13. Machine leaning 機械学習

- Neural networks (deep learning)
- Standardization of data
- Training neural networks

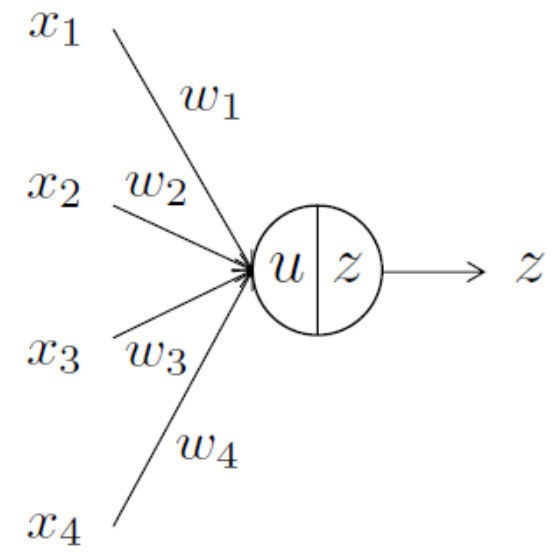
Neural network: Unit and activation functions

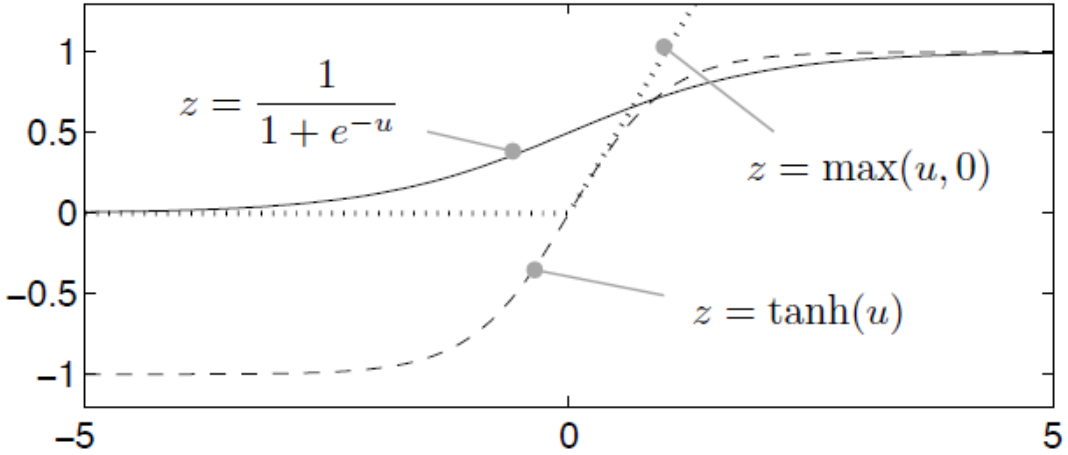
- Unitは複数の入力信号を重みを付けて合計し，
  非線形関数を通して出力する
  - Neuronの簡単なモデル

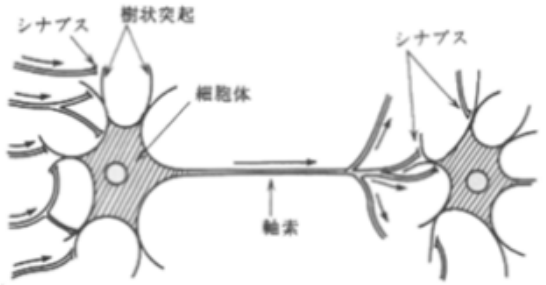
$$u = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b$$

