

# 浅(くて広い)層学習



少データでお手軽機械学習

越智優真

# 自己紹介

- 千葉大学教育学部附属中学校 3年  
4月から木更津工業高等専門学校情報工学科 1年
- 機械学習
  - Kaggle, SIGNATE Expert
  - 専らデータサイエンスをしてる
- 量子コンピュータ
  - 未経験
  - 量子ビットって何？からはじまった
  - 数理科学の量子コンピュータ特集を買って読んでる(難しい)

# アピールポイント

世界に一つ！

- 機械学習
  - QBoostとNNの融合
  - スパースモデリング
- 実行速度
  - 並列処理
  - スパースモデリング
- 汎用性
  - 数値なら何でもOK
  - ハミルトニアンを書き換える必要なし
- 実用性
  - 重みの可視化が楽
  - 明示的な特徴抽出可
  - 様々なタスクにおいて高い精度を確認済み



## 作成したモデル(ハミルトニアン)の概要

# アンサンブル学習 とNNの融合

- NNっぽさ
  - 疑似特徴抽出
  - 拡張可能
- アンサンブル学習っぽさ
  - QBoostの応用
  - n層作って平均化



**QBoost**とは (ざっ  
くり)

$$H(w) = \sum_{s=1}^S \left( \frac{1}{N} \sum_{i=1}^N w_i h_i(x_{si}) - y_s \right)^2 + \lambda \|w\|_0$$

## QBoost

- アンサンブル機械学習
- 2乗誤差
- 正則化項の存在
- QUBO形式      入力と出力が離散値でつながっているため

# 表現力が低い

(汎用性が低い)



