

# Neural Network の基礎

本谷 秀堅

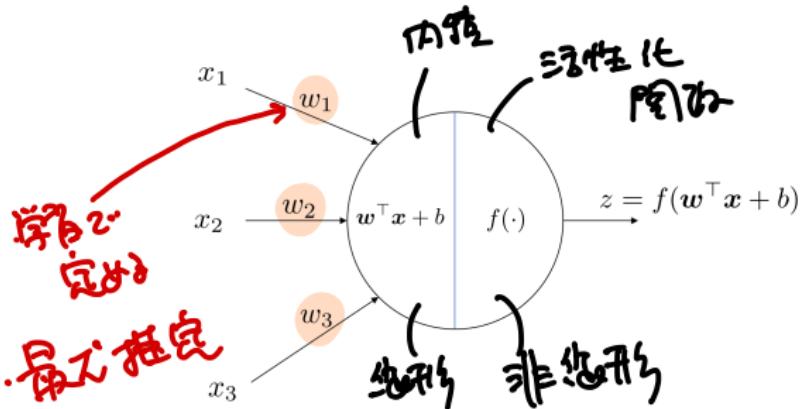
名古屋工業大学

# ニューラルネットワークによる関数の表現と学習

## ユニットの入出力

- 入力 :  $\mathbf{x} = (x_1, x_2, \dots, )^\top$
- 出力 :  $y = f(u)$ 
  - まず重みとの内積+定数を計算 :  
 $u = w_1x_1 + w_2x_2 + \dots + b = \mathbf{w}^\top \mathbf{x} + b$
  - 次に活性化関数 :  $f(u) = f(\mathbf{w}^\top \mathbf{x} + b)$

# ニューラルネットワークの構成単位



## ユニットの入出力

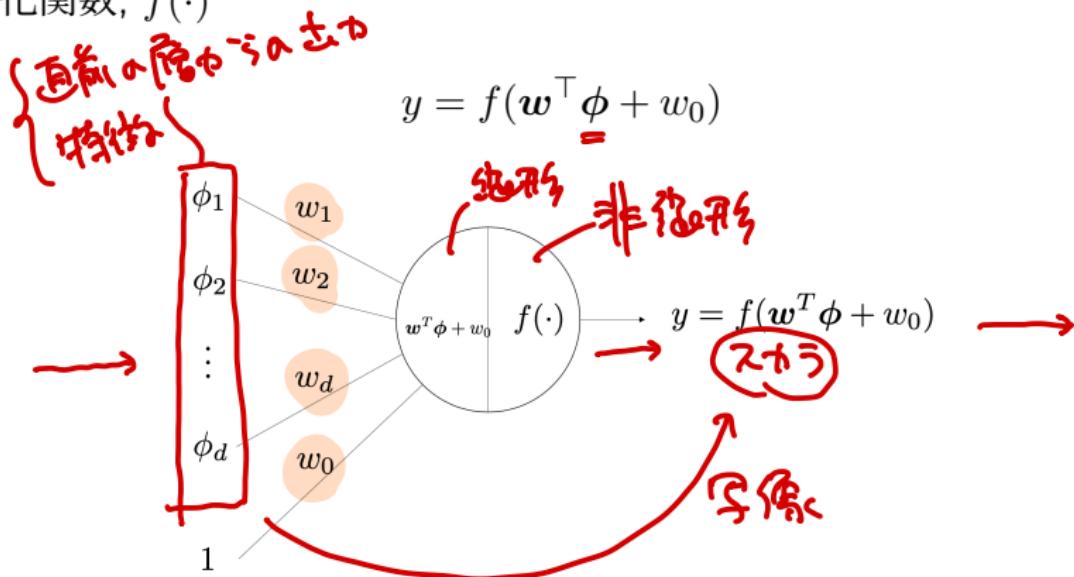
- 入力 :  $\mathbf{x} = (x_1, x_2, \dots, )^\top$
- 出力 :  $y = f(u)$ 
  - まず重みとの内積+定数を計算 :  
 $u = w_1x_1 + w_2x_2 + \dots + b = \mathbf{w}^\top \mathbf{x} + b$
  - 次に活性化関数 :  $f(u) = f(\mathbf{w}^\top \mathbf{x} + b)$

# これまでとの比較

$$\phi(x) = [\phi_1, \phi_2, \dots, \phi_d]^\top$$

活性化関数,  $f(\cdot)$

最終層.



