

2018年7月25日更新

Exercises in Computer-Aided Problem Solving

13. 機械学習II

Machine learning II



東北大学 大学院工学研究科
嶋田 慶太
shimada@m.tohoku.ac.jp



TOHOKU
UNIVERSITY



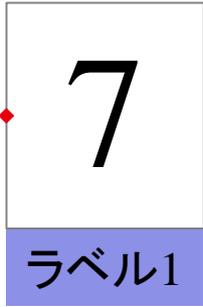
- ニューラルネットワーク(ディープラーニング)
- データの標準化
- ニューラルネットワークのトレーニング



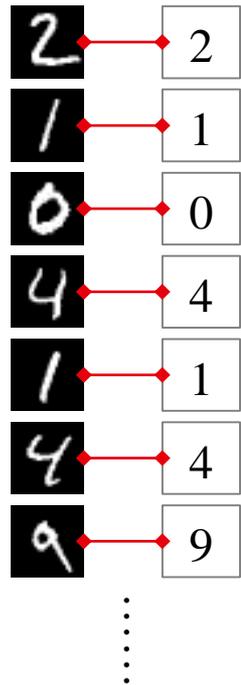
今回の課題(実は前回と同じ)



28×28次元の要素
▶ 数値が色の濃さに対応



"7"という数値

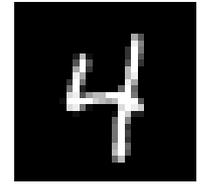


ベクトルとラベルの対応を
「学習」



0 は 0, 1 は 1, 5 は 5...

テストとして



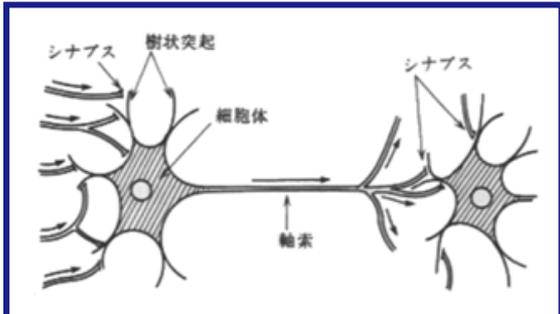
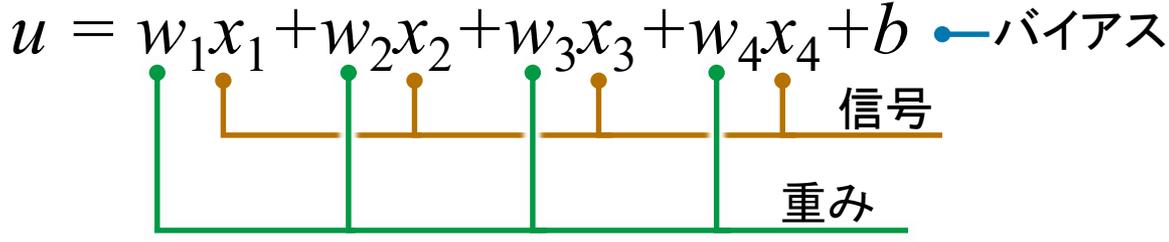
は何か？
について機械が
答える

なおこのような学習法を
教師あり学習
という



ニューラルネットワーク(略称: NN)

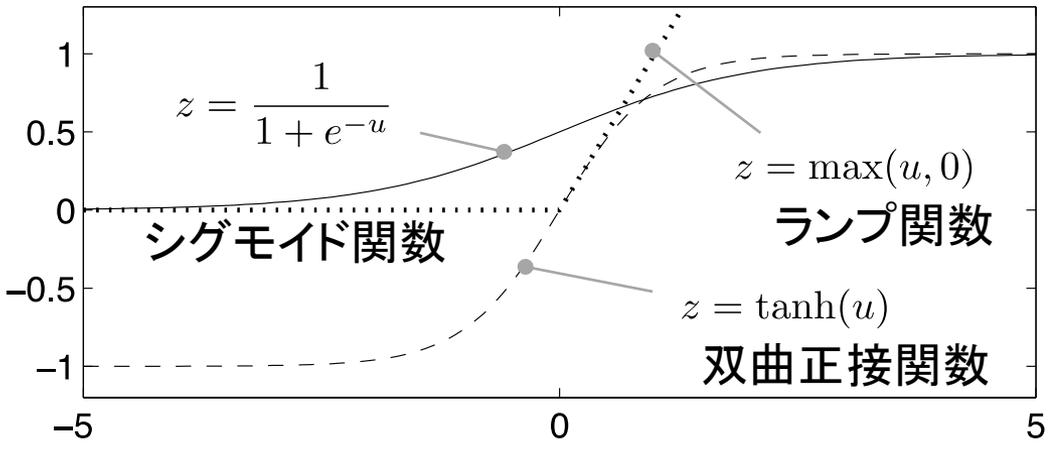
▶ ユニット (Unit) 複数の入力信号 x_i に重み w_i を掛けて足し合わせる



