知能工学特別講義 第６講

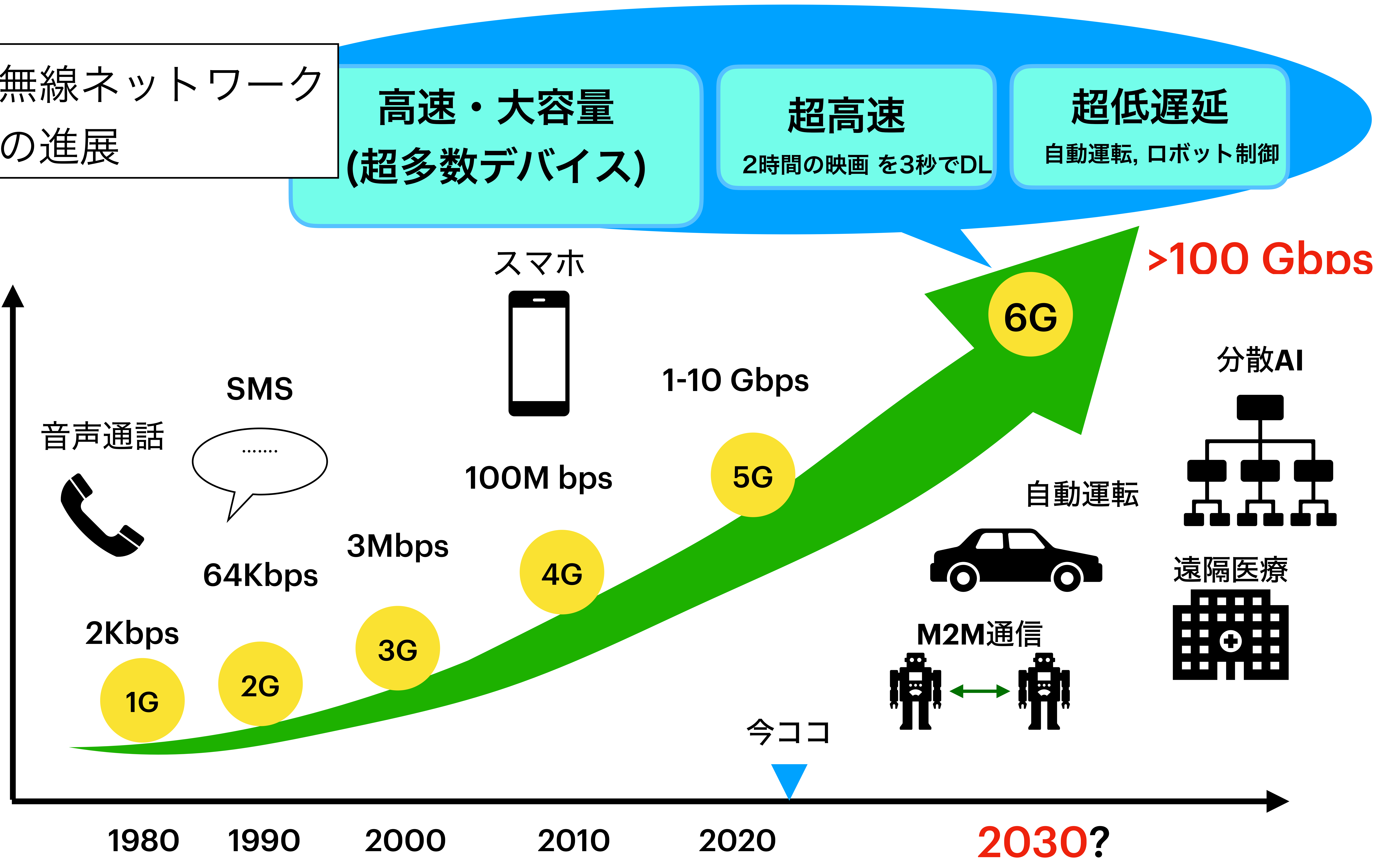
担当：和田山 正

名古屋工業大学

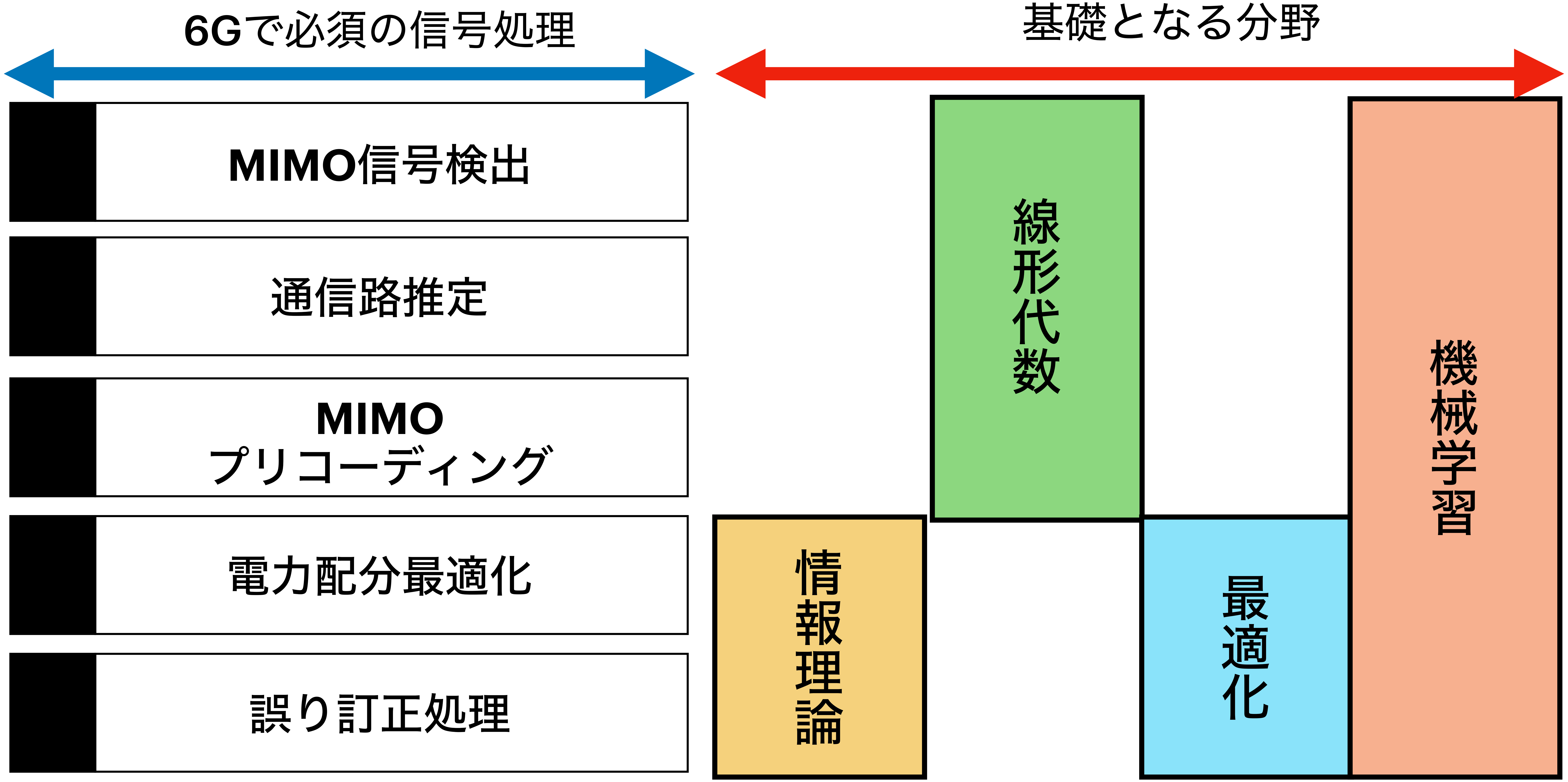
本講義の内容

- 信号処理における深層学習
- MIMO通信路と学習型信号検出アルゴリズム
- 深層展開による反復型アルゴリズムの性能改善

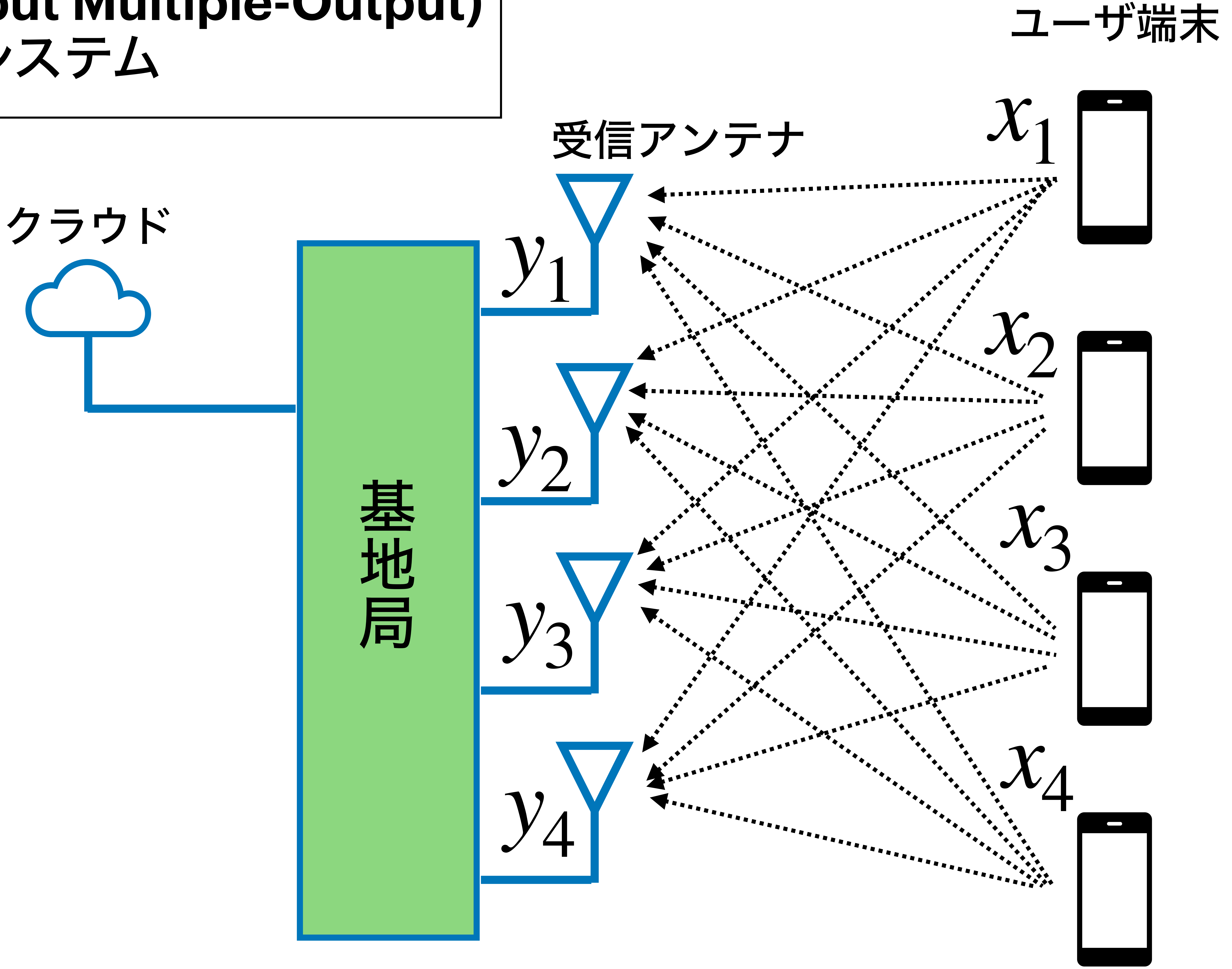
無線ネットワークの進展



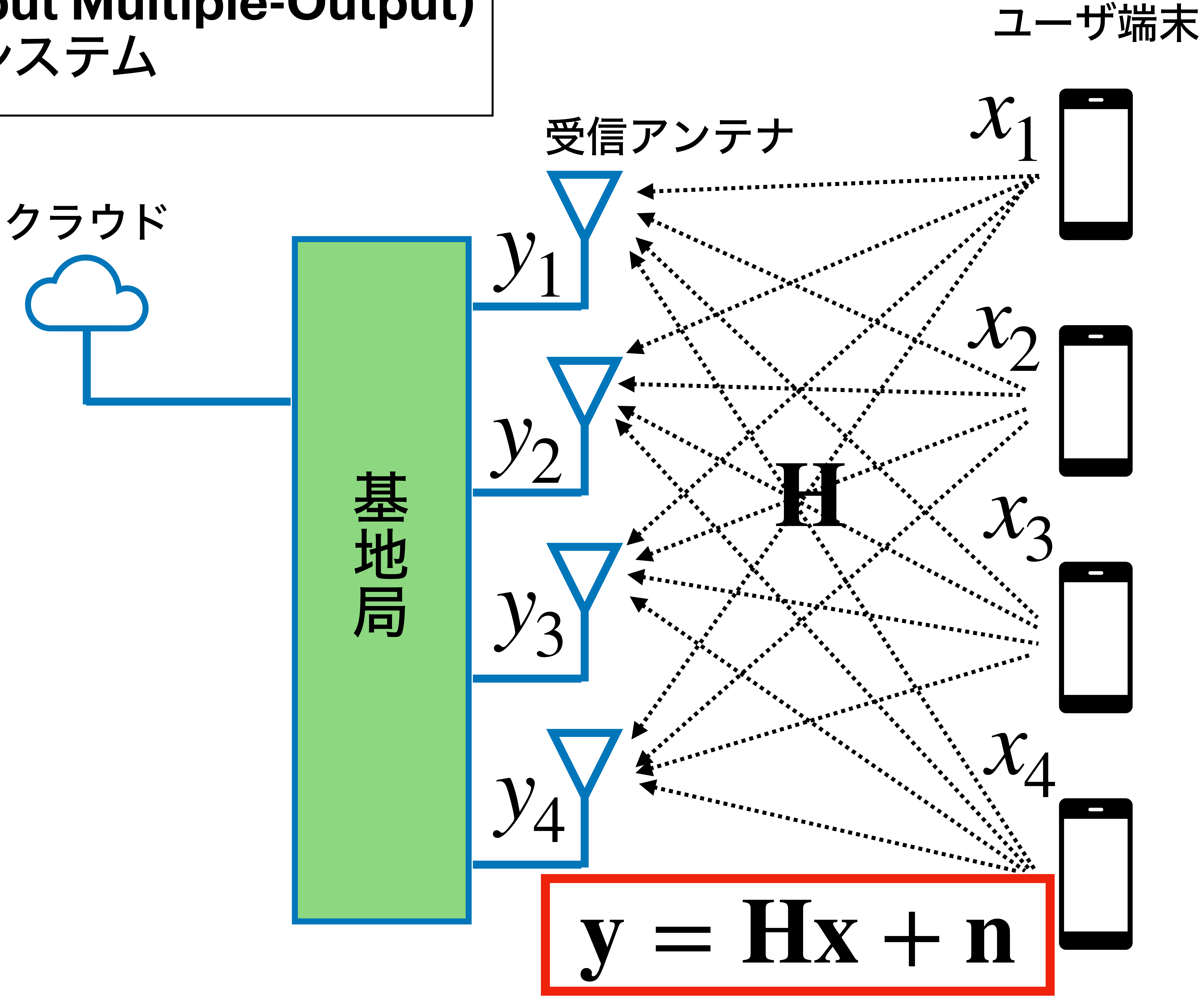
6G信号処理とその基礎



MIMO(Multiple-Input Multiple-Output) 通信システム