# これまでの復習

## 2クラス識別（ロジスティック回帰）

入力,  $\mathbf{x}$ , と出力,  $z \in \mathbb{R}$  の関係:

$$z(x) = p(y=1|\mathbf{x}) = \sigma(a) = \frac{1}{1 + \exp(-(\mathbf{w}^\top \phi(\mathbf{x}) + w_0))}$$

学習データ  $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, 2, \dots, N, y_i \in \{0, 1\}\}$  が与えられている  
最尤法による係数  $\mathbf{w}$  の求める:

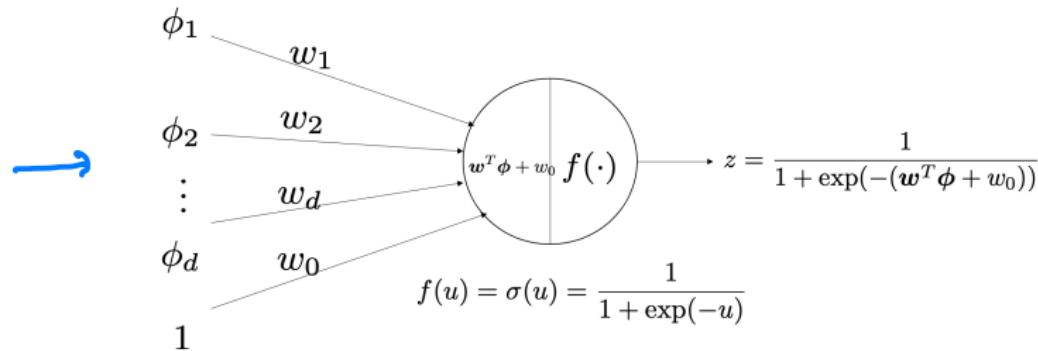
$$\begin{aligned}\hat{\mathbf{w}} &= \arg \min_{\mathbf{w}} \left[ - \sum_{i=1}^N y_i \log p(y_i|\mathbf{x}_i) \right] \\ &= \arg \min_{\mathbf{w}} \left[ - \sum_{i=1}^N \{y_i \log z(\mathbf{x}) + (1 - y_i) \log(1 - z(\mathbf{x}))\} \right]\end{aligned}$$

ナラエントロピー

# ニューラルネットワークによる2クラス識別

入力,  $\mathbf{x}$ , と出力,  $z \in (0, 1)$  の関係

$$z(\mathbf{x}) = p(y=1|\mathbf{x}) = \sigma(a) = \frac{1}{1 + \exp(-(\mathbf{w}^\top \phi(\mathbf{x}) + w_0))}$$



係数,  $\mathbf{w}$ , 決定のためのコスト関数

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \left[ - \sum_{i=1}^N \{y_i \log z(\mathbf{x}) + (1 - y_i) \log(1 - z(\mathbf{x}))\} \right]$$

多クラス辨別のときの教師信号

$$y_i = \begin{pmatrix} y_{i1} \\ y_{i2} \\ \vdots \\ y_{ik} \end{pmatrix}$$

↑  
二進数

← 当該位置 1.

Kラス

one-hot-vector

# これまでの復習

## K クラス識別

入力,  $x$ , と出力,  $z_k \in (0, 1)$  の関係 ( $k = 1, 2, \dots, K$ ) :

$$z_k(x) = p(y_k = 1|x) = \text{SM}_k(a) = \frac{\exp(a_k)}{\sum_{j=1}^K \exp(a_j)},$$

正解クラスの  $p(y|x)$  を大きくする.

$$a_k = \mathbf{w}_k^\top \boldsymbol{\phi} + w_{k0}$$

学習データ  $\mathcal{D} = \{(x_i, y_i) | i = 1, 2, \dots, N, y_{ik} \in \{0, 1\}, \sum_k y_{ik} = 1\}$

