0.

# 実データ(MNIST)に基づく実験

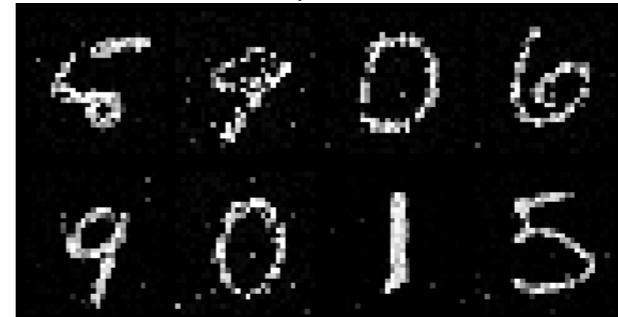
MNISTデータ  
セットに対する  
スパース信号再現

人工データだけではなく  
実データでも良好な特性

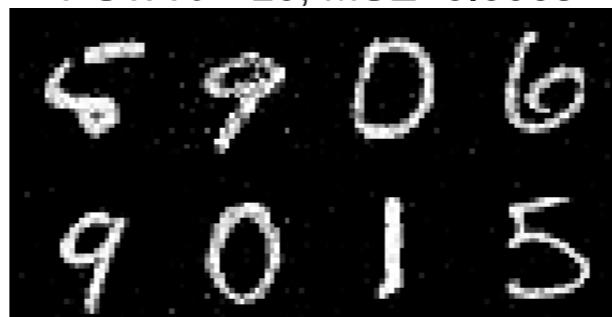
TISTA t = 5, MSE=0.0258



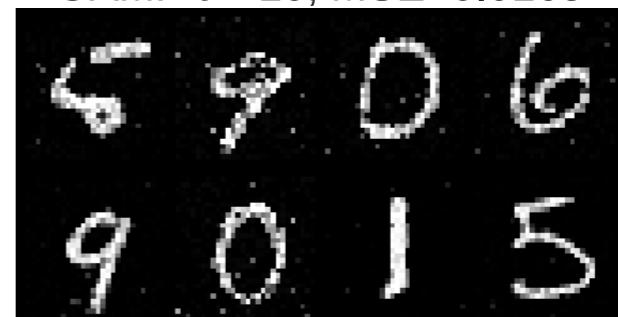
OAMP t = 5, MSE=0.0335



TISTA t = 10, MSE=0.0065



OAMP t = 10, MSE=0.0165



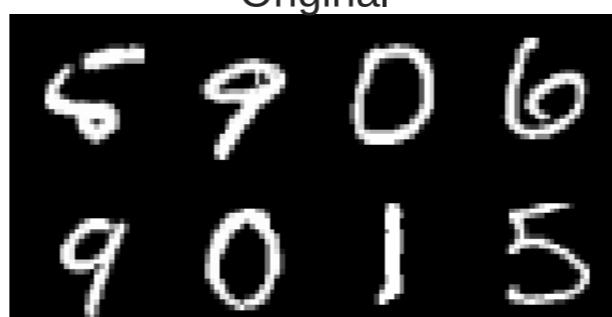
TISTA t = 20, MSE=0.0014



OAMP t = 20, MSE=0.0089



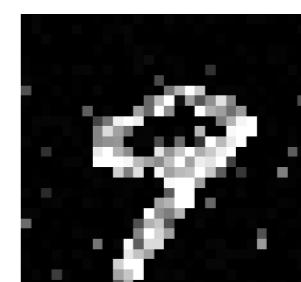