MIMO(Multiple-Input Multiple-Output)
通信システム



MIMO(Multiple-Input Multiple-Output)
通信システム

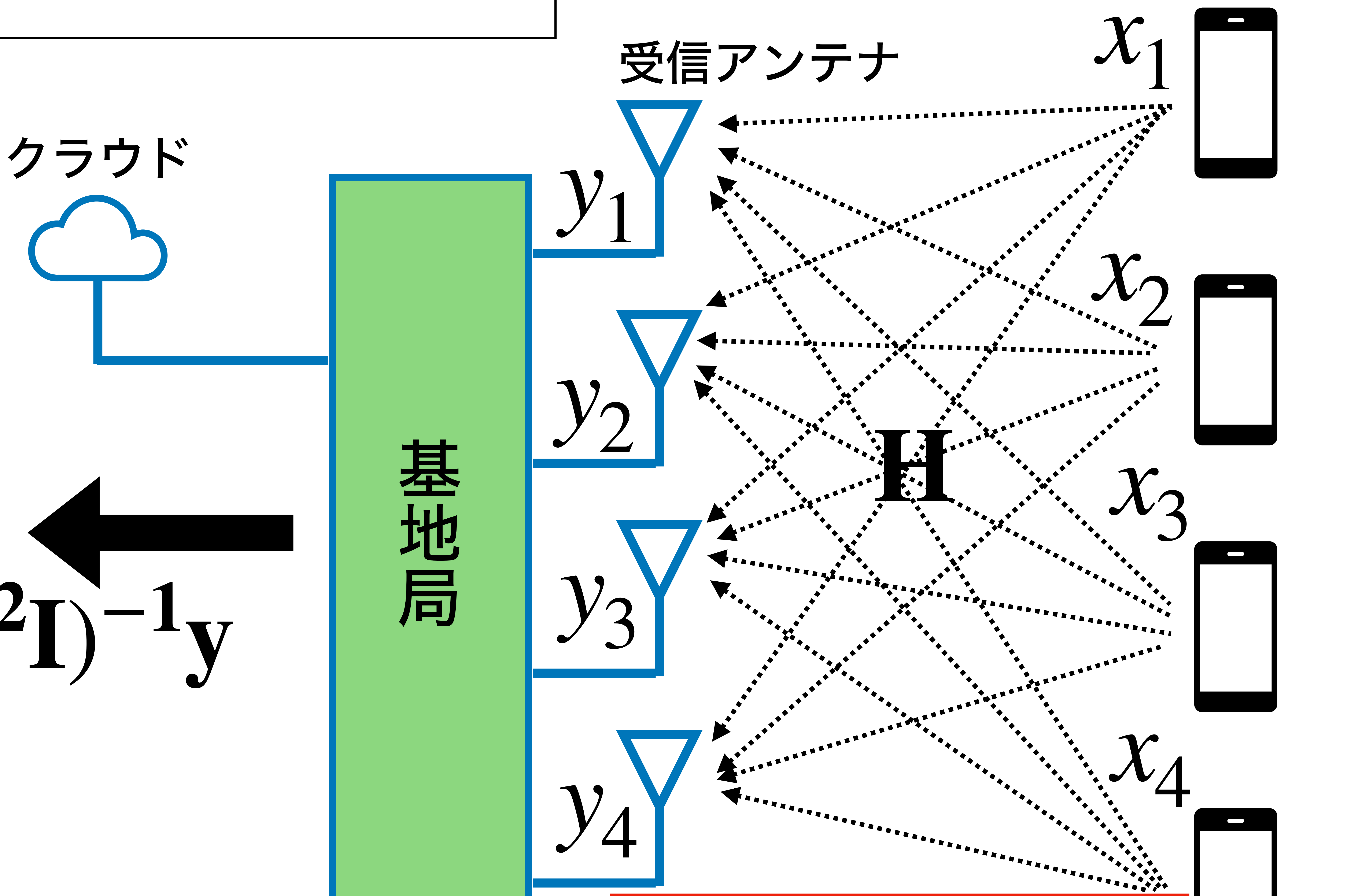
送信信号の推定

$$(x_1, x_2, x_3, x_4)$$

を推定したい

MMSE推定法

$$\hat{\mathbf{x}} = \mathbf{H}^{\mathrm{H}}(\mathbf{H}\mathbf{H}^{\mathrm{H}} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$$