最尤法による係数  $w$  の求める:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \left[ - \sum_{i=1}^N \sum_{k=1}^K y_{ik} \log \{p(y_{ik} = 1|x)\} \right]$$

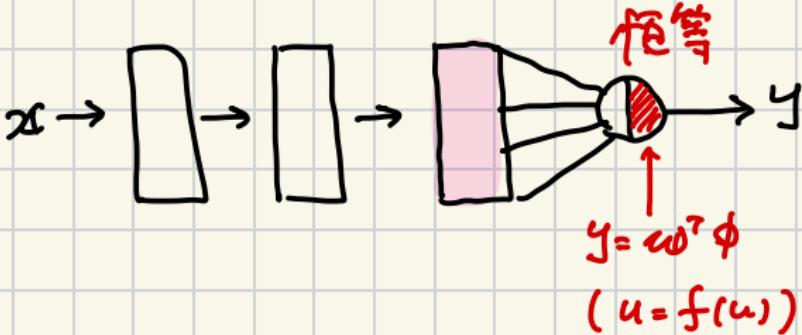
one-hot (given)

正解クラスの  $\log p(y|x)$

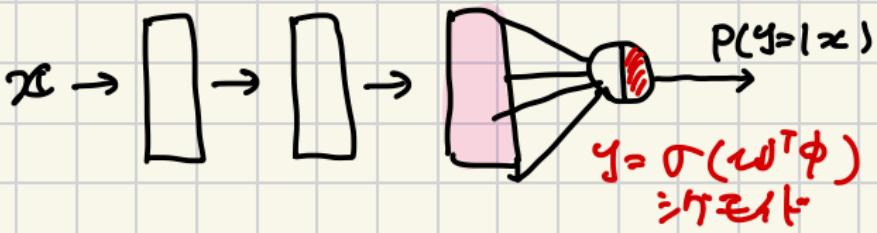
$$= \arg \min_{\mathbf{w}} \left[ - \sum_{i=1}^N \sum_{k=1}^K y_{ik} \log \frac{\exp(a_{ik})}{\sum_{j=1}^K \exp(a_{ij})} \right]$$