Original



t=20

TISTA

OAMP

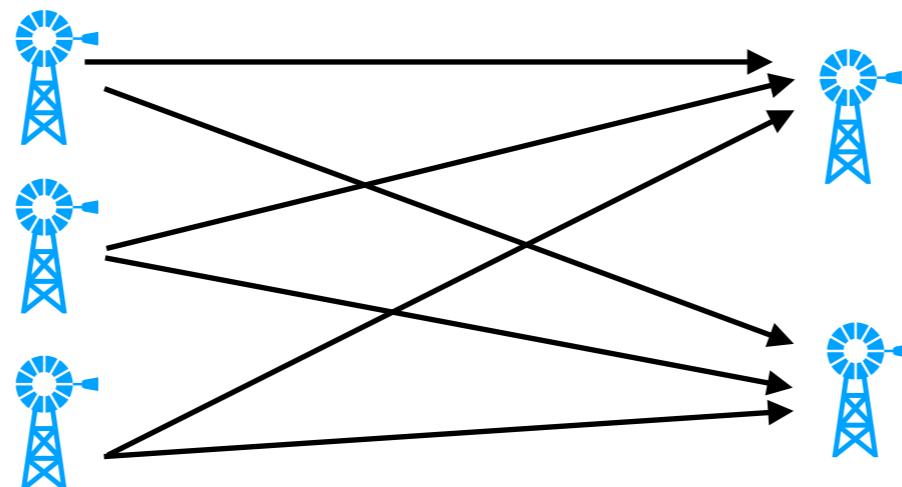


# TISTAのコード

<https://github.com/wadayama/TISTA>

# 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検出器

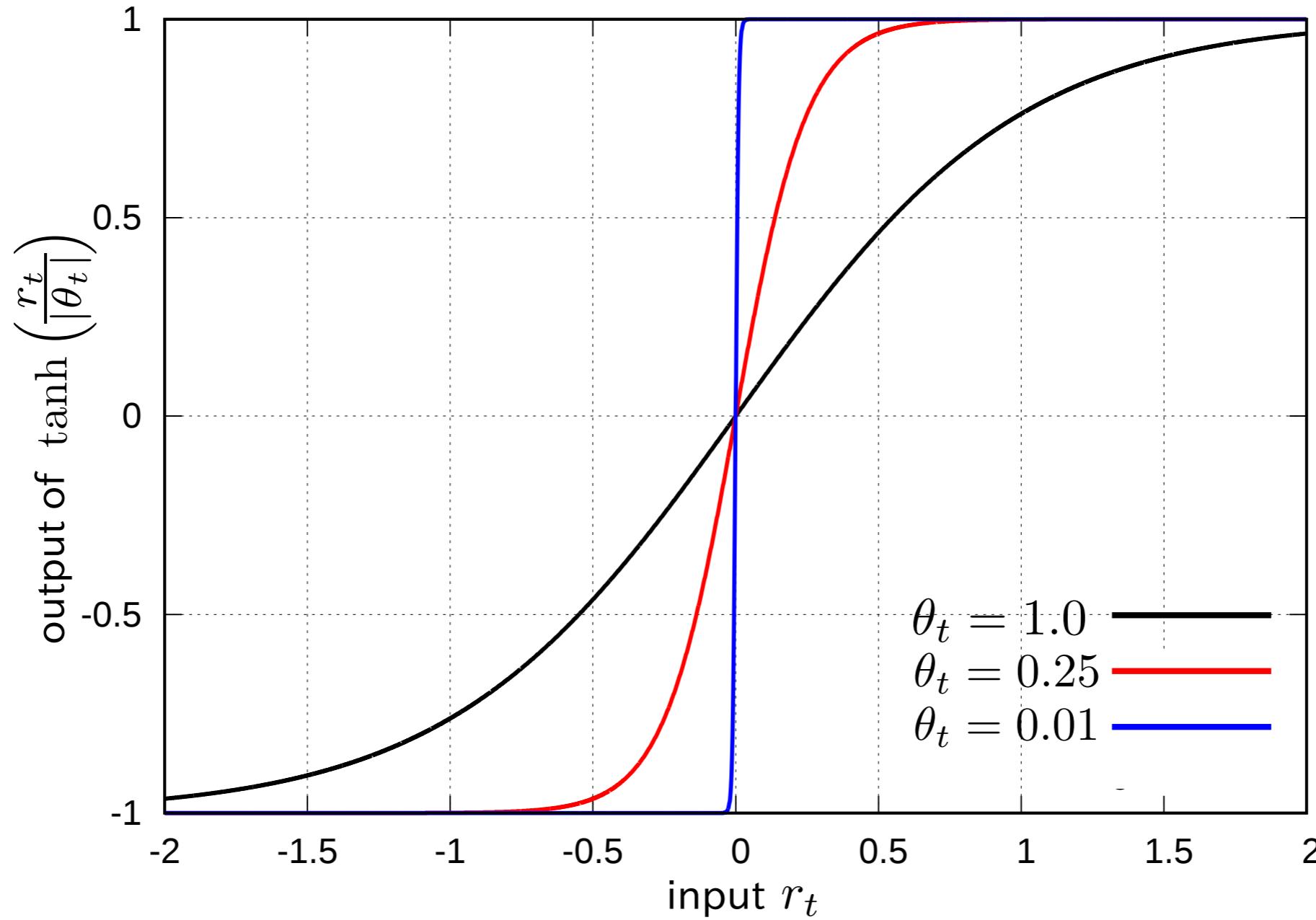
Satoshi Takabe, Masayuki Imanishi, Tadashi Wadayama, Ryo Hayakawa, Kazunori Hayashi, ``Trainable Projected Gradient Detector for Massive Overloaded MIMO Channels: Data-driven Tuning Approach", IEEE Access, July 2019.