$$z = f(u)$$

  - 関数fはactivation functionと呼ばれる
  - 様々な関数が用いられる





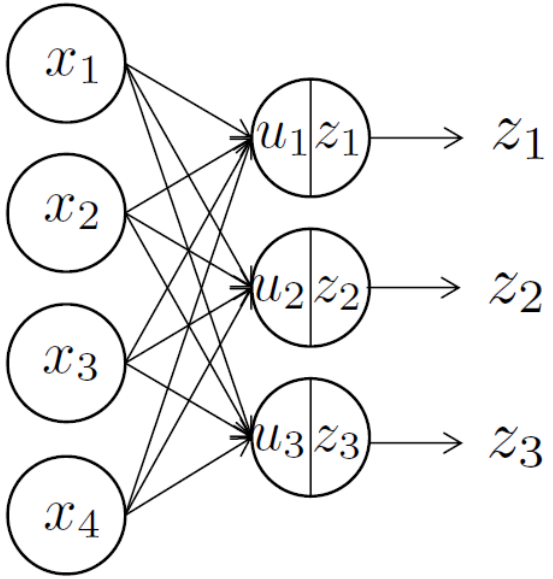


Neural network: single layer net

- 複数のunitから成るlayer（層）を作る
- ベクトル **x** を入力してベクトル **z** として出力する

$$u_j = \sum_{i=1}^{I} w_{ji} x_i + b_j \qquad \text{or} \qquad \mathbf{u} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$z_j = f(u_j) \qquad \mathbf{z} = \mathbf{f}(\mathbf{u})$$



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

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

Neural network: multi-layer net

- Multi layer net=Single-layerを束ねたもの feed-forward networkとも

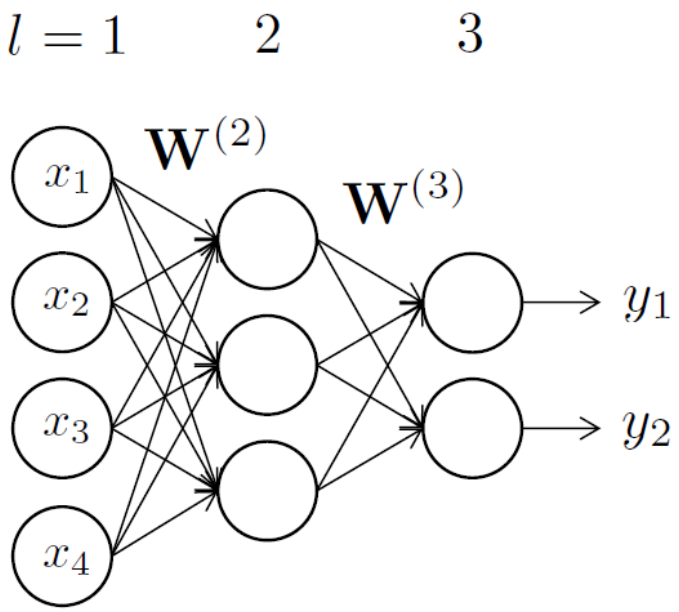
1st (input) layer $\quad \mathbf{x} \equiv \mathbf{z}^{(1)}$

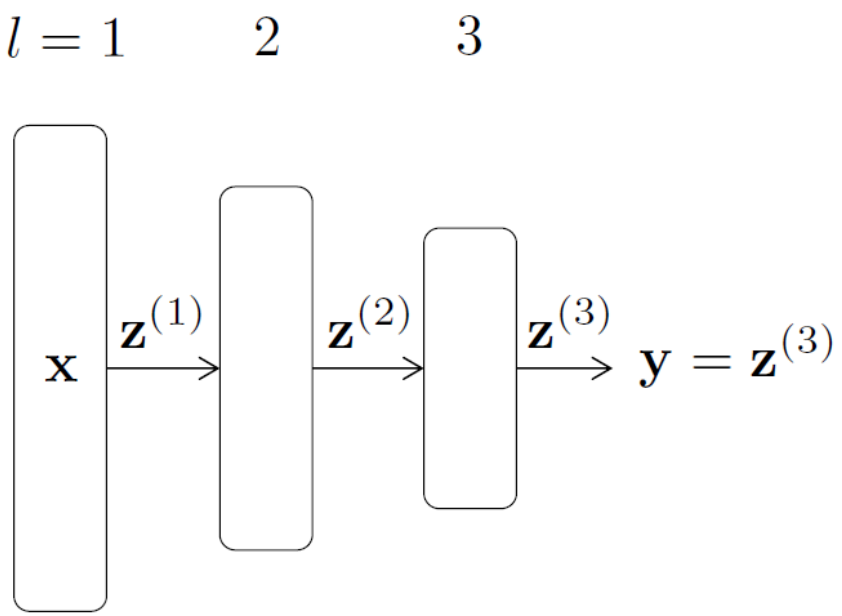
Propagation from lth to (l+1)th layer

$$\mathbf{u}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{z}^{(l)} + \mathbf{b}^{(l+1)}$$