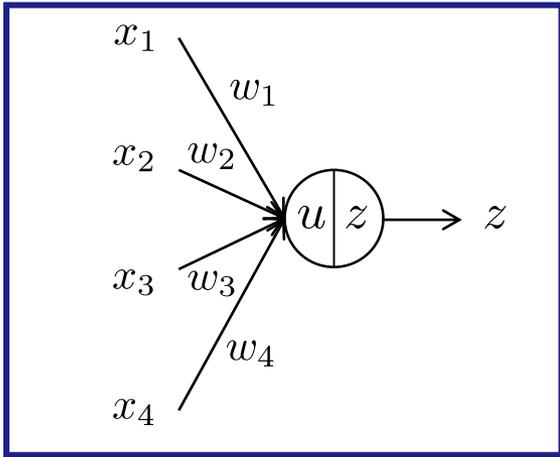
神経細胞

▶ 活性化関数 (Activation function)

$z = f(u)$ ユニットの出力を調整する関数



数理モデル化

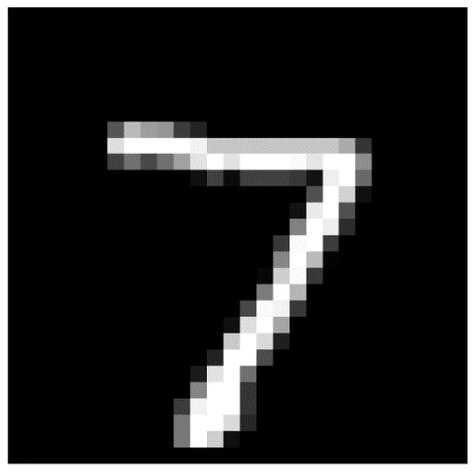


神経細胞

活性化関数には上記の3つ以外にもさまざまな提案がなされている



たとえばエッジ検出・モザイク処理



画像という信号

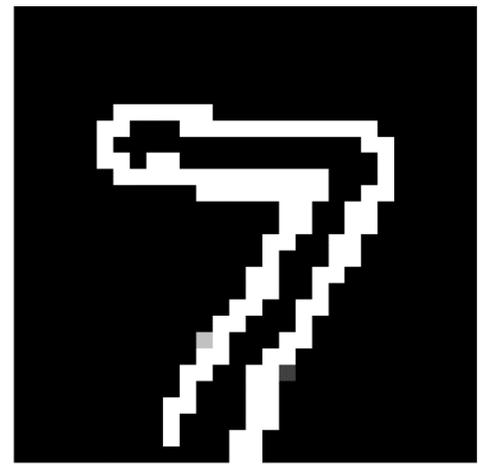
重みW

0	1	0
1	-4	1
0	1	0

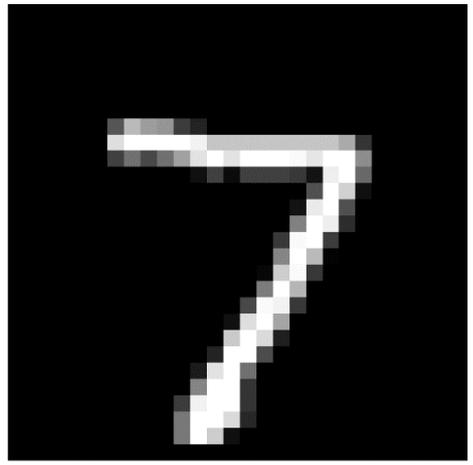
と3×3ピクセルに要素ごと積をして和を取る.

$$z = \max(u, 0)$$

中央の値を z で置き換える



処理後の信号



画像という信号

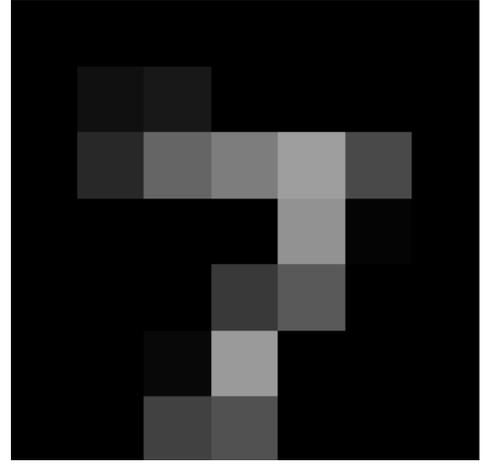
重みW

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

と3×3ピクセルに要素ごと積をして和を取る.