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

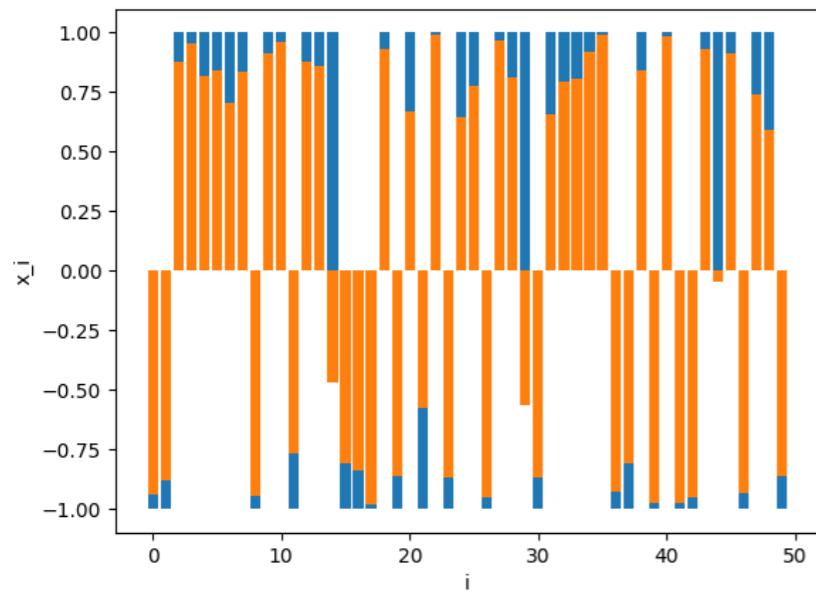
$$s_{t+1} = \tanh\left(\frac{r_t}{|\theta_t|}\right),$$