汎用性が低いことに対する

# 解決策



# 連続値の表現

重みを平均すれば連続に近くなる

- 理想は小数
- 現実は $\{-1, 1\}$   $\leftrightarrow$  Ising Poly

## ミニバッチ学習





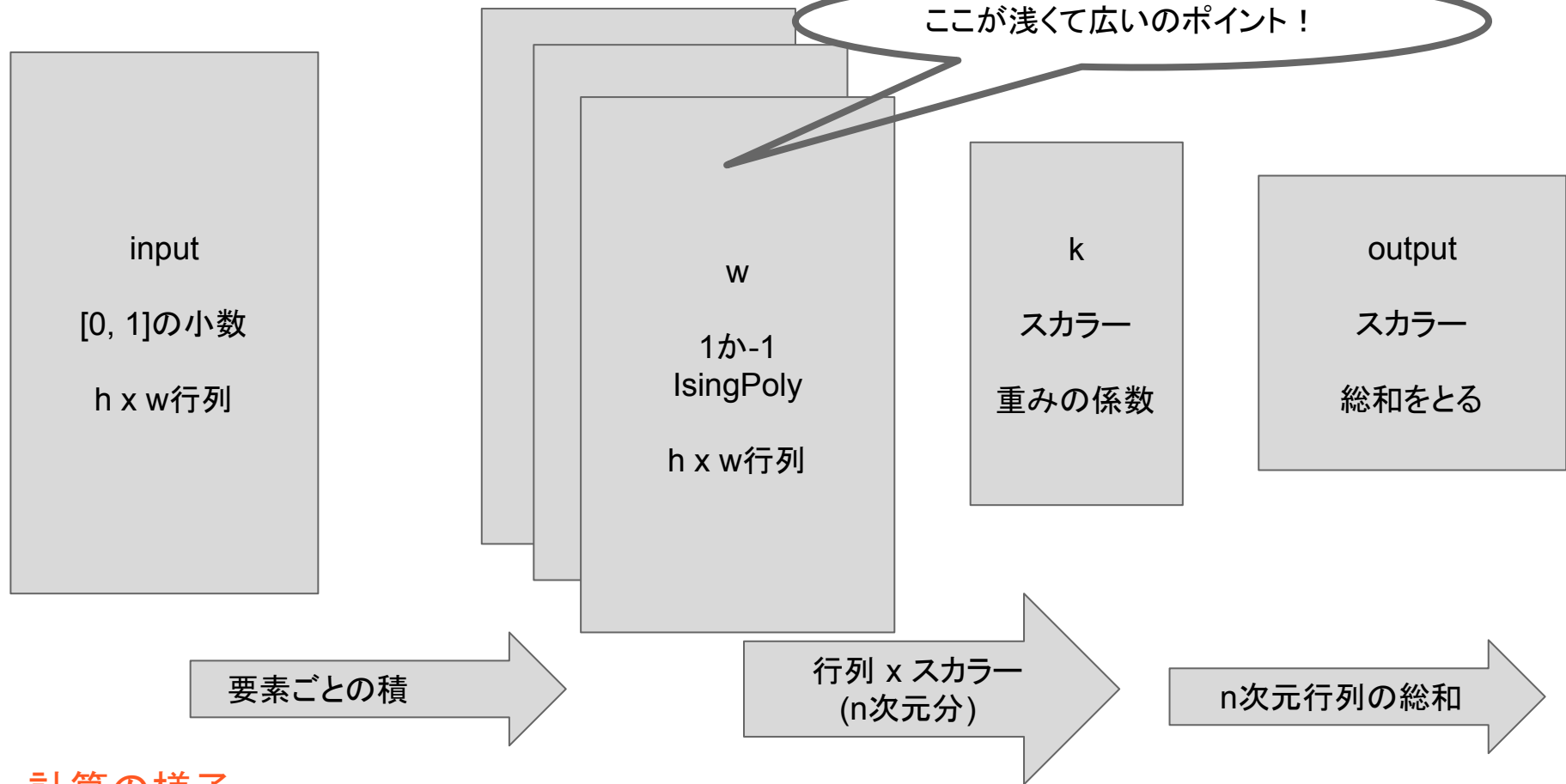
## 表現力

- 特徴抽出をしたい
- 二値では表現不可

# 重みを多層化

QBoostを $n$ 個持ってきて $n$ 層と言ってるのと同義





計算の様子

つまりアンサンブルの  
アンサンブルです

## 表現力向上

- 元々は入力も二値にしている
- 現実には小数、整数お構いなし
- QBoostでは入力を $\{-1, 1\}$ に変換している
  - 表現力低下

入力を  
正規化だ

# データセット

- 量子ビットとの積をとるので0以外の数が必要
  - 0の特徴量があると学習不可
- 負が存在しないので総和で $[0, 1]$ に計算させるのは難しい
  - ならば負を用意→→→→→→→→

0を-1に補完

Binary Poly  
→ Ising Poly

1. 抽象、汎用的なコーディング
2. スパースモデリング
3. 並列処理
4. 簡単な重みの可視化

# 工夫点

汎用、抽象的な  
コーディング

## 意識したこと

- 拡張性
- 汎用性
- 楽なコーディング  
& デバッグ
- 楽な実験管理

# 抽象かつ 汎用的な コーディング





# 例: PyTorch Dataset & DataLoader

- ミニバッチ御用
  - iterableなので
- 前処理、augmentationもできる
- 柔軟
- 書きやすい

深層学習用のライブラリでも使える



```
class MyDataset(Dataset):
    def __init__(self, data, label):
        self.data = data
        self.label = label

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx, :],
        self.label[idx]
ds = MyDataset(df, labels)
dl = DataLoader(ds, batch_size=bs)
```

## 例: Config管理

- コードは抽象、configは具体
  - スッキリ
  - ライブラリとしても機能
- 読み/使い やすい
  - 汎用的

```
base:
  api_key_path: "token/token.json"
  train_path: "../data/train_400.csv"
  seed: 67

dataset:
  img_size: 20
  features: 400 # 特徴量の数
  val_size: 300 # validationに使うデータ数
  target: 5 # ラベル

model:
  timeout: 3000 # ms 計算時間
  # 訓練に使うデータは(batch_size * n_iter)
  batch_size: 20 # バッチサイズ
  n_iter: 5 # ループ数
  l: 2 # 正則化項
  each_weight: 1 # 重み係数
  length_weight: 3 # 重みの層の数
  multiprocessing: true
```