$$z = \max(u, 0)$$

3×3ピクセルを z で置き換える



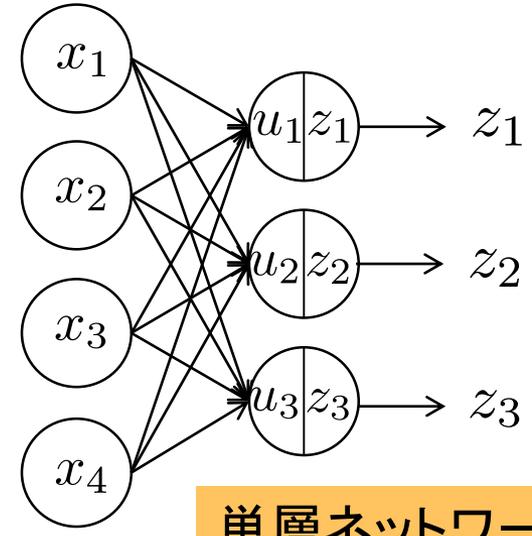
処理後の信号



単層ネットワーク

複数のユニットによる単層の構成

入力ベクトル \mathbf{x} と出力ベクトル \mathbf{z} の関係:



単層ネットワーク

$$u_j = \sum_{i=1}^I w_{ji}x_i + b_j$$

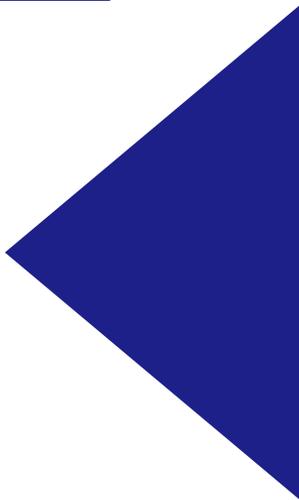
$$z_j = f(u_j)$$

書き下し

$$\mathbf{u} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{z} = \mathbf{f}(\mathbf{u})$$

ベクトル表現



$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_J \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_J \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_J \end{bmatrix},$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & \cdots & w_{1I} \\ \vdots & \ddots & \vdots \\ w_{J1} & \cdots & w_{JI} \end{bmatrix}, \quad \mathbf{f}(\mathbf{u}) = \begin{bmatrix} f(u_1) \\ \vdots \\ f(u_J) \end{bmatrix}$$



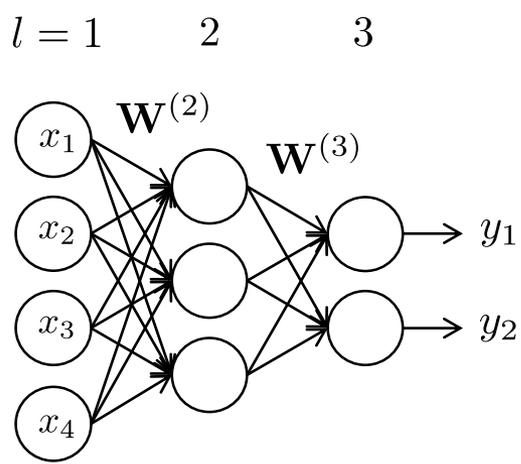
多層ネットワーク

単層ネットワークの積み重ね ▶ 多層ネット(フィードフォワードネットワーク)

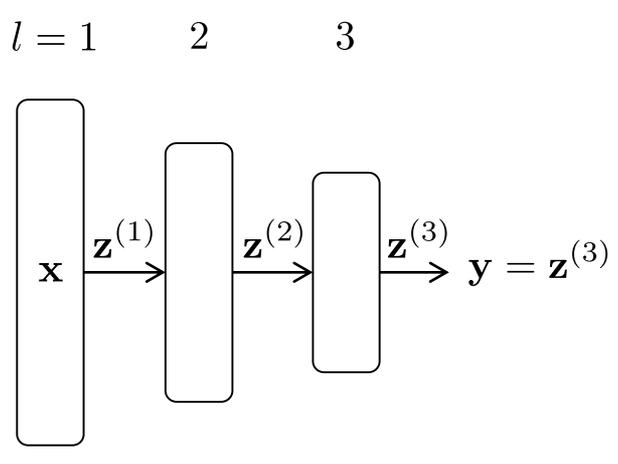
1 st (入力) 層	$\mathbf{x} \equiv \mathbf{z}^{(1)}$
------------------------	--------------------------------------

l^{th} 層から $(l+1)^{\text{th}}$ 層へ	$\mathbf{u}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{z}^{(l)} + \mathbf{b}^{(l+1)}$ $\mathbf{z}^{(l+1)} = \mathbf{f}(\mathbf{u}^{(l+1)})$
---	--