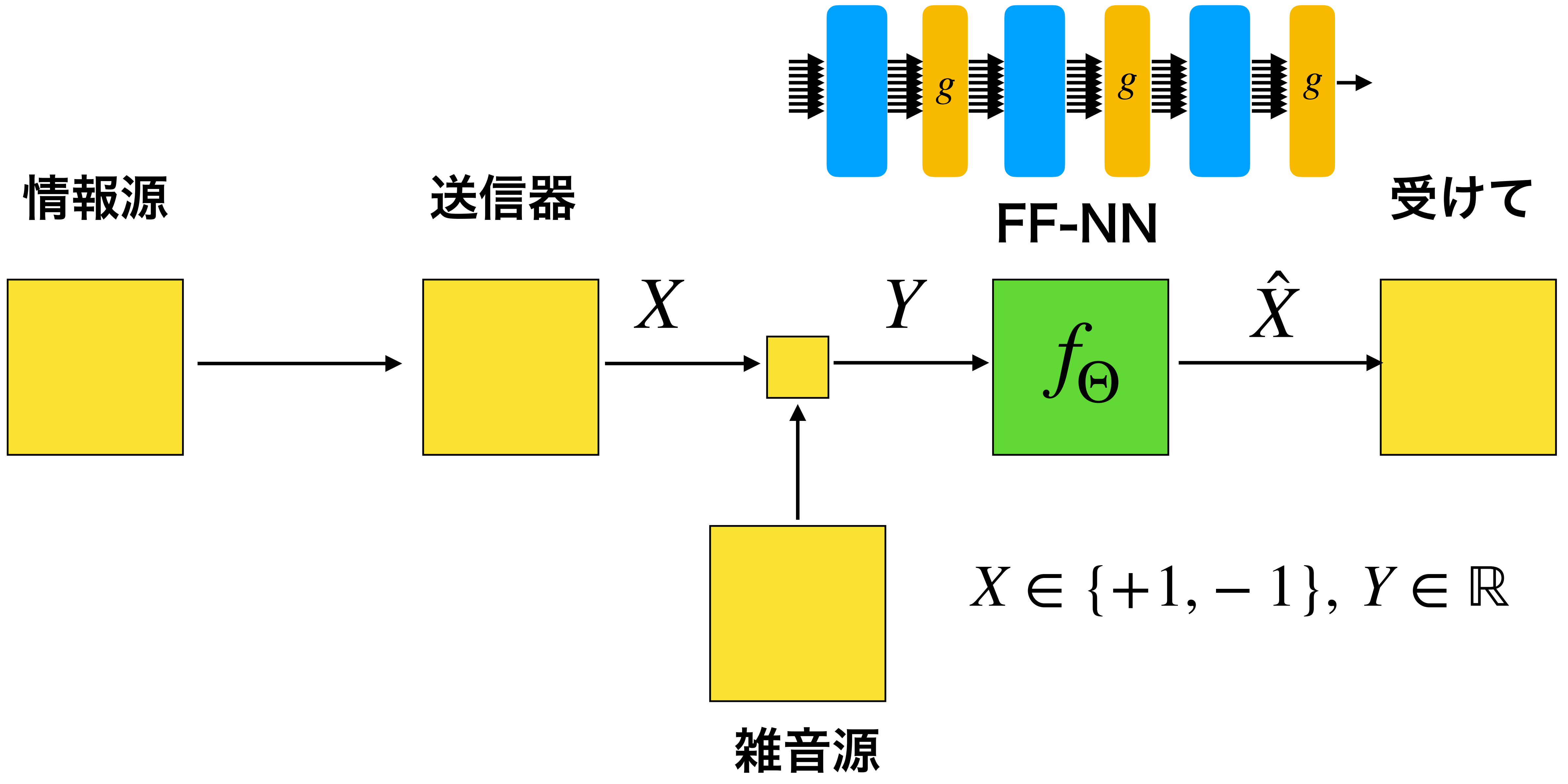


$$\boxed{\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}}$$

無線通信ネットワークの将来象: 無線通信 + AI

「無線通信＋AI」に強い興味が持たれてきている

深層ネットワークに基づく信号検出器



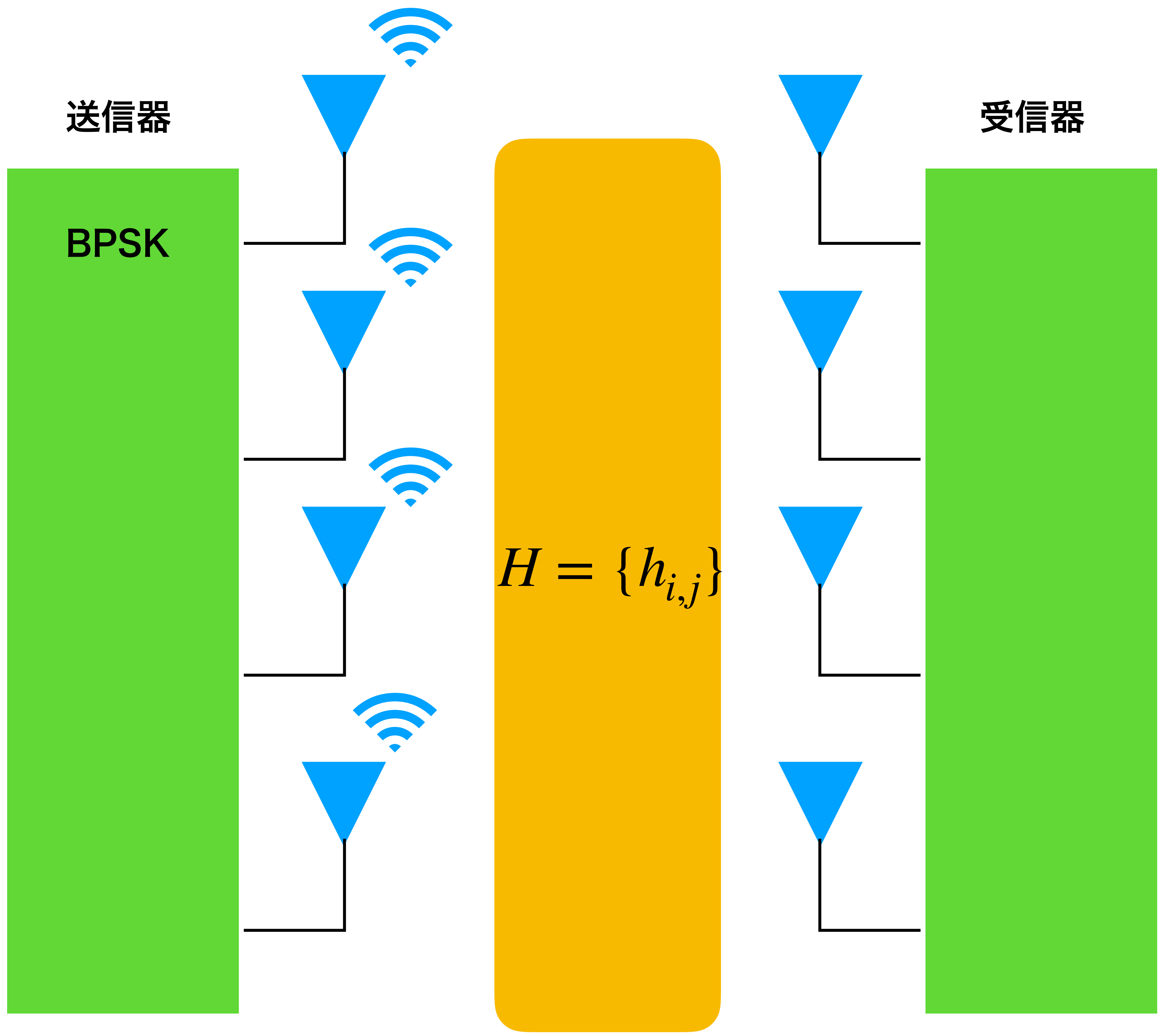
この形の検出器の論文はかなり数が出ています。その一例：

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

MIMO信号検出問題

送信信号: $\{+1,-1\}^n \quad H \in \mathbb{R}^{n\times n}$

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

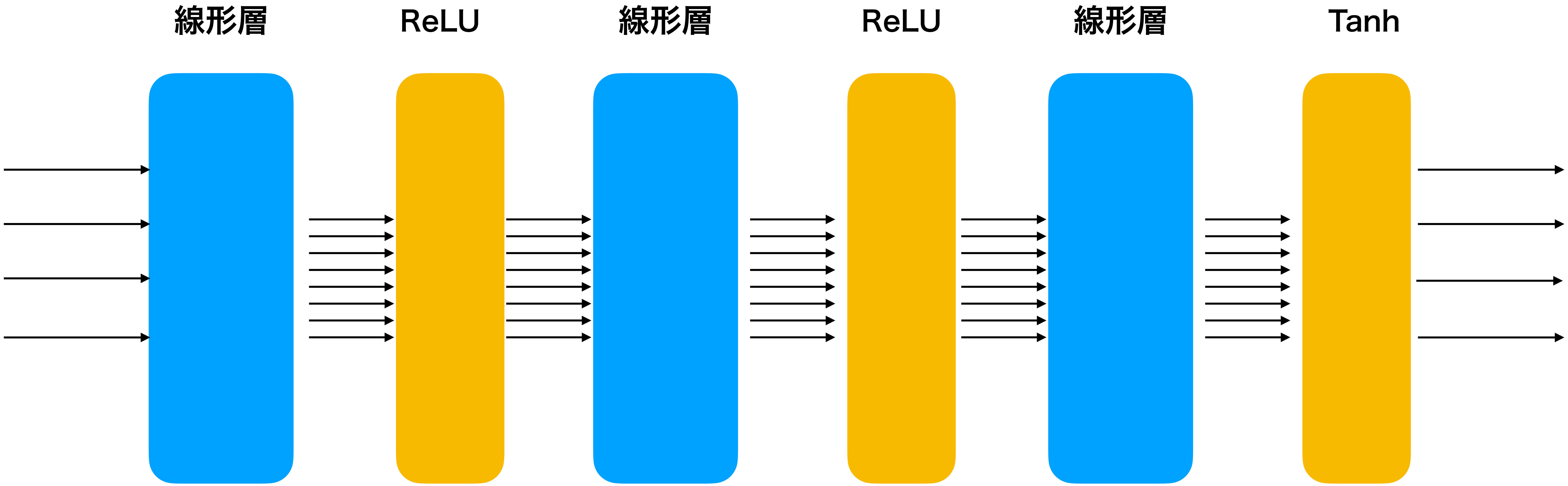
最尤推定則

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

- 最尤推定(ML)はビット誤り率的には最良の推定方法