スパースモデリング



少データで高精度、  
狙えます

# お気持ち

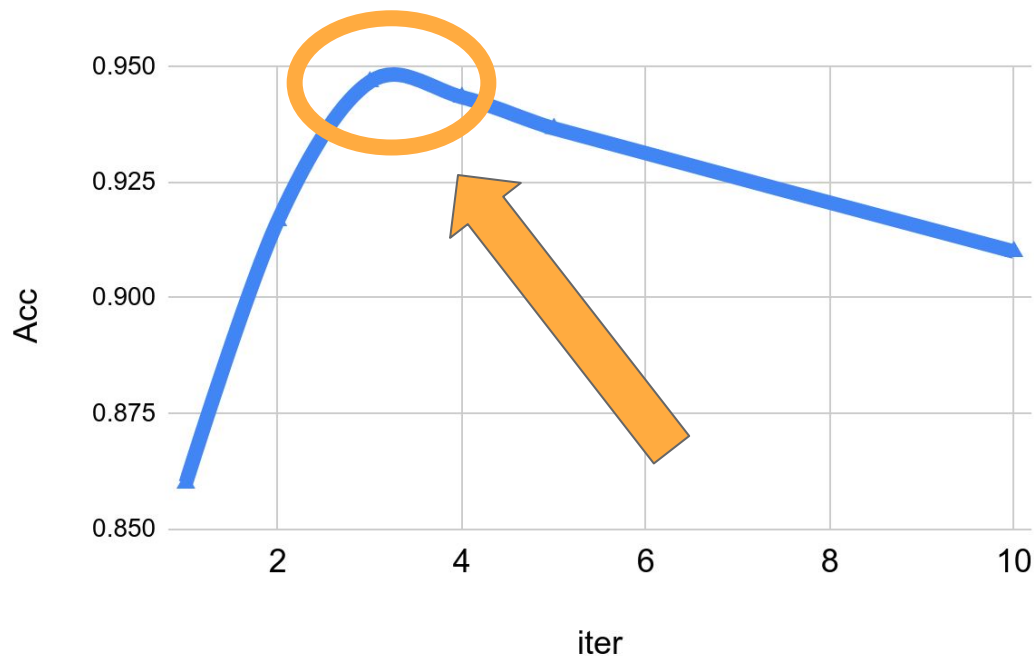
シンプルに。よりシンプルに、  
本質を抽出せよ。

同様のデータを説明する仮説が二つ  
ある場合、より単純な方の仮説を選択  
せよ。

オッカムの剃刀

---

## Batch Size = 10で実験



Bestはたったの  
**30~40**データ

MNISTの0ラベル 検証用データ300枚  
重み係数1で3層 iterのみ変更させた データ数は bs x iter

# 並列処理



# 並列処理

- プロセス番号: CPUのコア数が最大
- 前処理: solverの定義
- メイン: 計算(solve)

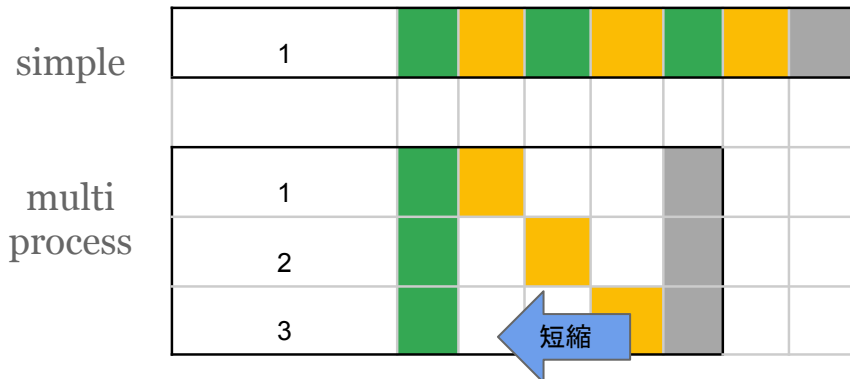
solver定義はローカル実行

→ 非同期処理可

solveはクラウド実行

→ 同期処理

MNISTだと緑1ブロックに  
5~30秒かかる(データサイズと相関有)