$$\mathbf{z}^{(l+1)} = \mathbf{f}(\mathbf{u}^{(l+1)})$$

Lth (output) layer $\quad \mathbf{y} \equiv \mathbf{z}^{(L)}$





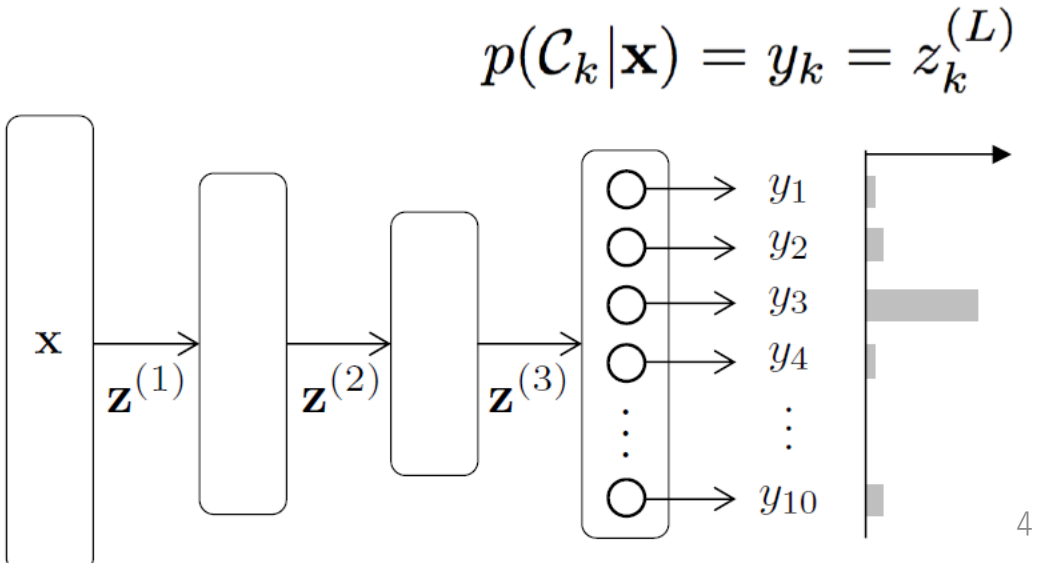
Neural network: Output layer and loss

- Output layer はクラス（分類）の数と同じにする
  k番目のoutputはk番目のクラスの確率
- クラスは長さKのベクトルdでコード化する
  クラスがkの場合はk番目の要素が1でほかは0(one-hotベクトル)
  - 10クラスのMNISTデータに対して
    one-hotベクトルを作る場合は

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

```
>> A=eye(10,10);
>> train_d=A(train_lbl+1,:);
>> test_d=A(test_lbl+1,:);
```