- 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に乗じる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？

ニューラルMIMO信号検出器

$$y = Hx + w$$



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。。。)

ニューラルMIMO信号検出器の実装(1)

グローバル定数の設定 ¶

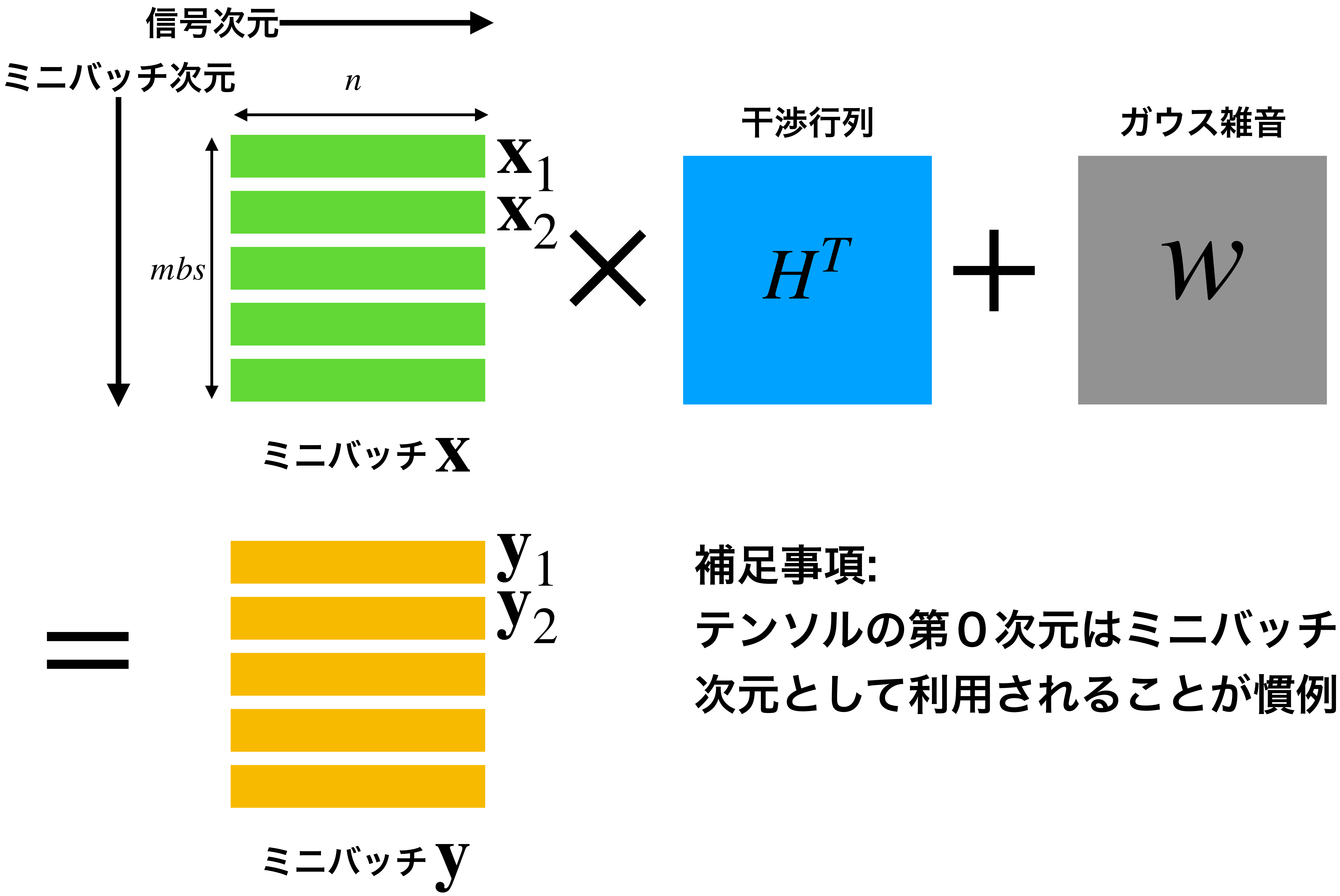
```
]: mbs = 50 # ミニバッチサイズ
   noise_std = 0.5 # 通信路において重畳される加法的白色ガウス雑音の標準偏差 (sigma)
   n = 4 # アンテナ数
   h = 50 # 隠れ層のユニット数
   H = torch.normal(mean=torch.zeros(n, n), std=1.0) # 干渉行列
   adam_lr = 0.001 # Adamの学習率
```

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

ニューラルMIMO信号検出器の実装(2)



補足事項:
テンソルの第０次元はミニバッチ次元として利用されることが慣例

ニューラルMIMO信号検出器の実装(3)