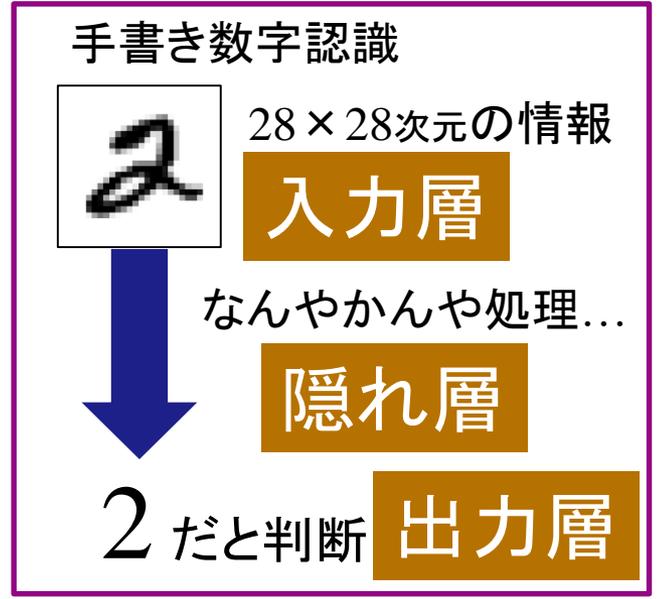
L^{th} (出力) 層	$\mathbf{y} \equiv \mathbf{z}^{(L)}$
------------------------	--------------------------------------



要素ごとのイメージ図



ベクトル的なイメージ図





出力層の条件

出力層のユニット ▶ 分類数(クラス)と同数が必要)

例 手書き数字認識なら0,1,2...9 あるので, 10クラスであり, 10ユニット必要

出力 y ▶ クラスの確率(もしくは尤度) $p(C_k | \mathbf{x}) = y_k = z_k^{(L)}$

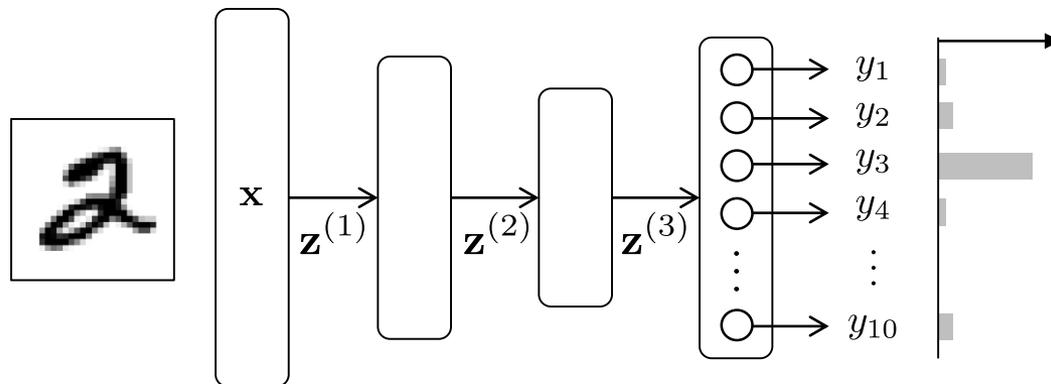
活性化関数: シグモイド関数やソフトマックス関数

◀ 差を強調する関数

エンコード(クラス分け) ▶ クラス数と同数の要素を持つベクトル \mathbf{d} を用いる

クラス k ▶ 第 k 要素のみ1, ほかは0 one-hot/one-of-K と呼ぶ

$$\mathbf{d} = [d_1, d_2, \dots, d_K]$$





フィードフォワードネットのトレーニングと損失

サンプルセット

$$\mathcal{S} = \{(\mathbf{x}_1, \mathbf{d}_1), \dots, (\mathbf{x}_N, \mathbf{d}_N)\}$$

入力 \mathbf{x}_i と 目標 (正解を表す one-hot ベクトル) \mathbf{d}_i

例えば
手書き数字と
正解を表すベクトル

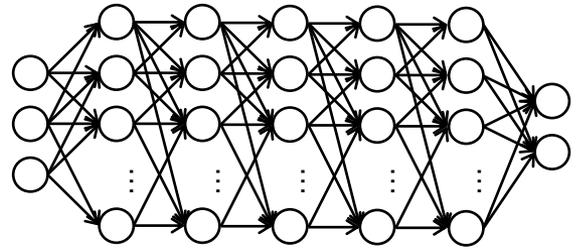


$$\{0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0\}^T$$

※0と対応するベクトルが1 0 0 ... のため、
2に対応するベクトルは3個目の要素が1

ネットワーク

\mathbf{x}_i
入力



\mathbf{y}_i
出力



\mathbf{d}_i
目標

重みとバイアス $\mathbf{w} = \{(\mathbf{W}^{(2)}, \mathbf{b}^{(2)}), \dots, (\mathbf{W}^{(L)}, \mathbf{b}^{(L)})\}$