変数にデータを読み込んでから
コマンドを実行する必要がある
7ページを参照

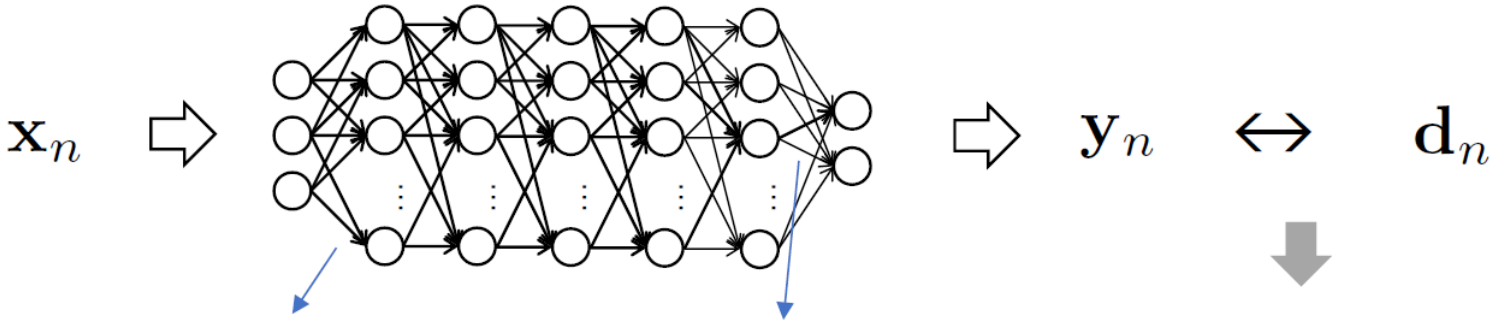


Feed-forward networkのトレーニング

- サンプル；入力**x**とターゲット**d**(入力に対する正しいクラスを示すone-hotベクトル)

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

- このサンプルに対してneural netをトレーニングして，入力**x**に対する出力**y**ができるだけ**d**と近くなることを目指す



Layerの重み　$\mathbf{w} = \{(\mathbf{W}^{(2)}, \mathbf{b}^{(2)}), \cdots, (\mathbf{W}^{(L)}, \mathbf{b}^{(L)})\}$

Loss
(**y**と**d**の差)　$E(\mathbf{w}, \mathcal{S})$

- つまり，これはlossの最小化問題である

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

Software library

- このコースではMATLAB/Octave用のライブラリを用いる

  https://github.com/rasmusbergpalm/DeepLearnToolbox

- 数理情報学演習コースページよりDeepLearnToolbox.zipをダウンロード
- 下のようにタイプ

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

MNIST 手書き文字認識

- 第12回に講義ではSVMsのトレーニングのためにt10k-ファイルの10,000サンプルの内の一部のみ使った
- NNsのトレーニングに60,000個のサンプルを使って10,000個でテストする

データをロード

```
>> 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);
```

データの規格化(1/2)

- データスペース上でデータのばらつきがある

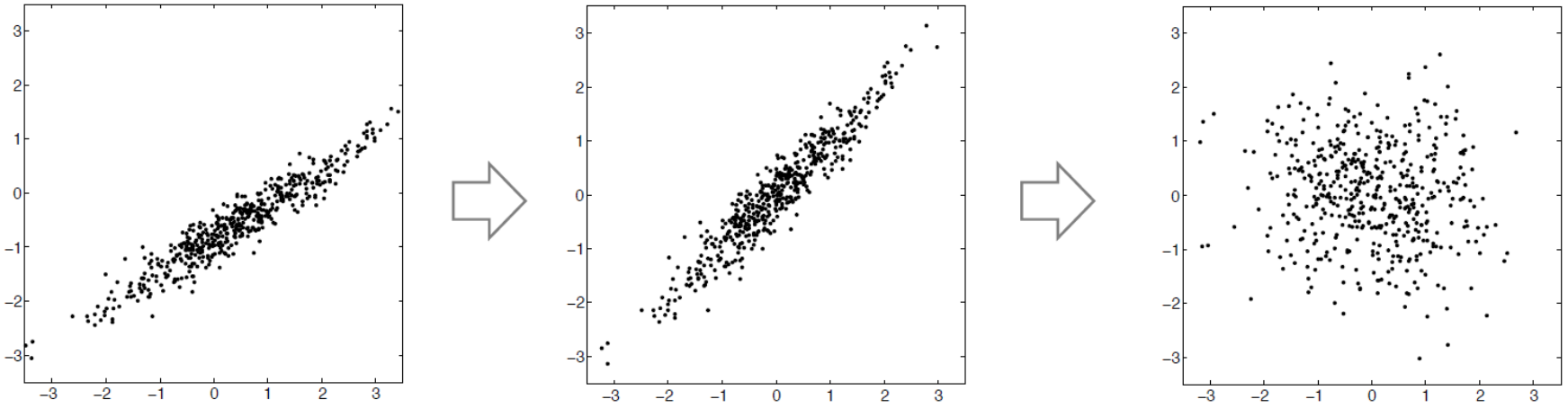
＝＞均一な分布を持つように変換するとNNsやSVMsのトレーニングの助けになる

- 平均０分散１にするとうまくいく

n^{th} sample: $\mathbf{x}_n = [x_{n1}, x_{n2}, \ldots, x_{nI}]^\top$

$$x_{ni} \leftarrow \frac{x_{ni} - \bar{x}_i}{\sigma_i} \qquad \bar{x}_i \equiv \sum_{n=1}^{N} x_{ni}/N \qquad \sigma_i = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_{ni} - \bar{x}_i)^2}$$

mean　　variance



standardization
(normalization)

whitening
(we don’t consider here)