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

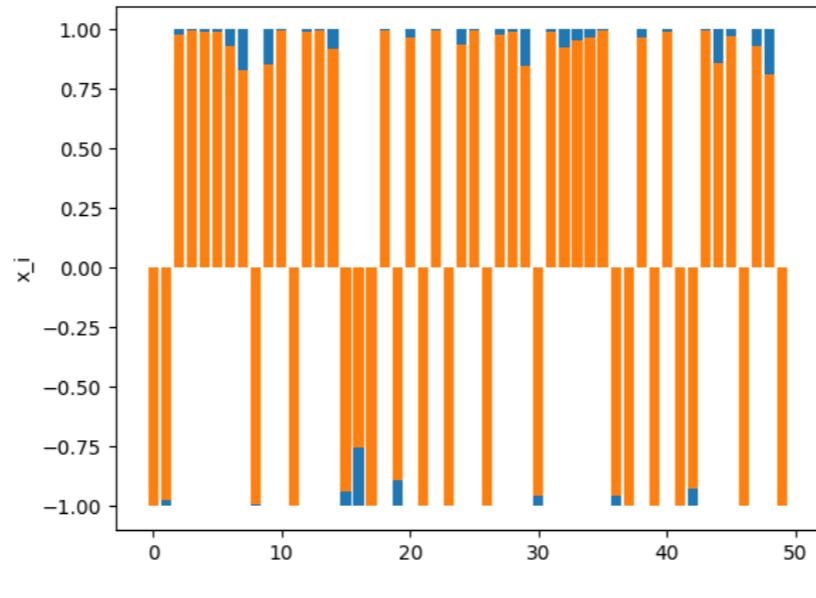
# ソフト射影関数



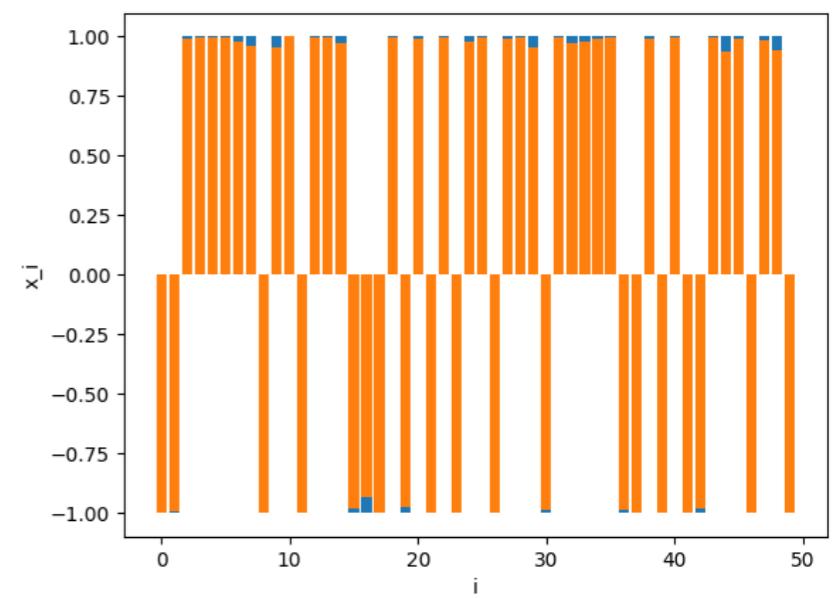
# 推定の様子



$t=1$

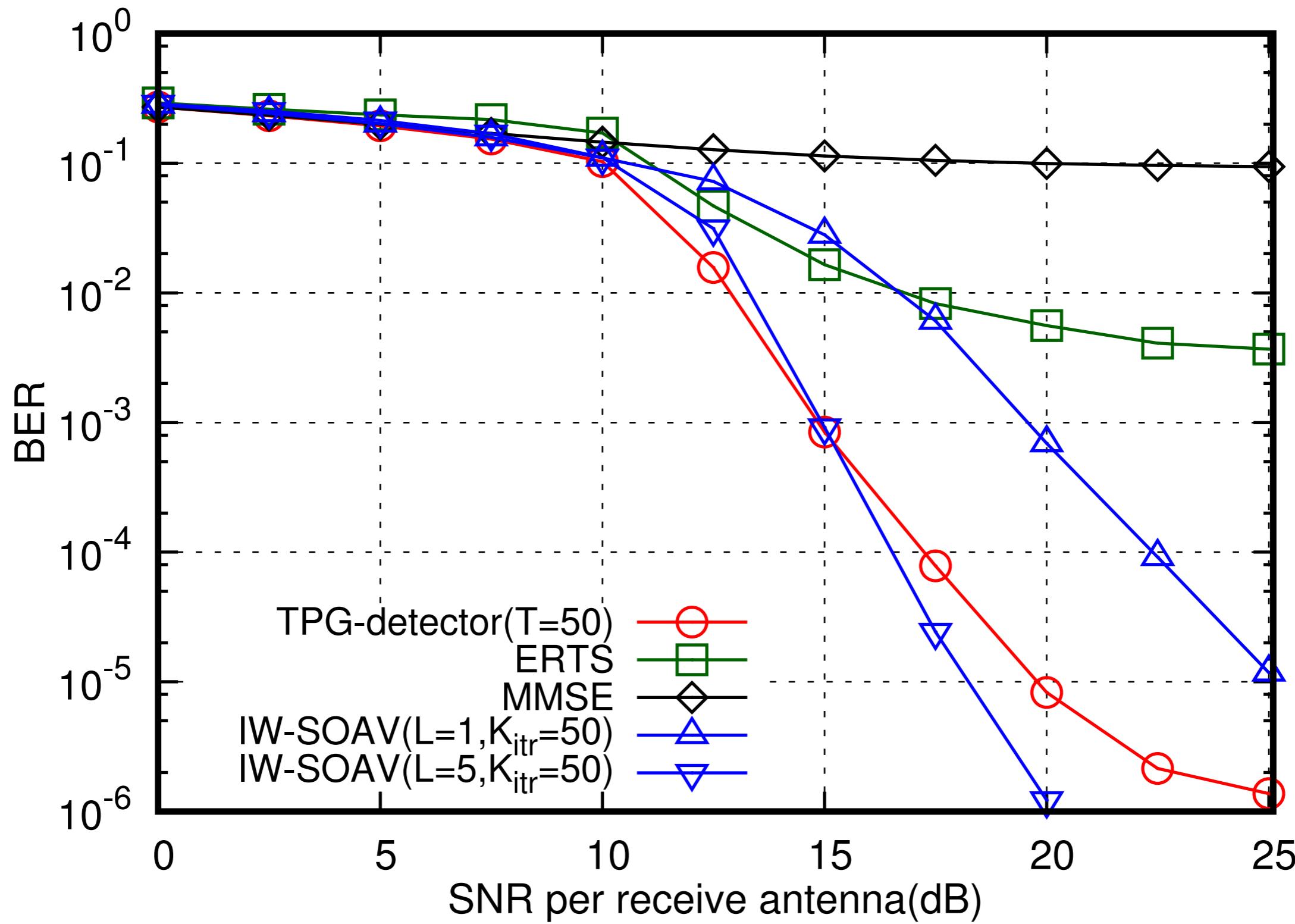


$t=5$



$t=10$

# BER比較: $(n, m) = (150, 96)$



# 複素TISTA(C-TISTA)

Takabe and Wadayama, <https://arxiv.org/abs/1904.07409>, 2019

**非線形観測問題** 関数 $f$ はコンポーネントワイズ関数で、非線形関数でもよい

$$\mathbf{y} = f(\mathbf{A}\mathbf{x}) + \mathbf{w},$$

## C-TISTA

$$\begin{aligned}\mathbf{r}^{(t)} &:= \mathbf{s}^{(t)} + \beta_t h(\mathbf{s}^{(t)}), \\ \mathbf{s}^{(t+1)} &:= \eta(\mathbf{r}^{(t)}; \lambda^{(t)}),\end{aligned}$$

(複素関数 $f$ のウィルティンガー  
微分のチェインルールに基づく)