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
```

深層展開 (Deep Unfolding)

- 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)
- モデルベース
- 多くの場合、収束速度の向上が得られる

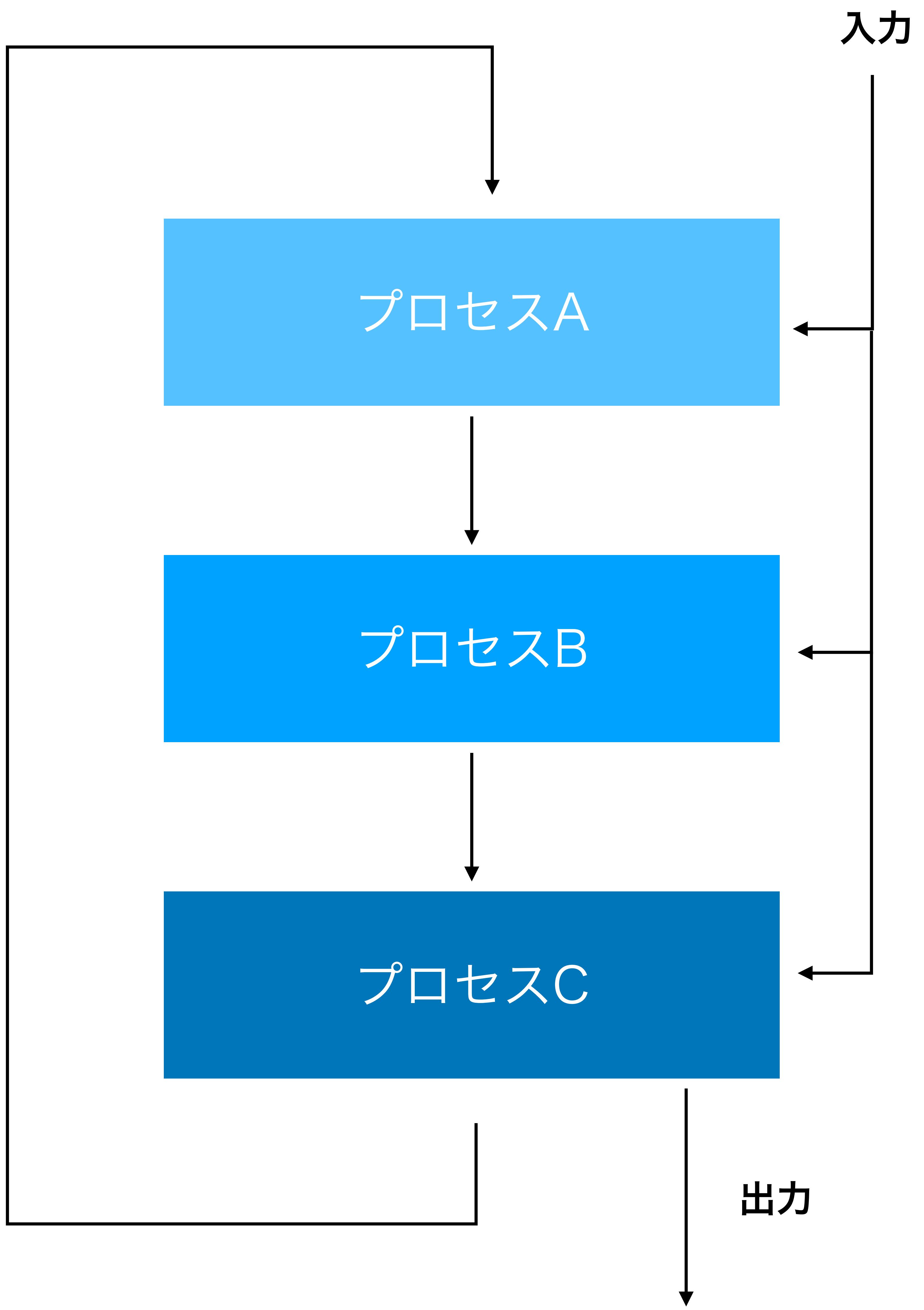
深層展開 (Deep Unfolding)

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)

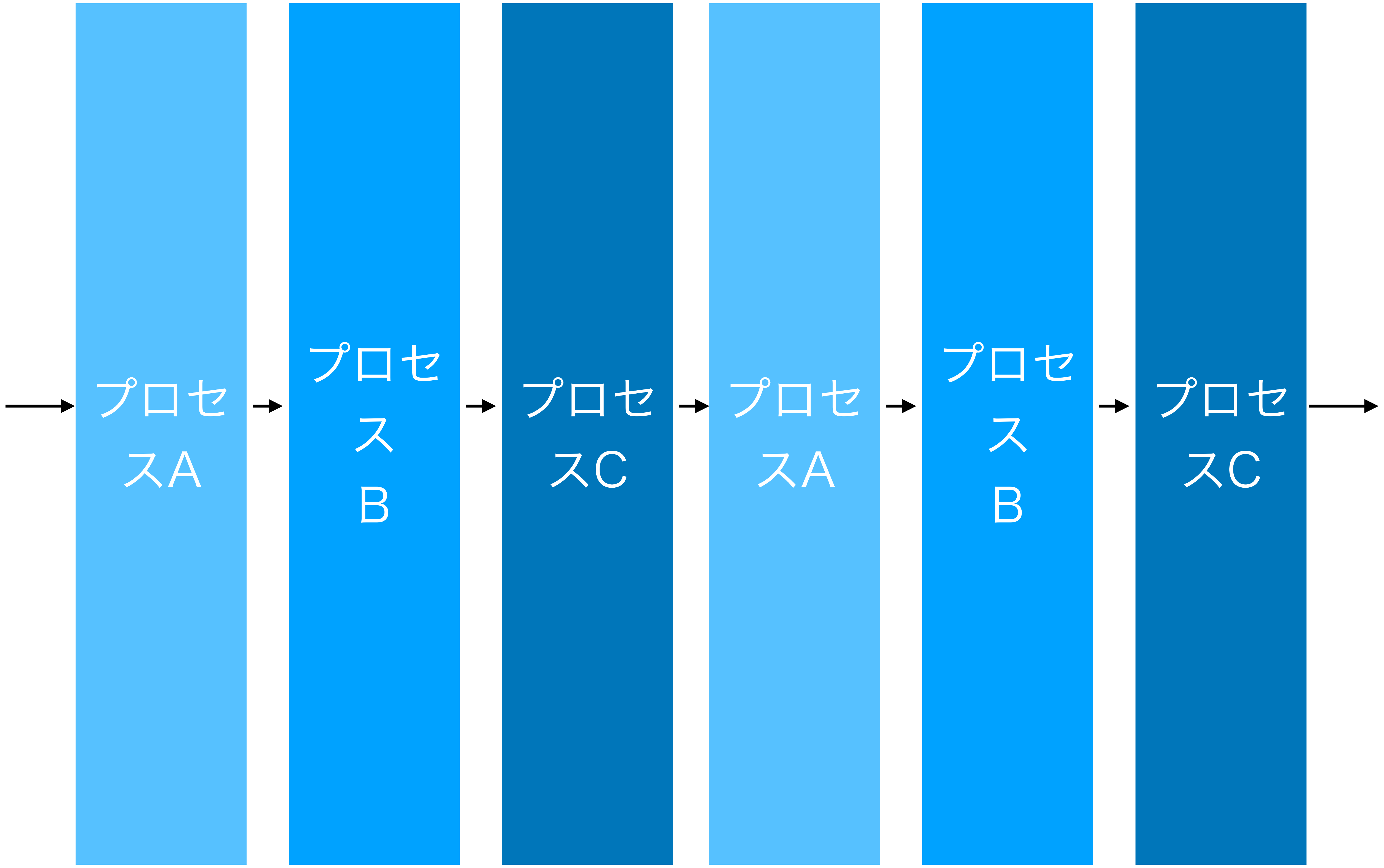
反復アルゴリズムの構造



- ビリーフプロパゲーション
- ガウス・ザイデル法
- ヤコビ法
- 共役勾配法
- 反復収縮法(ISTA)
- AMP
- ADMM

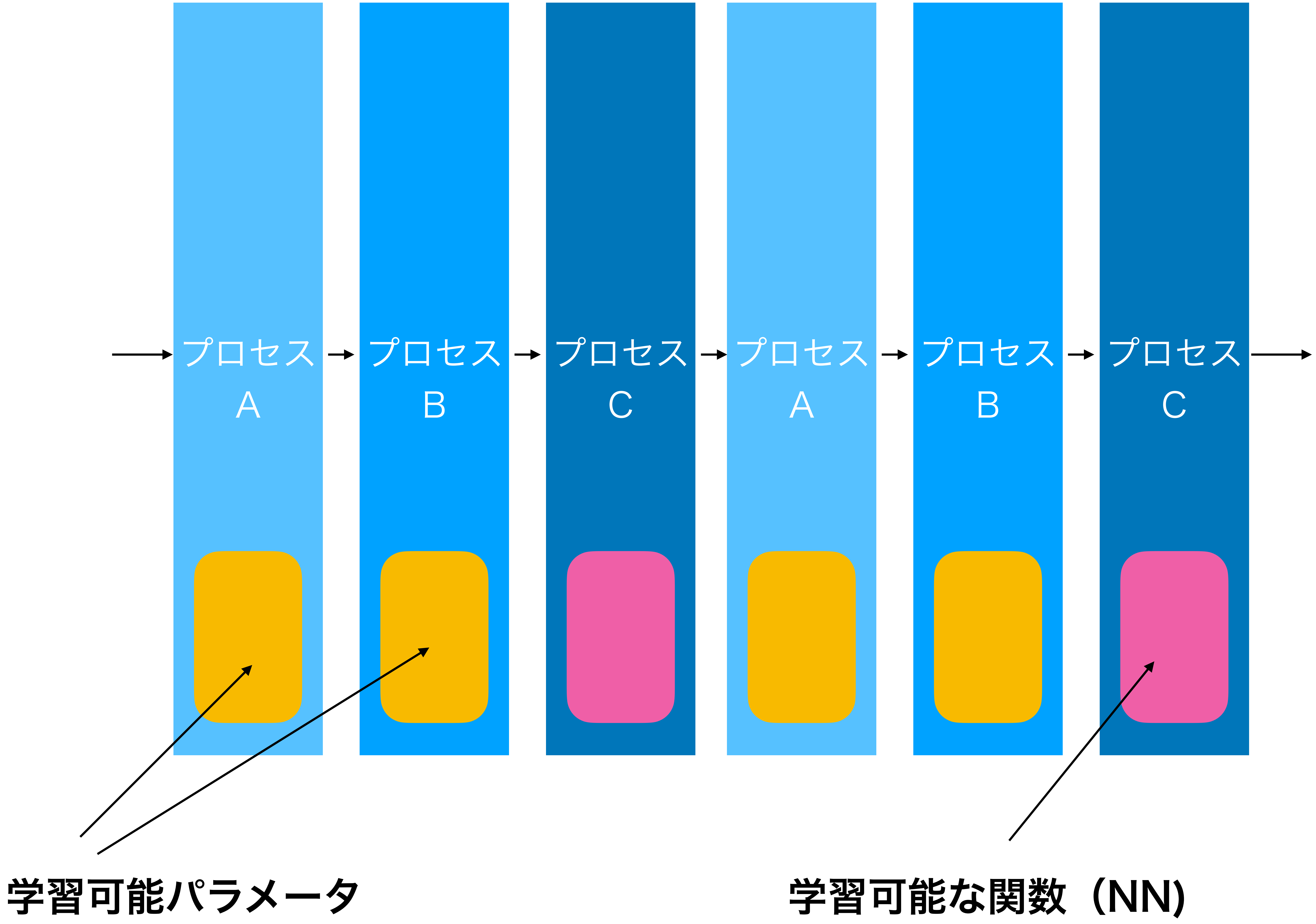
…

信号フローを時間方向に展開

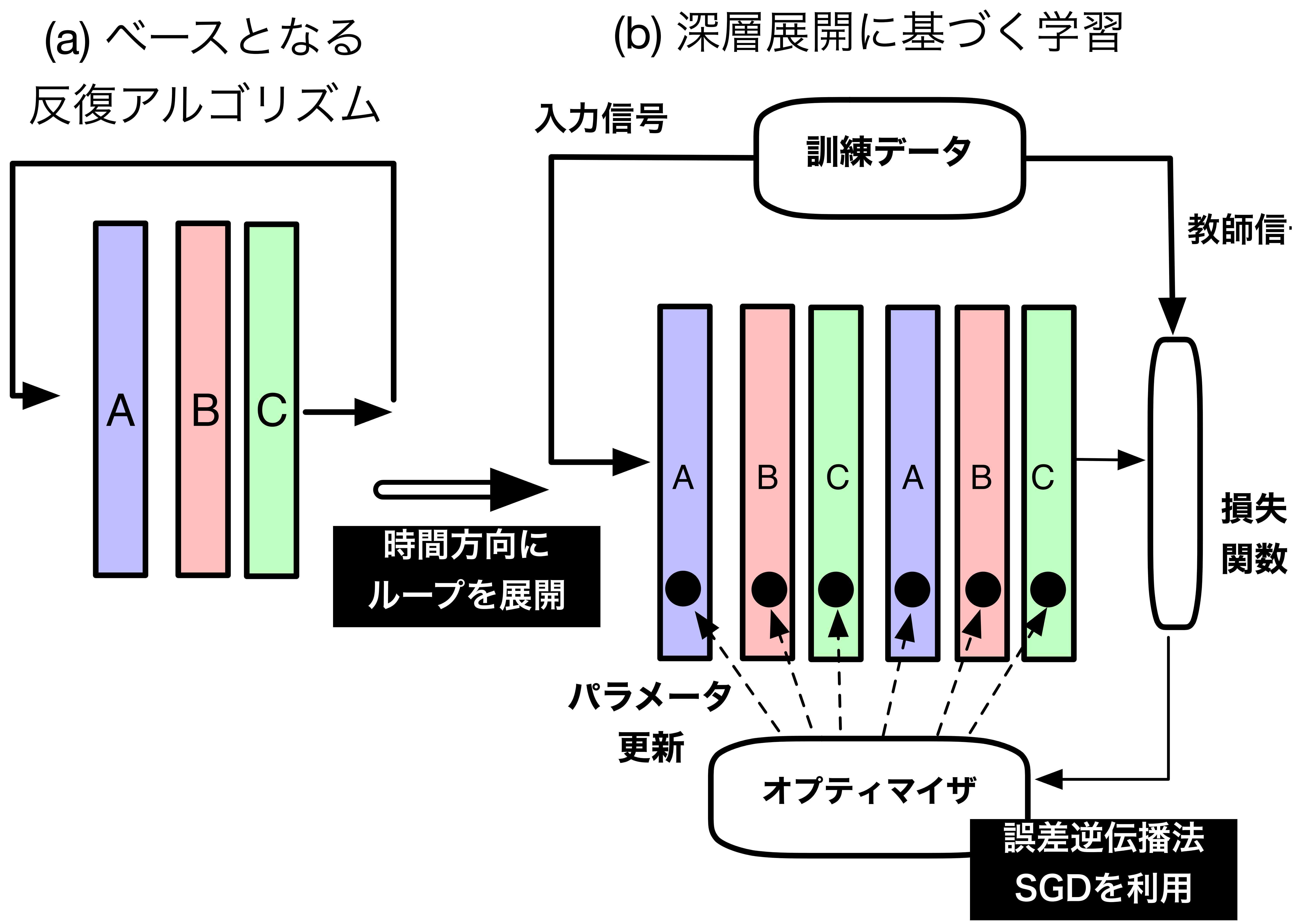


各プロセスが微分可能であり，かつ，その導関数がほぼ至るところでゼロでなければ，backprop可能

学習パラメータを埋め込む



深層展開



(a) ベースとなる
反復アルゴリズム
A
B
C
時間方向に
ループを展開
(b) 深層展開に基づく学習
入力信号
訓練データ
教師信
A
B
C
A
B
C
損失
関数