$a_{ik}$  のみ  
 $a_{ij}$  のみ

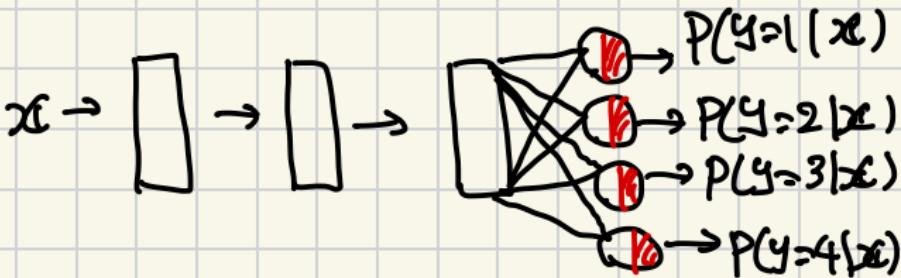
- 回帰



- 二段層神經網



- 多段層神經網



$y = \sin_k(\phi)$

✓ 7t2..72

# ニューラルネットワークによる多クラス識別

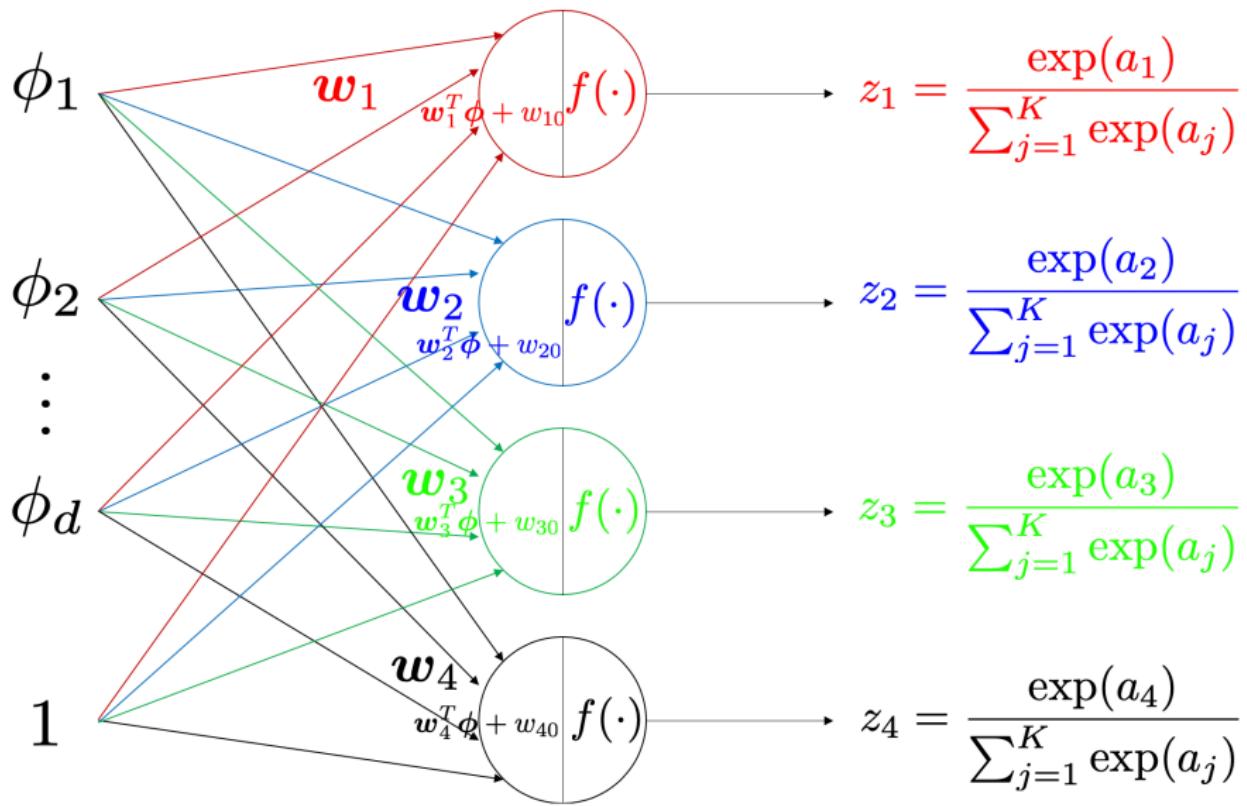
入力,  $\mathbf{x}$ , と出力,  $z_k \in (0, 1)$  の関係 ( $k = 1, 2, \dots, K$ ) :

$$z_k(x) = p(y_k = 1 | \mathbf{x}) = \text{SM}_k(a) = \frac{\exp(a_k)}{\sum_{j=1}^K \exp(a_j)}, \quad \text{ただし } a_k = \mathbf{w}_k^\top \phi + w_{k0}$$

最尤法による係数  $\mathbf{w}$  の求める:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \left[ - \sum_{i=1}^N \sum_{k=1}^K y_{ik} \log \frac{\exp(a_{ik})}{\sum_{j=1}^K \exp(a_{ij})} \right]$$

ニューラルネットワークによる多クラス識別



# 誤差の測り方（復習）

基本的に「負の対数尤度」で測る

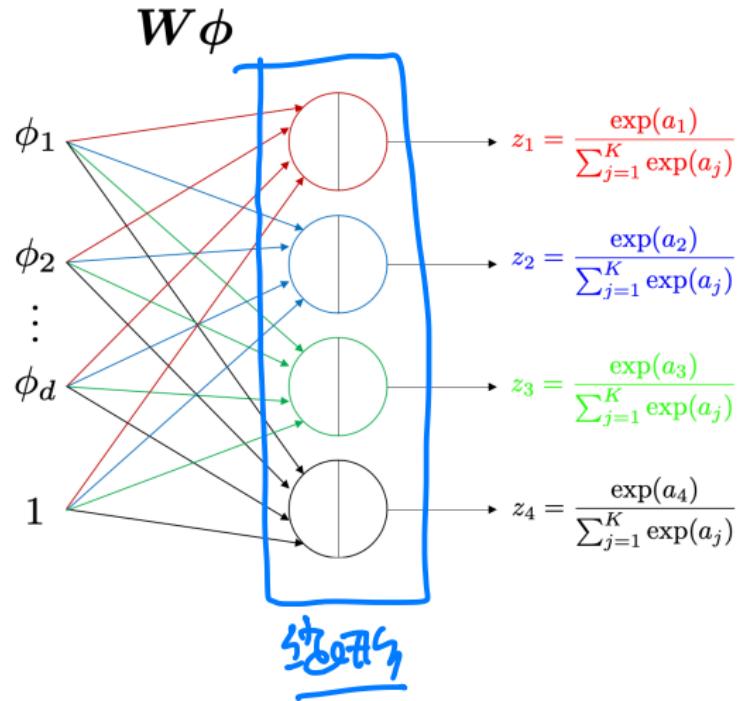
$$\log \exp(-\frac{(y-f)^2}{\sigma^2})$$