$E(\mathbf{w}, \mathcal{S})$ 損失 \mathbf{y} と \mathbf{d} の齟齬

▶ 損失を最小化するように重み \mathbf{w} を決定すればよい

$$\min_{\mathbf{w}} E(\mathbf{w}, \mathcal{S})$$

つまり、入力 \mathbf{x} に対して出力 \mathbf{y} が狙った値 \mathbf{d} になるように「ネットワークを鍛える」



実践編



ソフトウェアのダウンロード

(1) 授業のページから [DeepLearnToolbox.zip](https://github.com/rasmusbergpalm/DeepLearnToolbox) をダウンロード

▼このソフトウェアの詳細はここを読んで。▼

<https://github.com/rasmusbergpalm/DeepLearnToolbox>

ものすごくネガティブな
作者のコメントが読める...

(2) zipを解凍する。

使用時はDeepLearnToolboxの2つのフォルダにパスを通す。

```
>> addpath('DeepLearnToolbox/NN');  
>> addpath('DeepLearnToolbox/util');
```



上記の場合、作業フォルダにDeepLearnToolboxというフォルダを置く必要がある。
別の場所にフォルダを置いた場合には絶対パスで記述すればよい。

例: CドライブのOctave ▶ Octave-4.4.0 にDeep~ を置いた場合

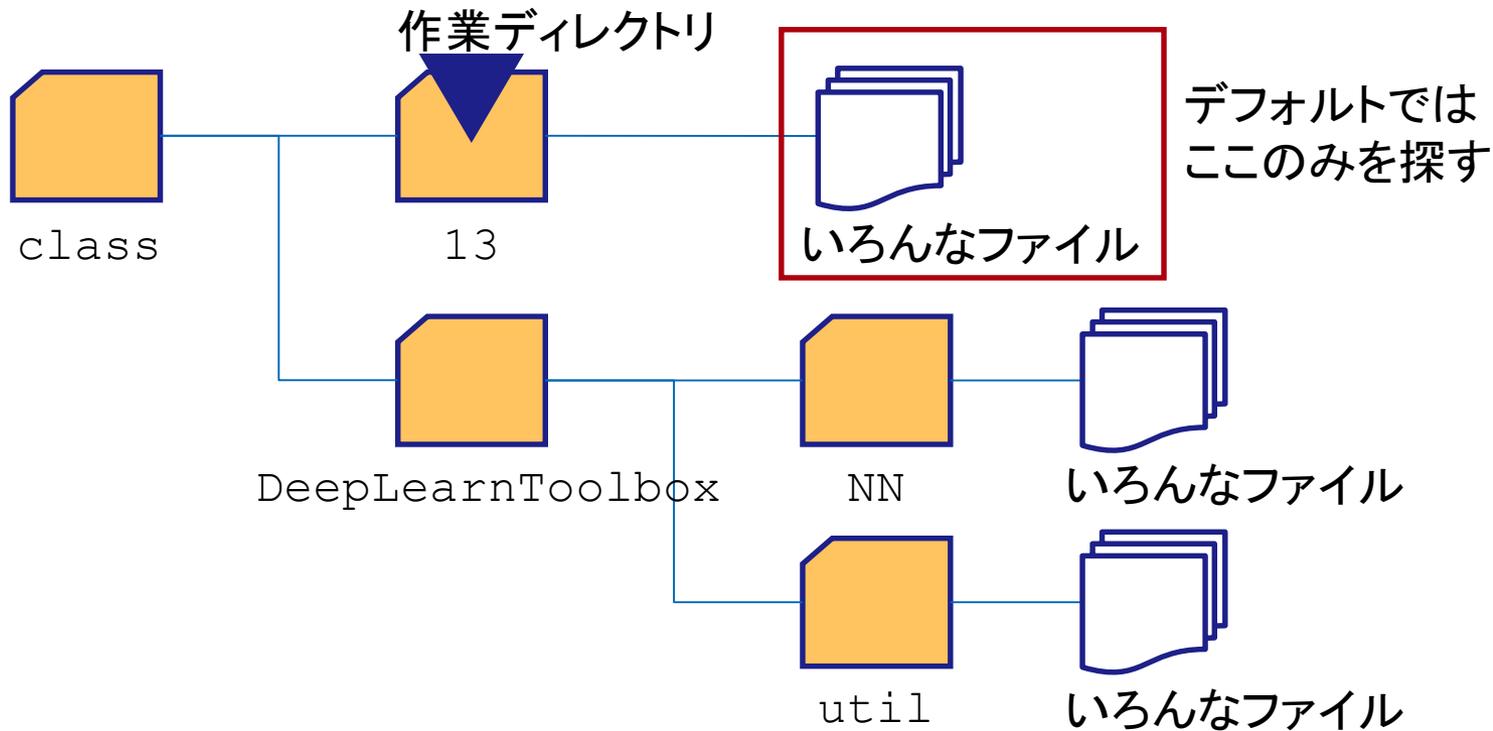
```
>> addpath('C:/Octave/Octave-4.4.0/DeepLearnToolbox/NN')  
>> addpath('C:/Octave/Octave-  
4.4.0/DeepLearnToolbox/util')
```