パラメータ
更新
オプティマイザ
誤差逆伝播法
SGDを利用

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

深層展開のメリット

- 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

$$f(x_1, x_2) = x_1^2 + q x_2^2$$

通常の勾配法(GD)の基本ステップ

$$\boldsymbol{s}_{t+1} = \boldsymbol{s}_t - \gamma \nabla f(\boldsymbol{s}_t)$$

学習可能パラメータを含む勾配法(TGD)の基本ステップ

$$\boldsymbol{s}_{t+1} = \boldsymbol{s}_t - \gamma_t \nabla f(\boldsymbol{s}_t).$$

Trainable Gradient Descent の実装

TGD クラス (Trainable Gradient Descent)

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

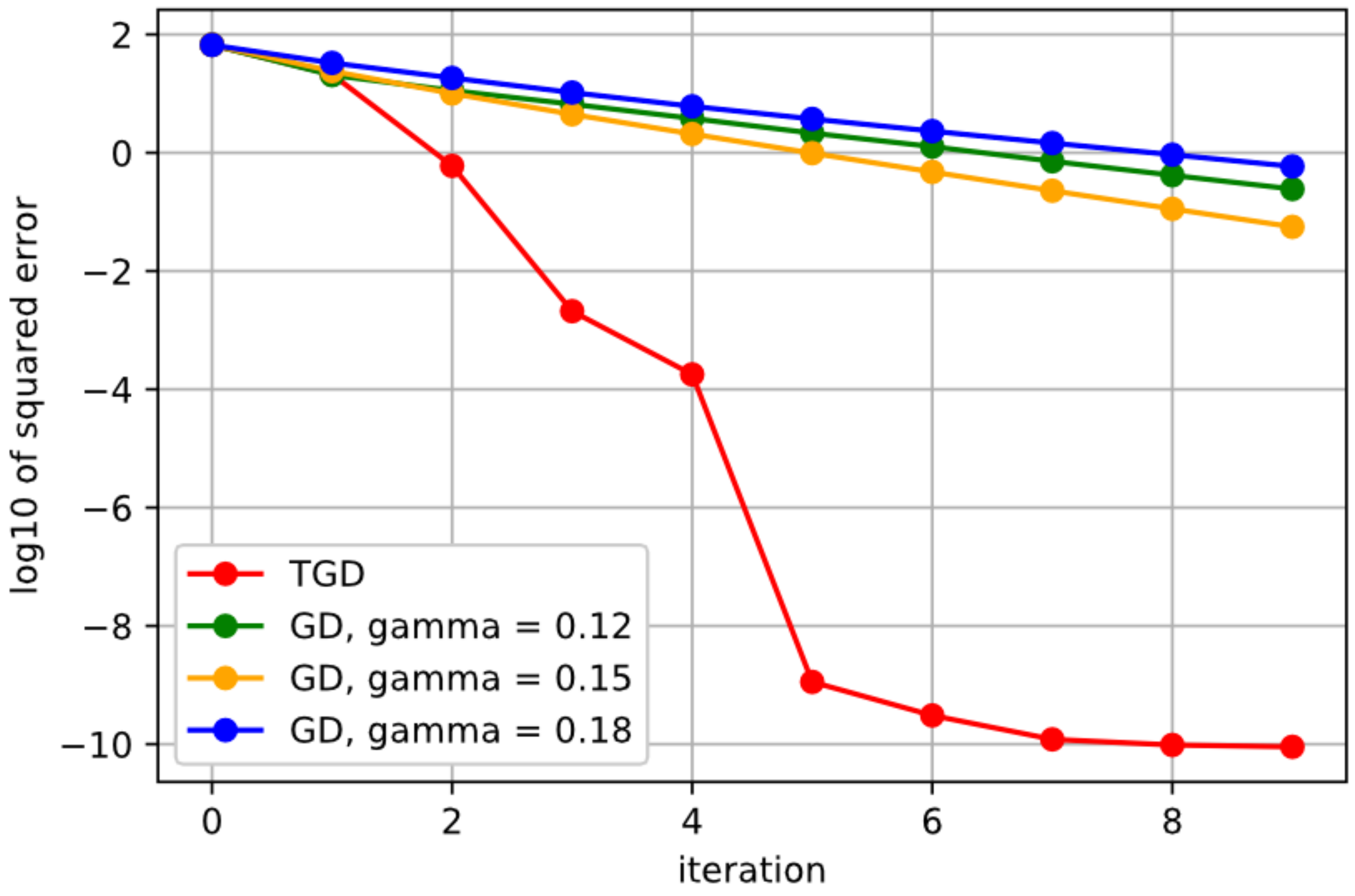
```
model = TGD(itr)
opt   = optim.Adam(model.parameters(), lr=0.001)
loss_func = nn.MSELoss()