問題	出力層の活性化関数	典型的な誤差関数
回帰	恒等写像	二乗誤差
2クラス識別	ロジスティック関数	交差エントロピー
多クラス識別	ソフトマックス関数	交差エントロピー

$$\log \prod_n p_n^{y_n}$$

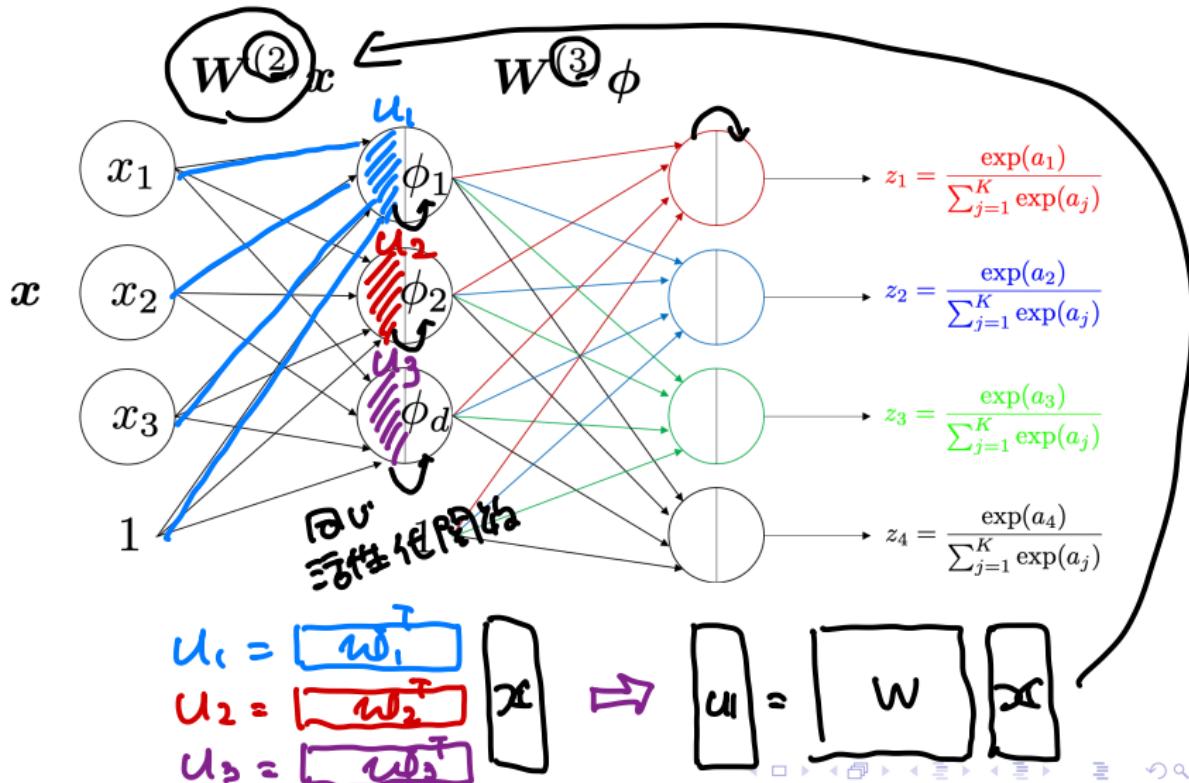
# 特徴 $\phi$ の作り方

入力  $x$  を変換するために層を増やす



# 特徴 $\phi$ の作り方

入力  $x$  を変換するために層を増やす



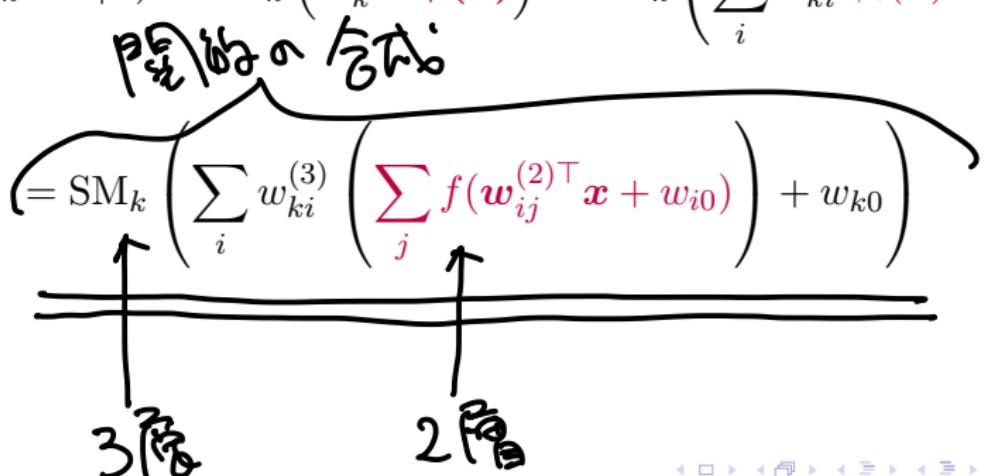
# 特徴 $\phi$ の作り方

層を増やした後の関数：

$$z = \text{SM}(\mathbf{W}^{(3)}\phi(\mathbf{x}))$$

クラス  $k$  に属する事後確率

$$z_k = p(y_k = 1 | \mathbf{x}) = \text{SM}_k \left( \mathbf{w}_k^{(3)\top} \phi(\mathbf{x}) \right) = \text{SM}_k \left( \sum_i w_{ki}^{(3)} \phi_i(\mathbf{x}) + w_{k0} \right)$$



# 特徴 $\phi$ の作り方

ニューラルネットワークの長所

最尤法による係数  $w$  の推定により 適切な特徴 (写像)  $\phi$  を自動で獲得する

例:  $K$  クラス識別

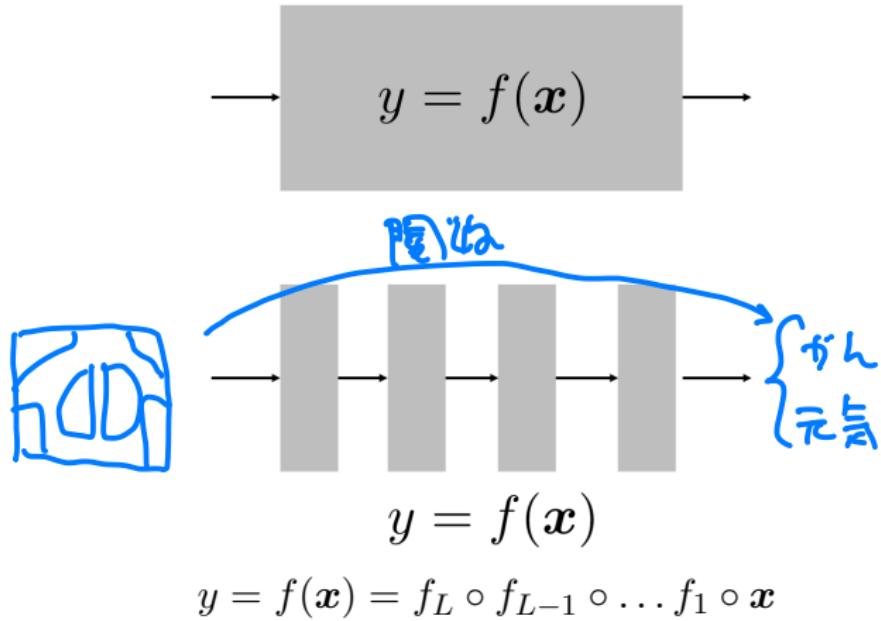
$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \left[ - \sum_{i=1}^N \sum_{k=1}^K y_{ik} \log \frac{\exp(a_{ik})}{\sum_{j=1}^K \exp(a_{ij})} \right], \quad (a_{ik} = \mathbf{w}_k^\top \phi(\mathbf{x}_i) + w_{k0})$$