そもそもaddpathって？

基本的にOctaveでは作業ディレクトリのみ探索する。
(ほとんどのプログラム言語でもそうだけど)

addpath: 探索パスを増やす



```
>> addpath(' ../DeepLearnToolbox/NN') でNNのディレクトリも調べてくれる
```



MNIST 手書き数字認識

前回の授業 (Support Vector Machine): 10,000個のデータのみ使用

今回の授業: 60,000個のデータ▶トレーニング用

10,000個のデータ▶テスト用(トレーニングの成果を検証)

▼この通り打ち込む...面倒なので▶ファイル

1

```
fid=fopen('t10k-images-idx3-ubyte','r','b');
fread(fid,4,'int32')
test_img=fread(fid,[28*28,10000],'uint8');
test_img=test_img';
fclose(fid);
```

2

```
fid=fopen('t10k-labels-idx1-ubyte','r','b');
fread(fid,2,'int32')
test_lbl=fread(fid,10000,'uint8');
fclose(fid);
```

3

```
fid=fopen('train-images-idx3-ubyte','r','b');
fread(fid,4,'int32')
train_img=fread(fid,[28*28,60000],'uint8');
train_img=train_img';
fclose(fid);
```

4

```
fid=fopen('train-labels-idx1-ubyte','r','b');
fread(fid,2,'int32')
train_lbl=fread(fid,60000,'uint8');
fclose(fid);
```

一応説明:各ブロックとも同様の4-5行構成

- ▶ `fid = fopen(~~);`
ファイルを開き, `fid`にIDを記録.
'r': 読み込み専用
'b': ビッグエンディアン(ここでは深追いしない)
- ▶ `fread(fid, ~~);`
`fread(fid, ~~)`でIDが`fid`のファイルを読む.
ファイルの最初に説明があるが今回は読み飛ばす
表示したくなければここにもセミコロンを打つ.
- ▶ `var_name = fread(fid,~~);`
今度は`fread`で読んだ内容を変数に格納
なお'unit8'や'int32'は変数の型名
- ▶ 画像の場合はここで転置して格納し直す
- ▶ `fclose(fid);`
開けたファイルを閉じる.

- 1: テスト用画像を`test_img`に格納
- 2: テスト用ラベルを`test_lbl`に格納
- 3: トレーニング用画像を`train_img`に格納
- 4: トレーニング用ラベルを`train_lbl`に格納



データの規格化 (1/2)

▶ データの標準化 自然なデータ: 分布に傾向があり, 学習には適さないことが多い

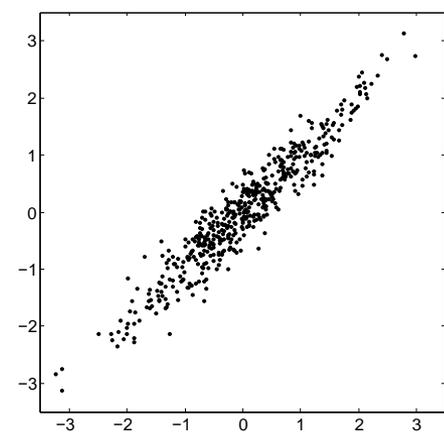
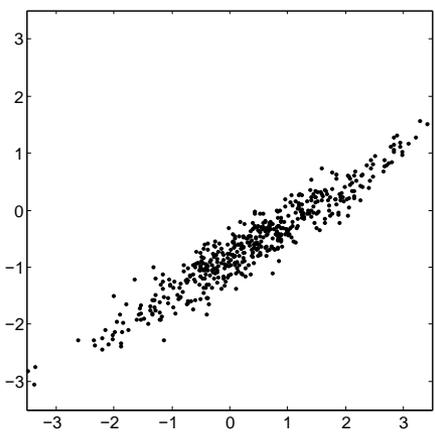
平均0 分散1 となるよう線形変換 ▶ NNやSVMに通常役立つ

j 列のサンプル:

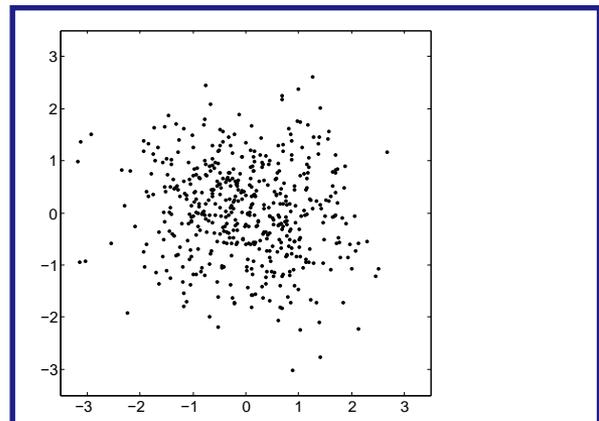
$$\widetilde{\{x_{i,j}\}} = \frac{\{x_{i,j} - \mu_j\}}{\sigma_j}$$