solution = torch.tensor([[0.0, 0.0]]).repeat(bs,1) #解
for gen in range(itr):
    for i in range(1000):
        opt.zero_grad()
        x_hat = model(gen + 1, bs)
        loss  = loss_func(x_hat, solution)
        loss.backward()
        opt.step()
    print(gen, loss.item())
```

インクリメンタルトレーニング
(二重ループになっている)

２次関数最小化における深層展開

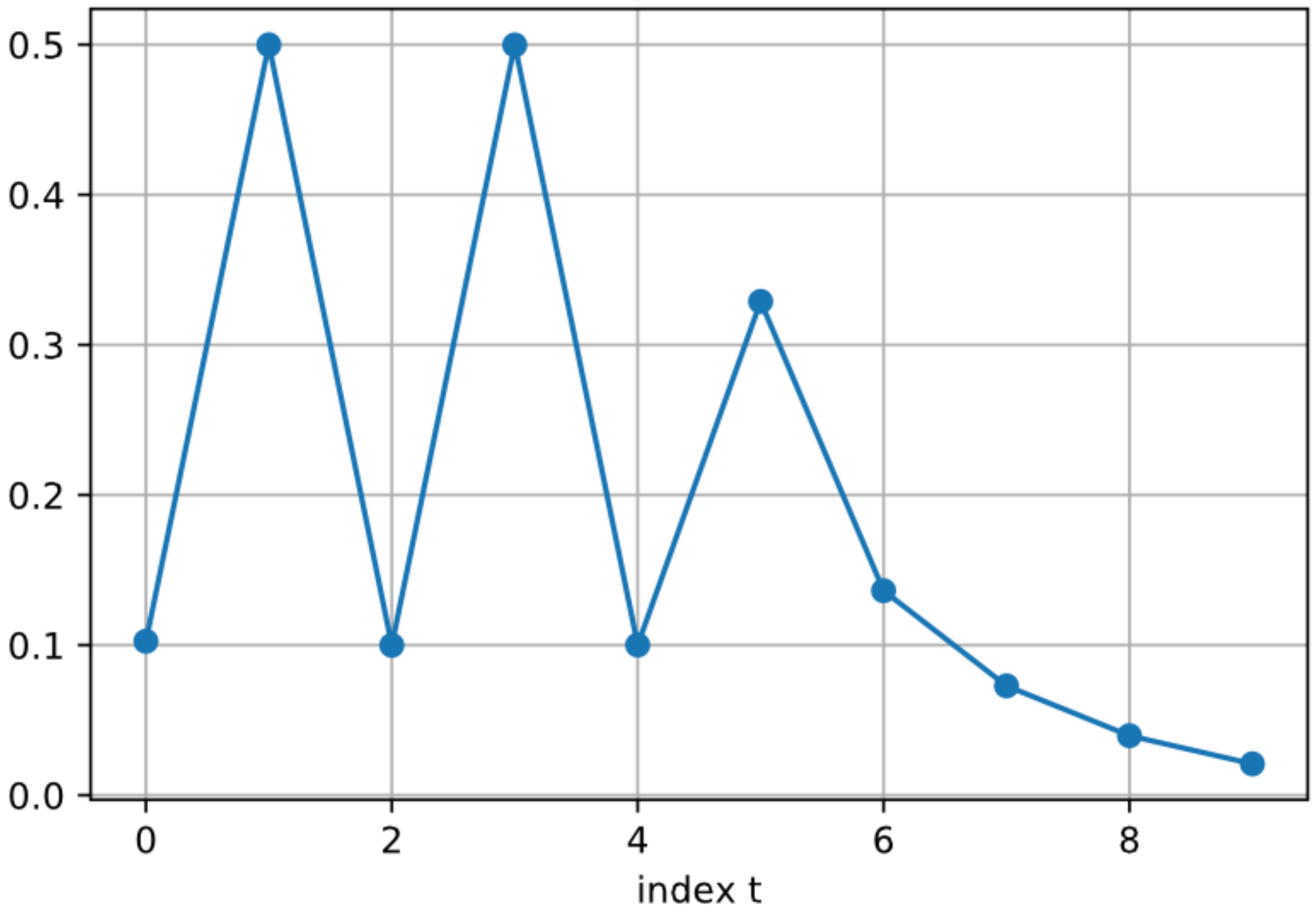
真値からの誤差

q = 5



学習可能パラメータの値

q = 5



実験結果からの観察

- 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

ISTA: LASSO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)

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

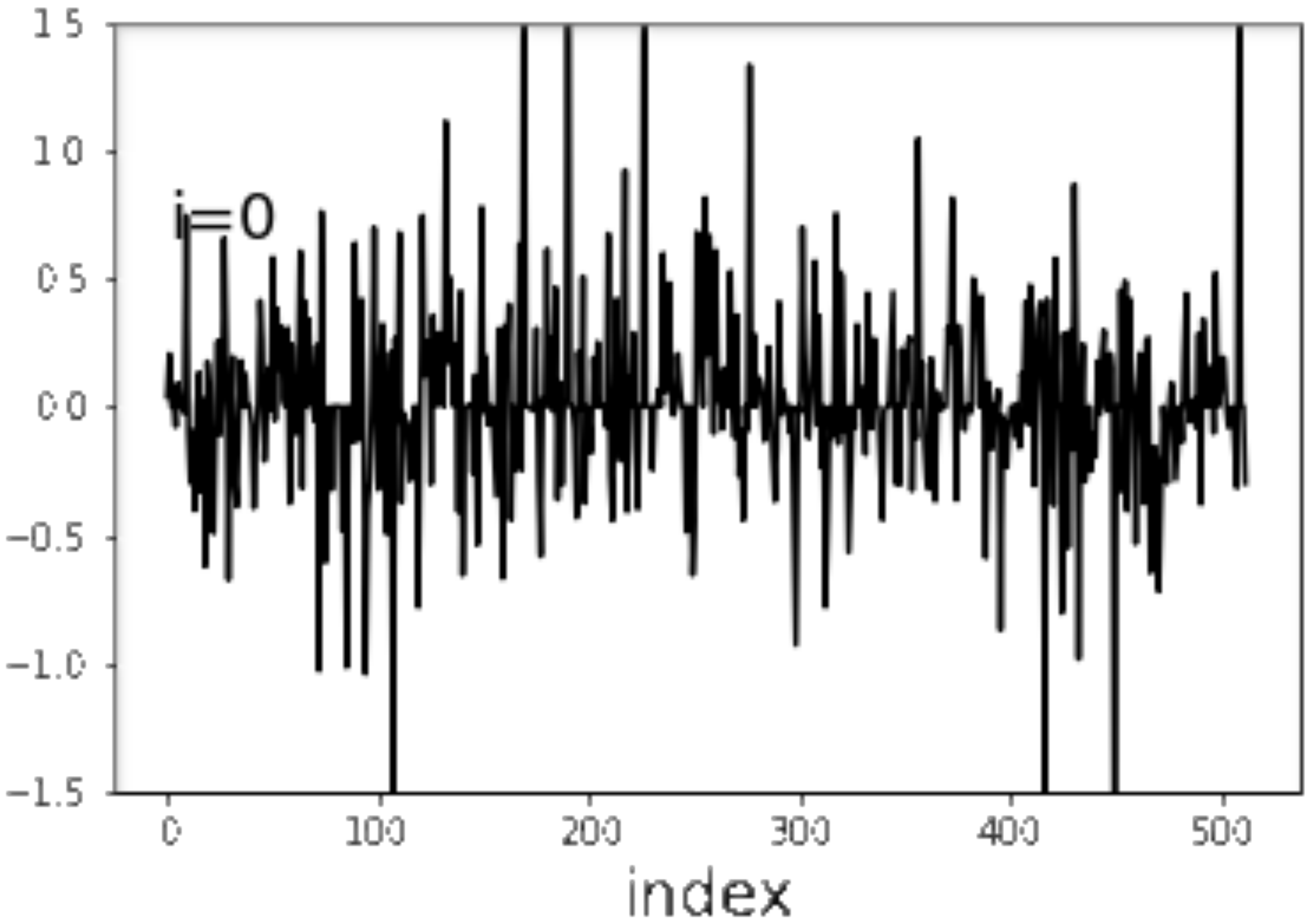
$$s_{t+1} := \eta(r_t; \tau)$$ ：近接写像ステップ

学習可能ISTA