## 時間がかかる 前処理が爆速に

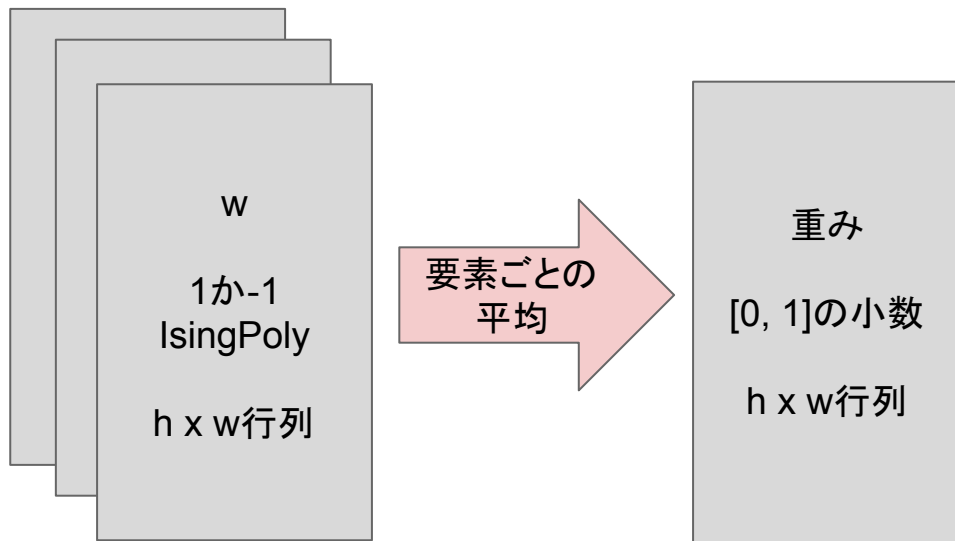


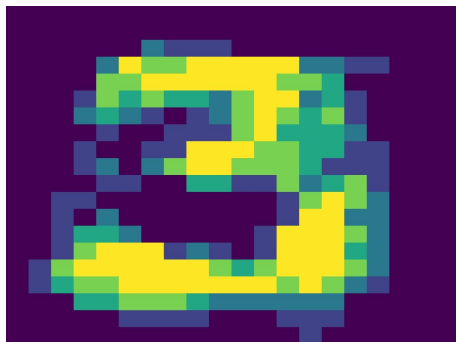
# 簡単な重みの可視化



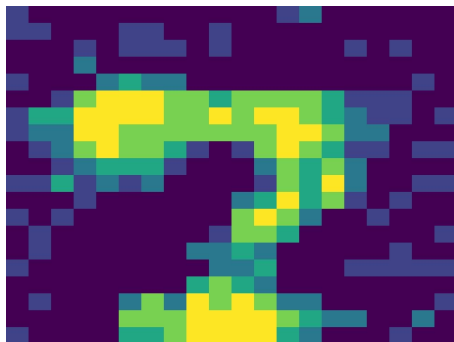
# 方法

1. 学習
2. 学習済みの重みを取得
3. 要素ごとの平均
4. 以上

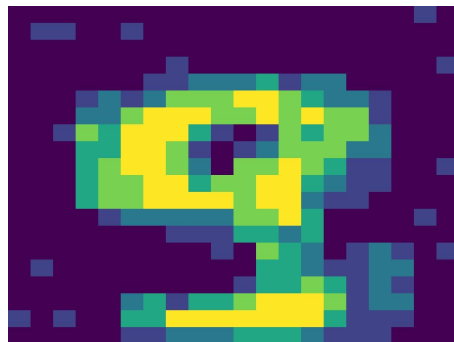




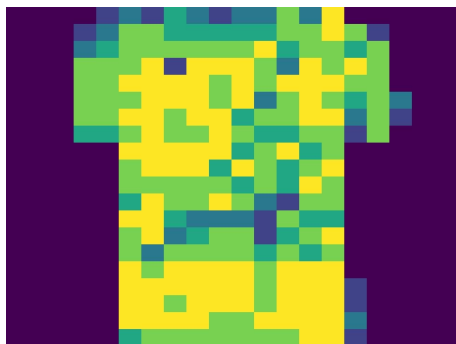
3



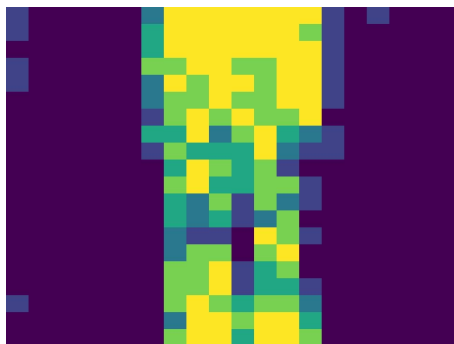
7



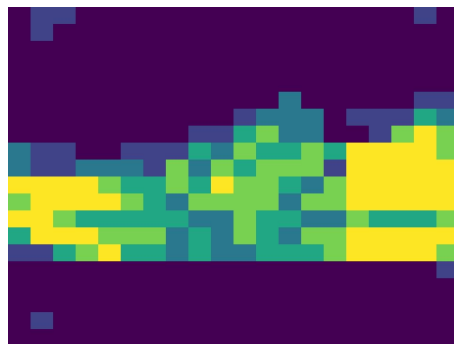
9



Tシャツ



ズボン



スニーカー

重み可視化の例

# 役に立つこと

- 人に説明できる