平均 $\mu_j = \frac{\{1\}^T \{x_{i,j}\}}{N}$

標準偏差 $\sigma_j = \sqrt{\frac{\{x_i - \mu_j\}^T \{x_i - \mu_j\}}{N}}$



標準化(座標軸の中心とスケールの変更)



類似の操作に
共分散行列の対角化を用いて
直交化して標準化する

白色化というものもある。
$$\{x_{i,j} - \mu_j\} V W^{-\frac{1}{2}}$$

V: 固有ベクトルを並べた行列
W: 固有値の平方根の逆数



データの規格化 (2/2)

トレーニング用サンプル \mathbf{x}_{trg} (ここではtrain_img) の平均 μ_{trg} , 標準偏差 σ_{trg} を計算

```
mu = mean(train_img);  
sigma = max(std(train_img, 1), eps);
```

$(\mathbf{x}_{\text{trg}} - \mu_{\text{trg}}) ./ \sigma_{\text{trg}}$ により標準化

```
train_img = (train_img - mu) ./ sigma;
```

テスト用サンプル \mathbf{x}_{test} (ここではtest_img) に対しても μ_{trg} と σ_{trg} を用いて変換

```
test_img = (test_img - mu) ./ sigma;
```

◀ eps: 計算機イプシロンと呼ばれる定数.
(ざっくりいうと計算機の数値の最小区切り)
ここでは分母0を避けるため,
std(~) = 0のときにepsを選ぶ.

 μ_{test} と σ_{test} を計算してはいけない



ラベルをOne-hotベクトルに変換

```

onehot=eye(10,10);
train_d=onehot(train_lbl+1,:);
test_d=onehot(test_lbl+1,:);

```

◀ One-hot ベクトルの準備(単位行列)

◀ ラベルの数値は0~9であるが、
行数は1~10なので、
1ずらして対応づける

train_lbl

5

6行に対応

0

1行に対応

4

5行に対応

1

2行に対応

9

10行に対応

2

3行に対応

:

:

train_d

0 0 0 0 0 1 0 0 0 0

1 0 0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 0 0

0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1

0 0 1 0 0 0 0 0 0 0

lbl では数値なのに対して、
train_dではOne-hotベクトルが格納されている



ネットワークのトレーニング

2層のNN(入力:784要素, 中間層:100ユニット, 出力:10ユニット)

```
nn = nnsetup([784 100 10]);
```

▲
DeeplearnToolbox内の関数 指定された構造でNNを構築

トレーニング用サンプルを用いてネットワークをトレーニング

```
opts.numepochs = 1;          ◀ エポック数: データセットを反復学習する回数  
opts.batchsize = 100;       ◀ バッチサイズ: 重みを更新する周期(小さいほど早い)  
[nn, L] = nntrain(nn, train_img, train_d, opts);
```

▲
DeeplearnToolbox内の関数 NNをトレーニングする



トレーニングしたネットワークのテスト

トレーニングしたネットワークにテスト用サンプルを入れて評価

```
>> pred = nnpredict(nn, test_img);
```

▲
DeeplearnToolbox内の関数 トレーニングしたNNで予想する

クラス番号とラベルのずれを修正(スライド13の逆)

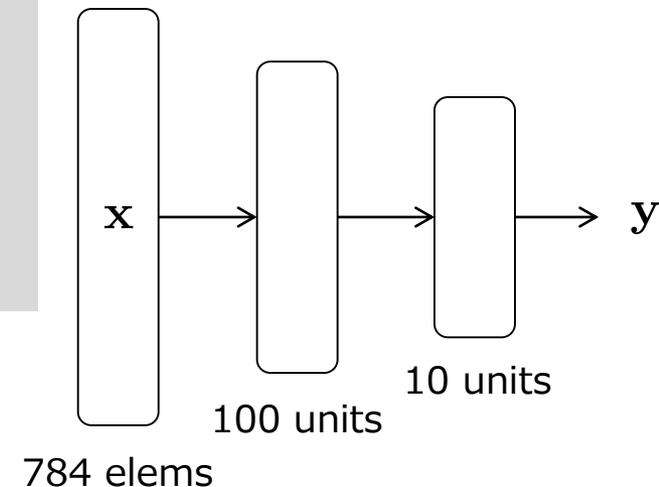
```
>> pred = pred - 1;
```

現状: クラス1に0, クラス2に1, ...クラス10に9という対応

とりあえず1~10番目のテスト結果と実際のラベルを表示

```
>> pred(1:10) ' ◀ 結果(数字画像を見て機械が判別した数字)
ans =
>> 7     2     1     0     4     1     4     9     5     9
test_lbl(1:10) ' ◀ ラベル(答え)
ans =
     7     2     1     0     4     1     4     9     5     9
```

```
>> mean(pred==test_lbl)*100
ans = 92.900 一致度(パーセント表示)
```