```
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
```

スパース信号再現例

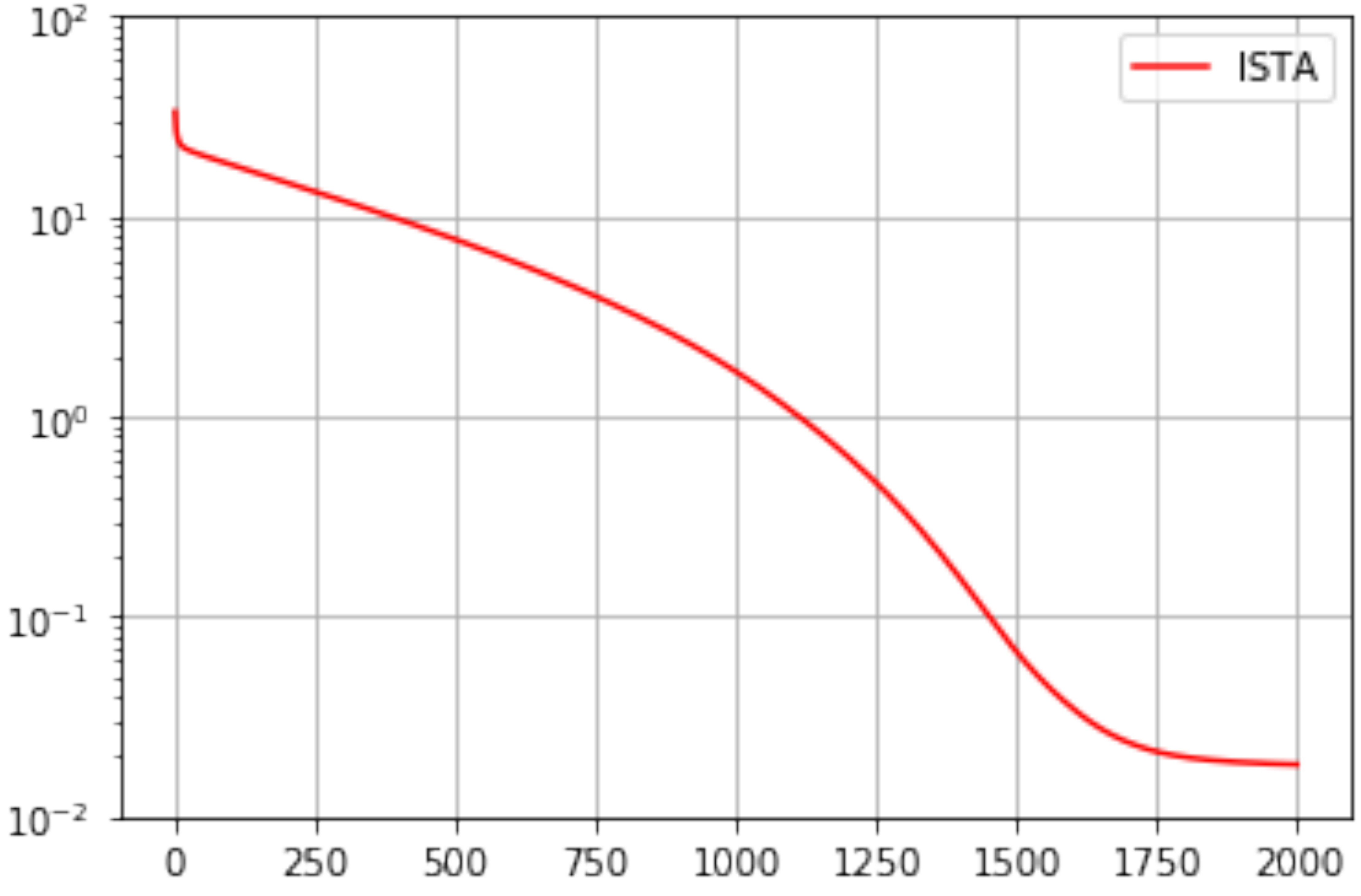
n = 300, m = 150, p = 0.1



二乗誤差の収束の様子

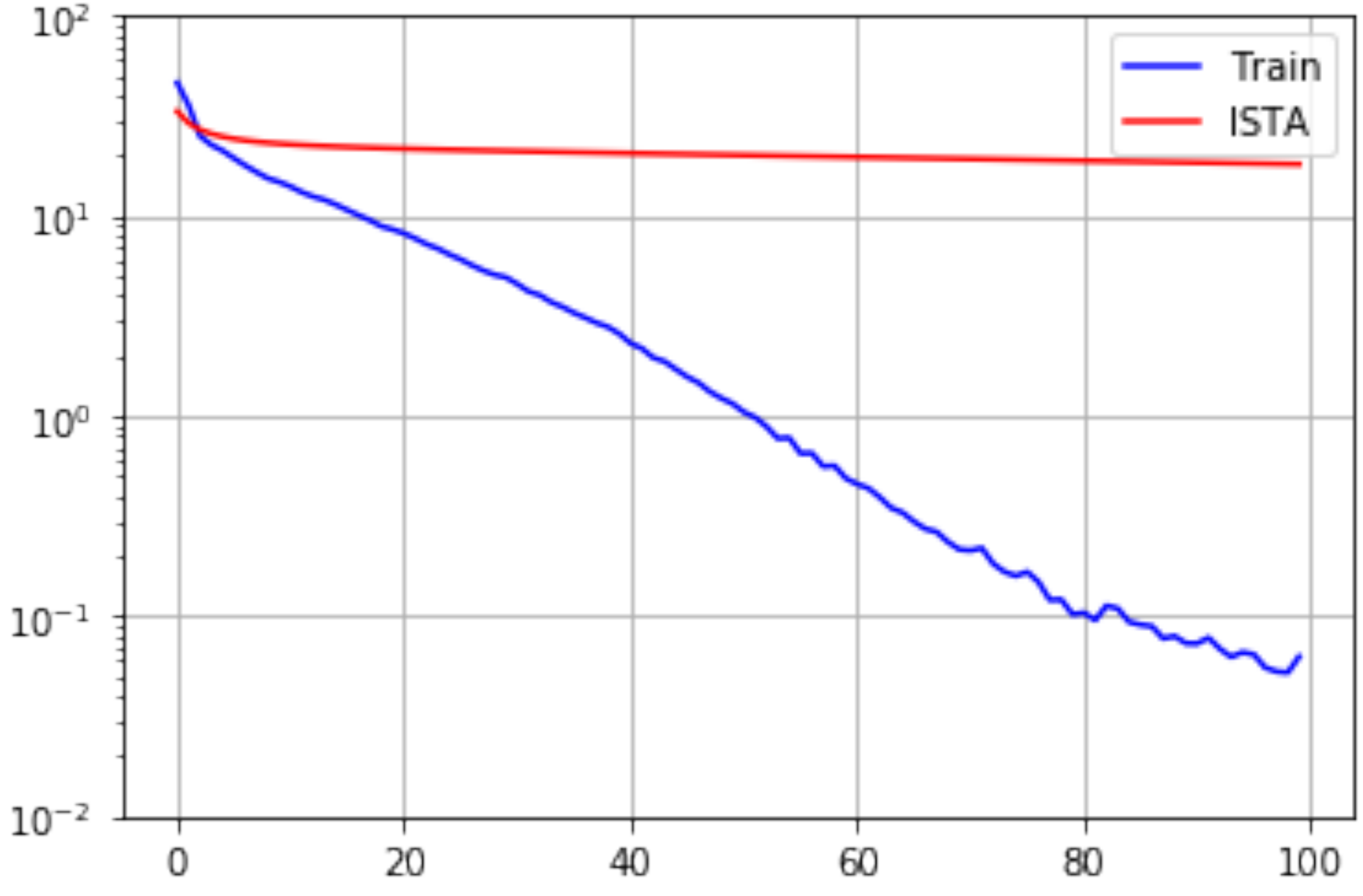
n = 512, m = 256, p = 0.1

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)

実験からの観察

- 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った？
- 学習により収束速度が大幅に向上
- 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

深層展開に基づくアルゴリズム設計フロー(1)

- 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)

深層展開に基づくアルゴリズム設計フロー(2)

SGDに基づく
訓練

学習可能パラメータの
埋め込み
(ステップサイズ・正規化係数)

近接法(proximal method):
近接勾配法 /ADMM/主双対法/
射影勾配法

自動的

アート的

数学的

本講義のまとめ

- 6Gでは高性能な信号処理アルゴリズムが求めれている
- 信号検出問題への深層学習によるアプローチ
- 深層展開による収束加速（勾配法・ISTA)
- 深層展開に基づくアルゴリズム設計フロー

参考文献: 本日の話に関してより詳しい説明やサンプルコードが下記のレポジトリにあります。

https://github.com/wadayama/MIKA2019