  - DNNの欠点は説明が難しいこと
  - 実用的
- 何で 上手くいく / いかない かがわかる
  - フィードバック可
  - 指標+重みをみてモデルを評価できる



どんだけすごい？

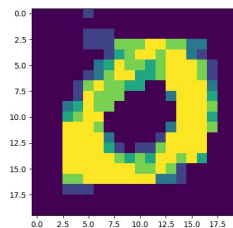
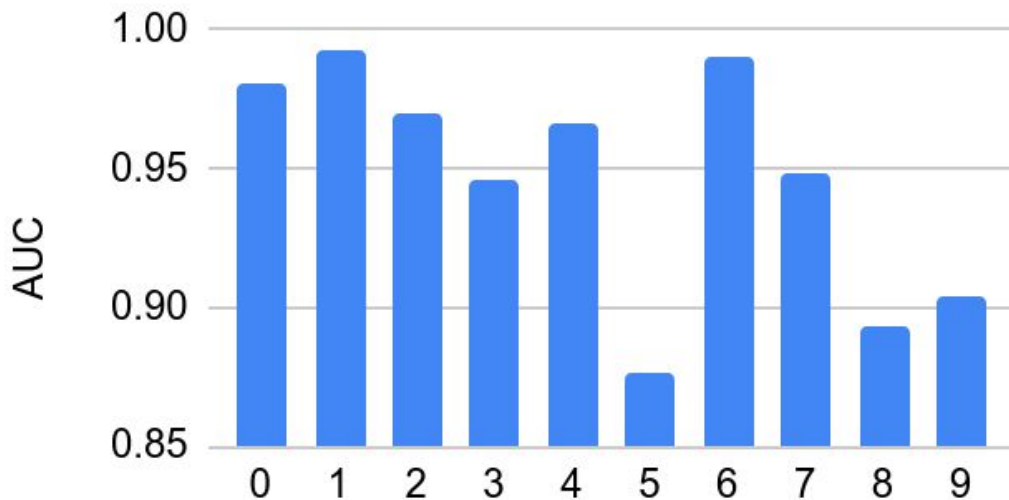
# 0.946

MNISTのサンプルのAUC score

# サンプル実装

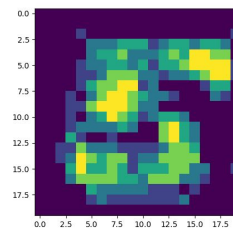
- データ
  - MNIST
  - 20 x 20 pixels
  - 訓練100枚
  - 検証300枚
- モデル(QBoost)
  - 重み: 3層
  - 重みの係数: 1
  - 正則化係数 $\lambda$ : 2
- 結果
  - ラベルごとの平均AUC: 0.946
  - シンプルな数字程良い性能