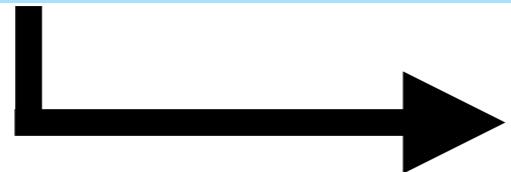
$$\lambda^{(t)} := a_t + b_t \frac{\|\mathbf{y} - f(\mathbf{A}\mathbf{s}^{(t)})\|_2^2}{\text{Tr}(\mathbf{A}^H \mathbf{A})},$$

$$\begin{aligned}h(\mathbf{s}) &:= \mathbf{W} \left[ \{\mathbf{y} - f(\mathbf{A}\mathbf{s})\}^* \odot \frac{\partial f}{\partial z^*}(\mathbf{A}\mathbf{s}) \right. \\ &\quad \left. + \{\mathbf{y} - f(\mathbf{A}\mathbf{s})\} \odot \frac{\partial f^*}{\partial z^*}(\mathbf{A}\mathbf{s}) \right]\end{aligned}$$

# C-TISTAの展開

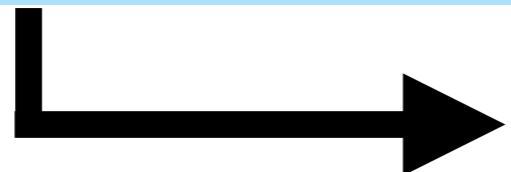
$$y = f(Ax) + w,$$

A:IDFT 行列、f:クリッピング関数



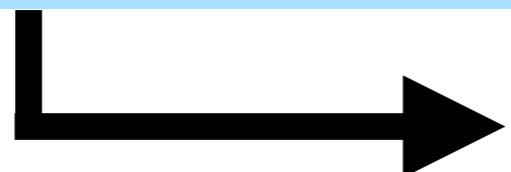
OFDMにおけるPAPR改善  
(クリッピング保証)