特徴  $\phi(\mathbf{x}_i)$  は、前項に沿って書くと、次式のとおり

$$\phi(\mathbf{x}_i) = f(\mathbf{W}^{(2)} \mathbf{x}_i) = \begin{bmatrix} \phi_1(\mathbf{x}_i) \\ \phi_2(\mathbf{x}_i) \\ \phi_3(\mathbf{x}_i) \end{bmatrix} = \begin{bmatrix} f(\mathbf{w}_1^{(2)\top} \mathbf{x}_i + w_{10}) \\ f(\mathbf{w}_2^{(2)\top} \mathbf{x}_i + w_{20}) \\ f(\mathbf{w}_3^{(2)\top} \mathbf{x}_i + w_{30}) \end{bmatrix}$$

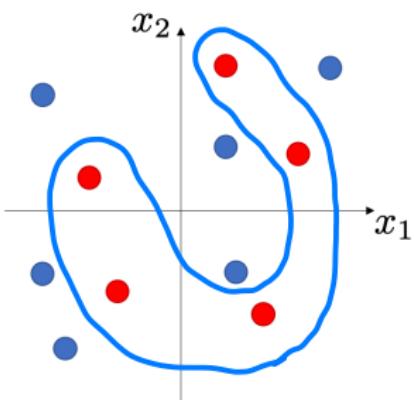
# 目的是適切な識別関数の実現

複数の素朴な関数の荷重和と合成で複雑な関数を表現



# 複雑な関数による適切な写像

パターン認識であれば線形分離性を高める特徴を得たい



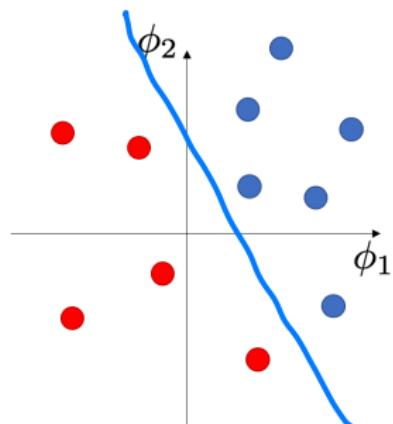
多段のニューラルネットワーク  
による写像

→  
線形化

同一カテゴリ

データへクラスタ化

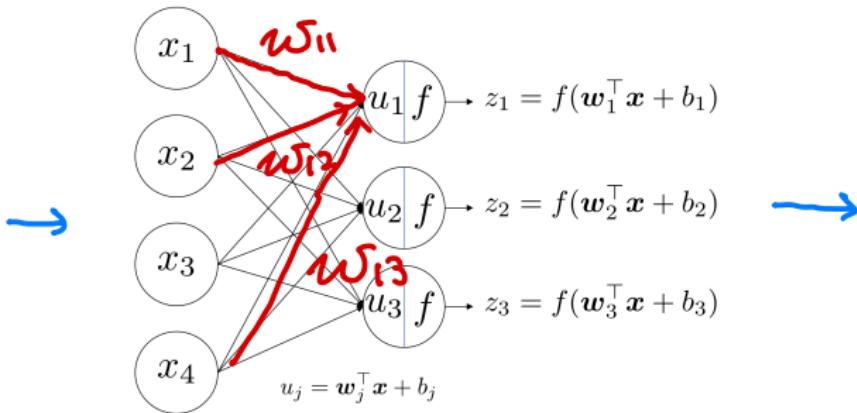
(クラス内変動に敏感)



# 参考書



# 行列表記



- 入力:  $\mathbf{x} = (x_1, x_2, x_3, x_4)^\top$
- 出力:  $\mathbf{z} = (z_1, z_2, z_3)^\top$
- $z_j = f(u_j), u_j = \mathbf{w}_j^\top \mathbf{x} + b_j$