ここまでをスクリプトにまとめると

13 load_MNIST_data

```
14 mu = mean(train_img);  
15 sigma = max(std(train_img,1),eps);  
train_img = (train_img - mu) ./ sigma;  
test_img = (test_img - mu) ./ sigma;
```

```
16 onehot=eye(10,10);  
train_d=onehot(train_lbl+1,:);  
test_d=onehot(test_lbl+1,:);
```

```
17 nn = nnsetup([784 100 10]);  
opts.numepochs = 1;  
opts.batchsize = 100;  
[nn, L] = nntrain(nn, train_img, train_d, opts);
```

```
18 pred = nnpredict(nn, test_img);  
pred = pred - 1;  
[pred(1:10), test_lbl(1:10)]  
mean(pred==test_lbl)*100
```

load_MNIST_data.m (データ読み込み部)

```
addpath('C:/Octave/Octave-4.4.0/DeepLearnToolbox/NN');  
addpath('C:/Octave/Octave-4.4.0/DeepLearnToolbox/util');  
  
fid=fopen('t10k-images-idx3-ubyte','r','b');  
fread(fid,4,'int32')  
test_img=fread(fid,[28*28,10000],'uint8');  
test_img=test_img';  
fclose(fid);  
  
fid=fopen('t10k-labels-idx1-ubyte','r','b');  
fread(fid,2,'int32')  
test_lbl=fread(fid,10000,'uint8');  
fclose(fid);  
  
fid=fopen('train-images-idx3-ubyte','r','b');  
fread(fid,4,'int32')  
train_img=fread(fid,[28*28,60000],'uint8');  
train_img=train_img';  
fclose(fid);  
  
fid=fopen('train-labels-idx1-ubyte','r','b');  
fread(fid,2,'int32')  
train_lbl=fread(fid,60000,'uint8');  
fclose(fid);
```

同一ディレクトリにload_MNIST_data.m
および読み込ませるファイルが必要。

▲
対応するスライド番号



実行時の出力に関して

実行初回のwarning

```
warning: function C:/Octave/Octave-4.2.1/DeepLearnToolbox/util¥randp.m shadows a built-in function
warning: called from
  load_MNIST_data at line 4 column 1
  ex13_01 at line 1 column 1
warning: function C:/Octave/Octave-4.2.1/DeepLearnToolbox/util¥zscore.m shadows a core library function
warning: called from
  load_MNIST_data at line 4 column 1
  ex13_01 at line 1 column 1
```

DeepLearnToolbox内で `randp.m` と `zscore.m` という関数が定義されており、これらがビルトイン関数(標準装備)であるポアソン分布に則る乱数 `randp` とコアライブラリ関数(標準装備に近い関数)の `zscore` と干渉する. という警告. 今回はポアソン分布を使いたくならないので構わない.

(ユーザの指示優先, 類例: $\pi = 3$; と代入すると, π は clear するまで円周率ではなく 3 になる)

途中に表示されるこれは▼

```
epoch 1/1. Took 2.5115 seconds. Mini-batch mean squared error on training set is 0.16489;
Full-batch train err = 0.072168
```

▲これはトレーニングでの精度

計算時間やミニバッチ, フルバッチでのトレーニング精度を示す.

テストの精度ではないことに注意.



Exercises 13

トレーニングを繰り返すとき, エポック数を増やすほか,

```
>> [nn, L] = nntrain(nn, train_img, train_d, opts);
```

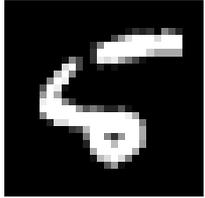
を繰り返し計算させてもよい. また, 初期化は,

```
>> nn = nnsetup([784 100 10]);
```

とコマンドすればよい. これを踏まえて下記の問題に答えよ.

1. 初期化から `nntrain` を10回繰り返した際のトレーニング回数と精度の関係をグラフでしめせ.
2. 構造を `[784 30 30 10]` とする3層NNを構築し, 前の2層NNの場合との正解率を比較せよ.

ヒント ▶ 今回は `for loop` を回さざるを得ない. 鍛えた `nn` を再度鍛えるには... ?
Loopごとに正解率を格納して, プロットできるようにすればいい.



これは何に見える？

個人的には ζ (シグマの異字体) 見えるが、

ラベル上は "5"