A:複素疎行列、f:恒等関数



SCDMAの検出アルゴリズム

A:IDFT 部分行列、f:クリッピング関数



複素数体上の誤り訂正符号

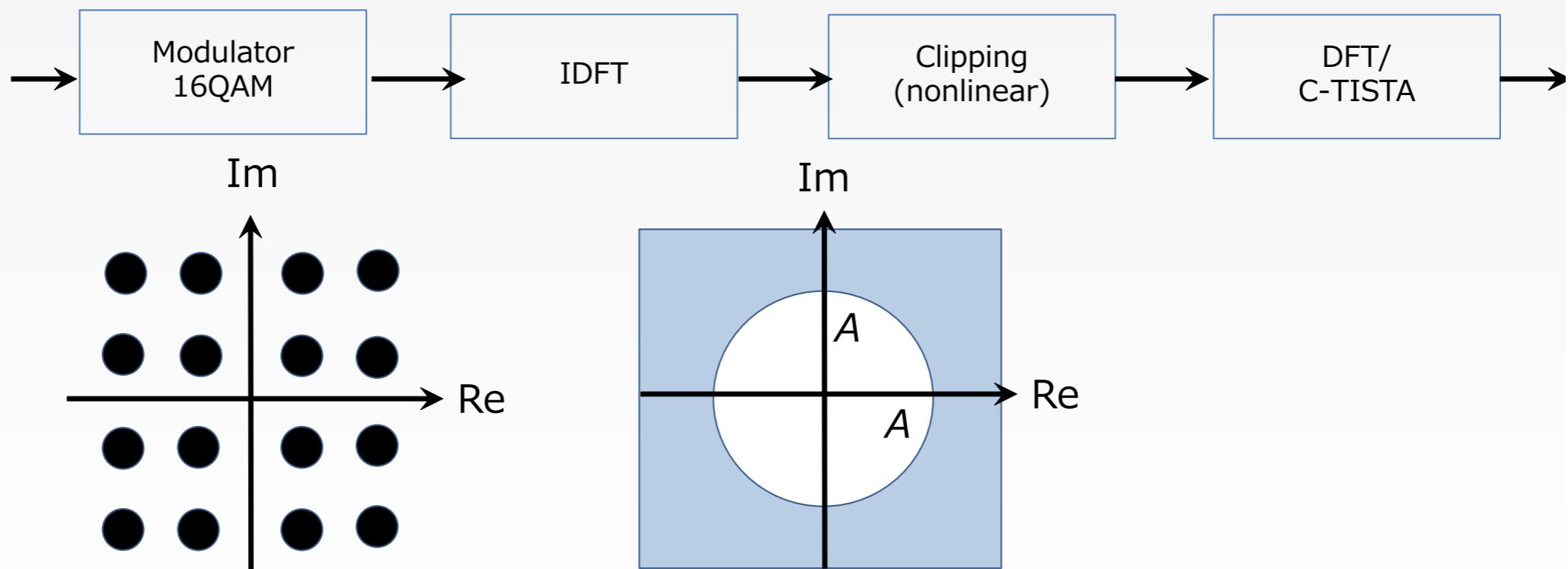
A:通信路行列、f:量子化関数



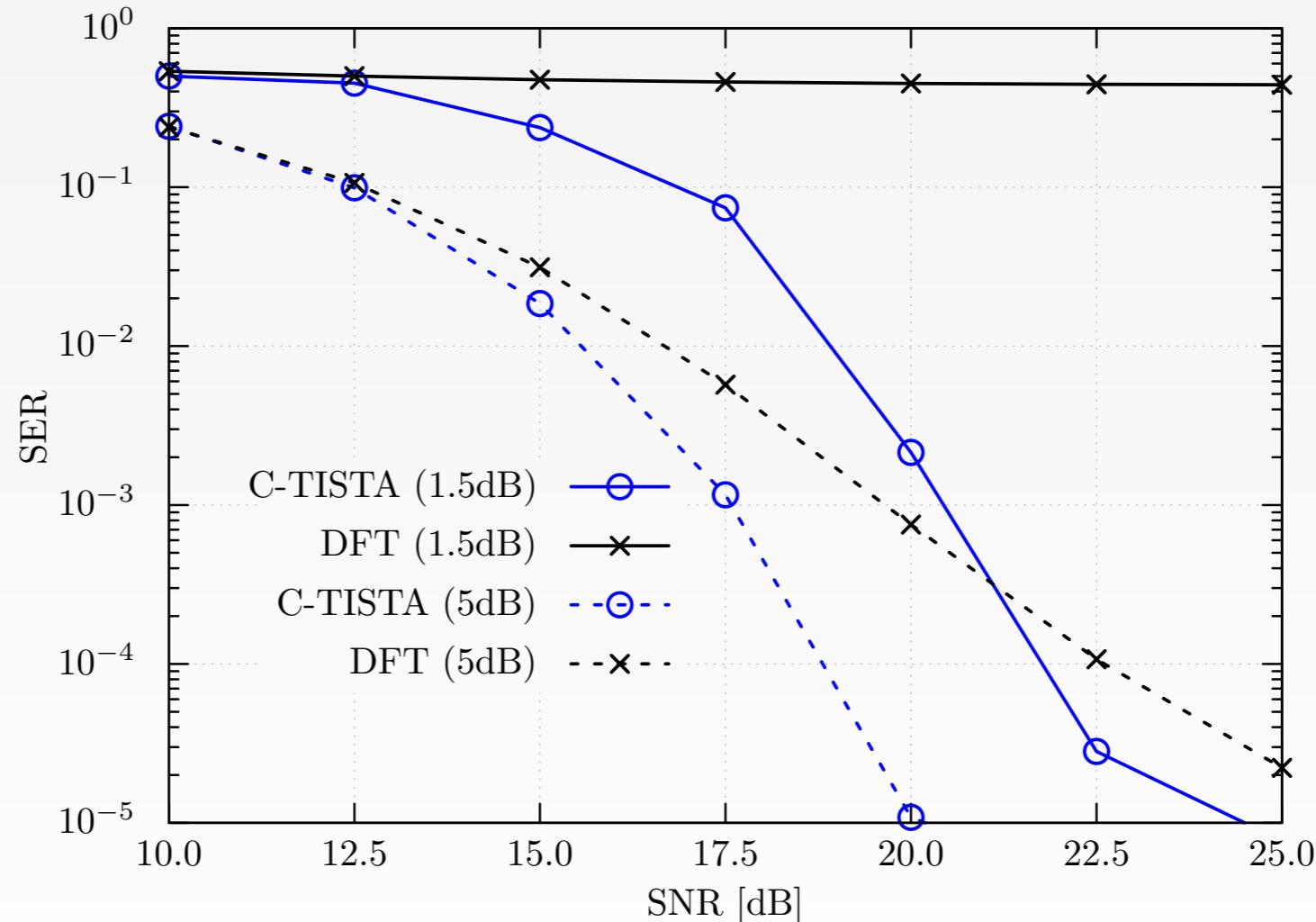
量子化を伴うOverloaded  
MIMO検出

# C-TISTAによるクリッピング補償

- 非線形の clipped OFDM  $y = f(Ax) + n$ 
  - ▶  $(N, M) = (128, 128)$ , ノイズ: SNR で指定
  - ▶  $A$ : 逆離散フーリエ変換 (IDFT) 行列
  - ▶  $f(z) = z (z \leq \alpha), \alpha e^{j\varphi(z)} (z > \alpha)$   
 $\alpha$ : クリッピングレベル (小さいと非線形性強)
  - ▶  $x$  の各成分  $\sim$  16QAM 信号点配置上の一様分布



# C-TISTAによるクリッピング補償



- DFT と比べクリッピングが強い状況でも正しく推定可能

# 複素ISTAについて

## 複素ISTAにFFTを利用する

このプログラムは、ライブラリcomplexlib.pyを利用してるので、手元で実行してください。

問題設定の概要は次の通りです。

- 通信路は複素AWGN通信路
- 送信信号は8PSK
- 送信側ではIFFTをかける
- 信号長は $n$

本コードの実行のためには、tqdmが必要です(インストールされてない場合は、インストールしてください)。ISTA(Takabe and Wadayama, 2019)をベースにしています。詳細については下記をご覧ください。