データの規格化(2/2)

- はじめにトレーニング用サンプル**x**の平均**μ**と標準偏差**σ**を計算する

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

- 次にトレーニングサンプル**x**と平均**μ**の差をとり，標準偏差**σ**を割る

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

- さらに同じ変換を同じ**μ**と**σ**でテスト用サンプルに対して行う

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

- One-hot vectorを作成

```
>> A=eye(10,10);
>> train_d=A(train_lbl+1,:);
>> test_d=A(test_lbl+1,:);
```

実行

- 入力が784(=28×28)要素，中間layerは100unit，出力10unitの two-layer NNをデザインする

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

- サンプルを使ってネットワークをトレーニングする

```
>> opts.numepochs=1;
>> opts.batchsize=100;
>> [nn,L]=nntrain(nn,train_img,train_d,opts);
```

ネットワークがサンプルで学習する回数
重みを更新するサンプル数

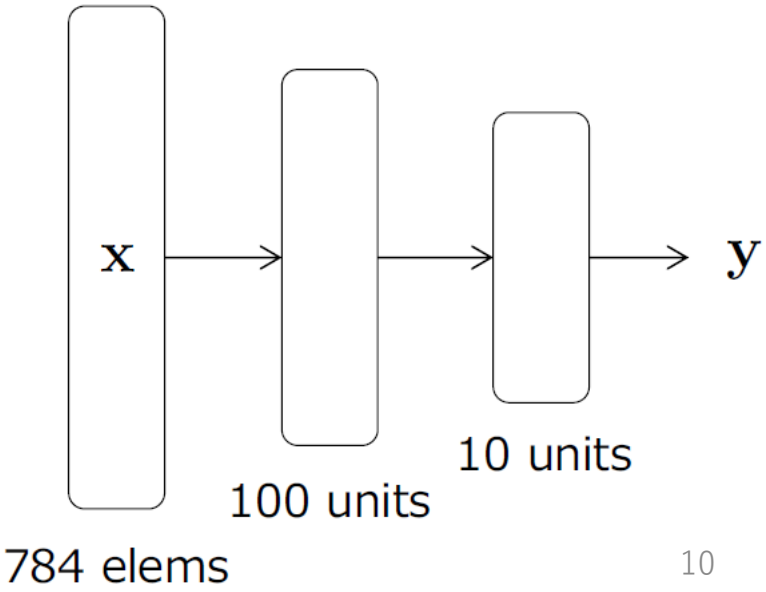
- テストサンプルを使ってパフォーマンスを評価する

```
>> pred=nnpredict(nn,test_img);
>> pred(1:10)'
ans =
   8    3    2    1    5    2    5   10    6   10
>> test_lbl(1:10)'
ans =
   7    2    1    0    4    1    4    9    5    9
>> sum(pred-1==test_lbl)/10000*100
ans =  92.500
```

この数字-1が予想値

正しい答え

正答率



Exercises 13

- 同じトレーニング用サンプルで，繰り返しトレーニングをさせることができる

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

  と入力するだけでいい.

- 初期化を行う場合は

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

  と入力するだけでいい.

問題１

初期化を行った後，10回トレーニングを繰り返す．1回トレーニングを行うごとにパフォーマンスを評価し，トレーニング回数と正答率をグラフにプロットせよ

問題２

30ユニットの中間層が2層あるニューラルネットワークをデザインし，トレーニングを行え．そのパフォーマンスを中間層１つの結果と比較せよ.

問題２　ニューラルネットワークのデザインについて

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

中間層は100unit

100unit one intermediate layer　[784 100 10]

入力が784要素

出力10unit

30unit two intermediate layer　[784 30 30 10]

中間層は30unitが2層

上の2つのパターンで正答率はどう変わるか比較する