## 申し分ない性能



Label

0と5の  
重みを可視化したもの



# タスクごとの**AUC** 学習は**150**データのみ

- MNIST
  - 手書き数字画像、400 pixels
  - **0.946**
- Fashion-MNIST
  - 服や靴の画像、400 pixels
  - **0.946**
- EMNIST
  - 手書き英文字画像、400 pixels
  - **0.865**
- otto
  - Kaggle多クラス分類コンペ 特徴量 93個のテーブルデータ
  - **0.879**

どんなデータでも  
**OK**





ハミルトニアンを  
いじる必要なし

お手軽**AI**爆誕！

# 解決

- 幅広いデータを受け入れる
  - 0以外の数値なら何でも
- 並列処理
  - 環境にもよるが最高で5倍高速化
- 多層化
  - 層の数と精度の関係
    - データが多い: 相関高
    - データが少ない: 相関低
- スパースモデリング
  - うまくいった
  - 層は多すぎると過学習しやすい

# これから

- 自動特徴抽出
  - $n$ 次多項式
  - 畳み込み
- 小さなバグ
  - 重みを正規化するときにInf, NaNになることがたまにある
- パラメータのチューニング
  - NNの勾配降下法っぽいことをしてうまくやりたい
- 異常検知系タスク
  - 特徴抽出次第だが期待大

# 以上です

ありがとうございました

これより後ろに補足を  
書いてあるので時間があれば  
目を通してもらえればと  
思います！

---

$$H(w) = \sum_{s=1}^S \left( \frac{1}{N} \sum_{i=1}^N \left( \sum_{l=1}^L k w_{li} \right) x_{si} - y_s \right)^2 + \lambda \sum_{l=1}^L \sum_{i=1}^N w_{li}$$

- データ数 $S$ 、特徴量数 $N$ の入力 $x$ 、出力 $y$
- 重みは係数 $k$ 、 $L$ 層の $w$
- 正則化係数 $\lambda$

$$w_{li} \in \{-1, 1\}$$

$$y_s \in \{0, 1\}$$

$$x_{si} \in [0, 1]$$

$$\lambda \in \mathbb{R}$$

作成したハミルトニアン

# 自作データセットで 試す方法

1. データのパスを記載
2. カラムを指定して説明変数と従属変数にわけ  
る
3. (必要ならば)  
前処理等を行う
4. パラメータを設定
5. 実行！

```
base:  
  train_path: "ここ.csv"
```

```
# 特徴量、ラベル  
train_ds = MyDataset(train[:, 1:], train[:, 0])  
valid_ds = MyDataset(val[:, 1:], val[:, 0])
```

```
model:  
  timeout: 3000 # ms 計算時間  
  # 訓練に使うデータは(batch_size * n_iter)個  
  batch_size: 20 # バッチサイズ  
  n_iter: 5 # ループ数  
  l: 2 # 正則化項  
  each_weight: 1 # 重み係数  
  length_weight: 3 # 重みの層の数  
  multiprocessing: true:], val[:, 0])
```

正則化項

## 正則化風項

- 重みの和
- -1と1のバランスを良くする

思いつきだがノイズが軽減され、  
スコアも上がった

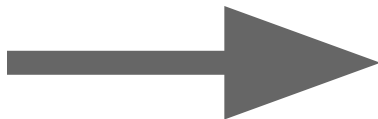
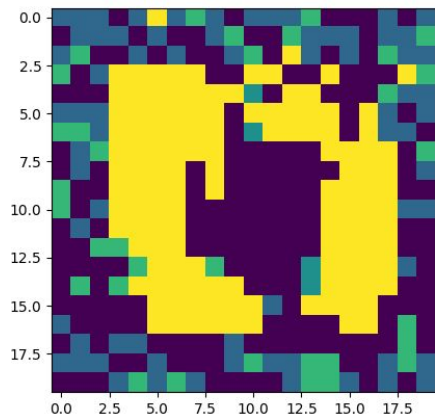
$$\lambda \sum_{l=1}^L \sum_{i=1}^N w_{li}$$

# 正則化風項の力

- 重みの和で表される
- 係数 $\lambda$ が大きいほど1の重みは少なくなる

下はMNISTの0のラベルを $\lambda$ のみを変えて学習させた後の重みを可視化したもの

$\lambda = 0$  score 0.87



$\lambda = 2$  score 0.95