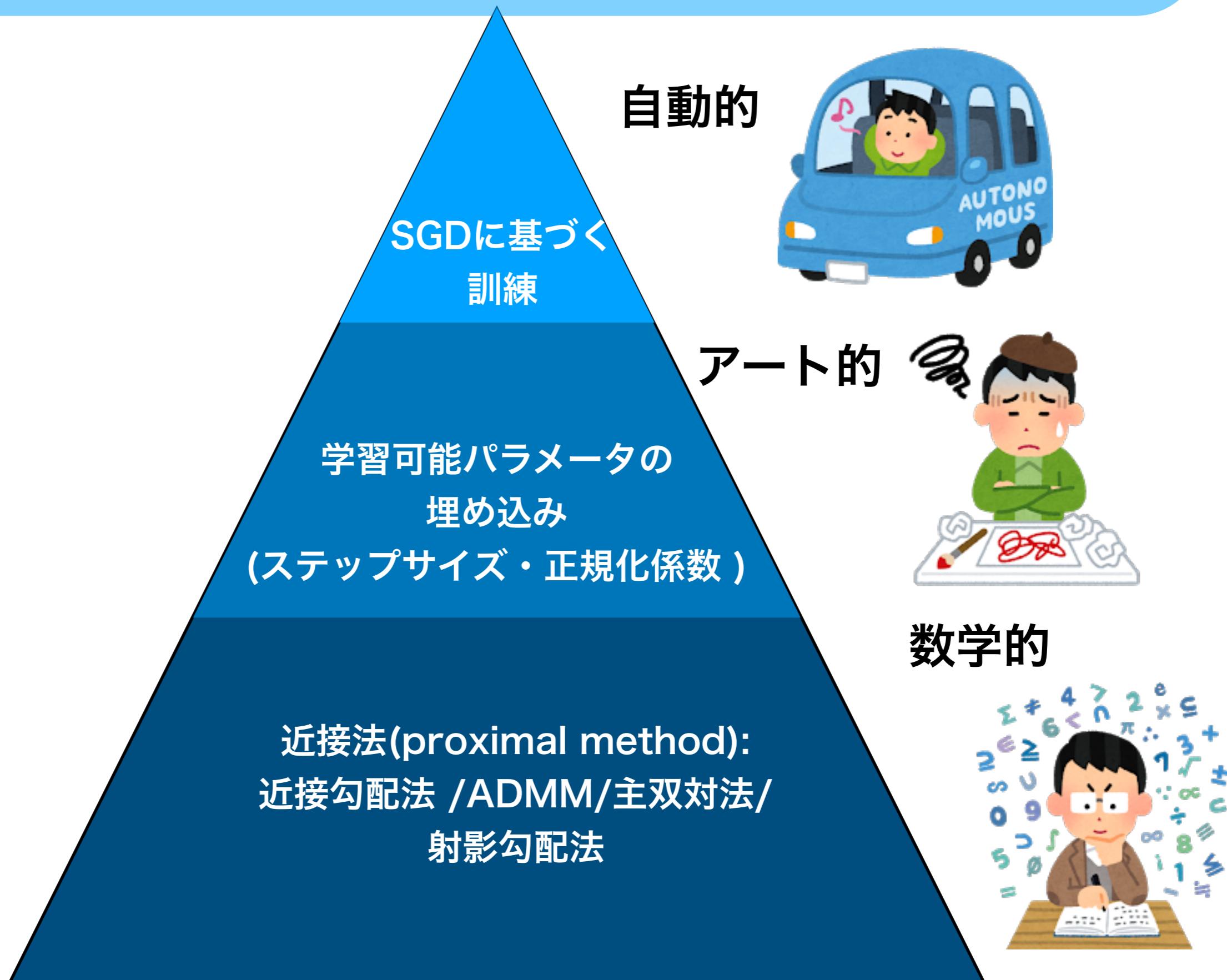
<https://arxiv.org/abs/1904.07409>

- ✓ PyTorchでは複素数は扱えない→複素テンソルライブラリを準備すれば特に問題なし

## 深層展開に基づく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(ステップサイズ・正則化パラメータ・その他)
- ・ 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)

# 近接勾配法の深層展開



## むすび

- 話さなかった話題も多いので、ご興味をお持ちいただいた方は、ぜひテキスト・Githubサイトのほうも覗いてみてください
- われわれの慣れている研究の方法論（原理→アルゴリズム：演繹的）とは異なる方法論（データ→アルゴリズム：帰納的）が必要であり**頭(考え方)の切り替え**が必要
- 深層展開は**シンプルかつ強力**、様々なアルゴリズムのスピードアップが可能
- **深層学習 × 無線通信**：研究テーマ探してらっしゃる若い方にお勧めです！

# ご清聴ありがとうございました

- 第一部：

- (a) 研究環境を整える
- (b) AND関数を学習する
- (c) MIMO検出器を作る

Q&A

- 第二部：

- (a) 深層展開
- (b) スパース信号復元とISTA(近接勾配法)
- (c) 「深層展開」に関する関連研究

Q&A

# 付録

# 深層学習関連の研究を進めていく上で

- 説得力のあるベースラインを用意する
- ハイパーパラメータ(学習率など)の調整に時間をかけすぎない
- Githubなどで公開されているコードを読む・利用させていただく
- 学習して得られた結果を解釈する努力をする
- 実験の再現性を確保する