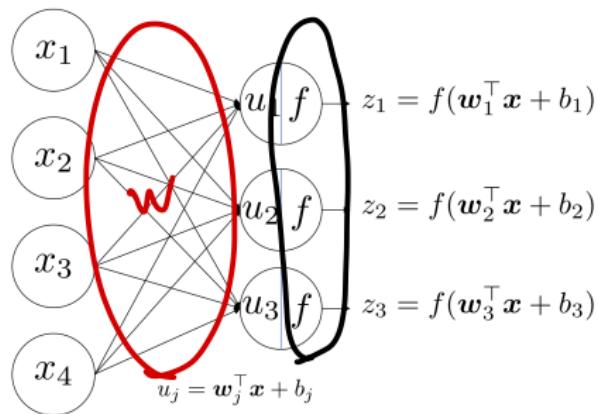
ただし ②

非恒形

① 恒形

$$\mathbf{w}_j = (w_{j1}, w_{j2}, \dots, w_{jI})^\top$$

# 行列表記



活性化関数も込みで、下記のように表す

$$z = f(u)$$

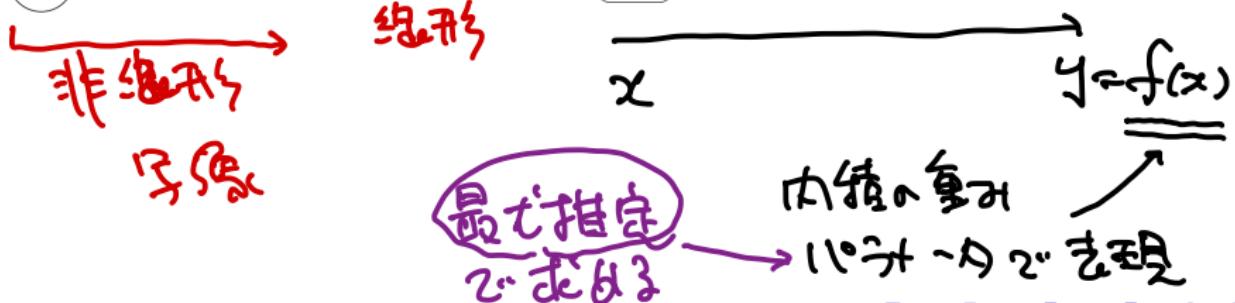
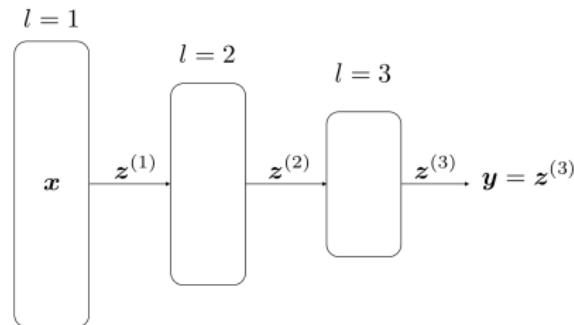
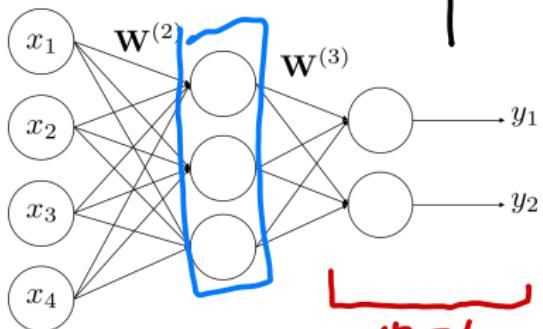
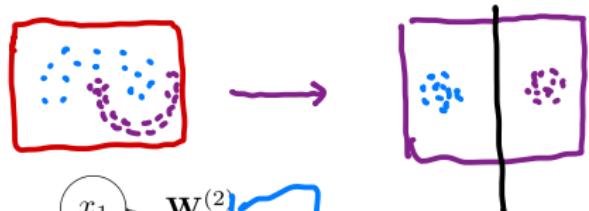
ただし  $u = \mathbf{Wx} + \mathbf{b}$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \mathbf{W} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \mathbf{b}$$

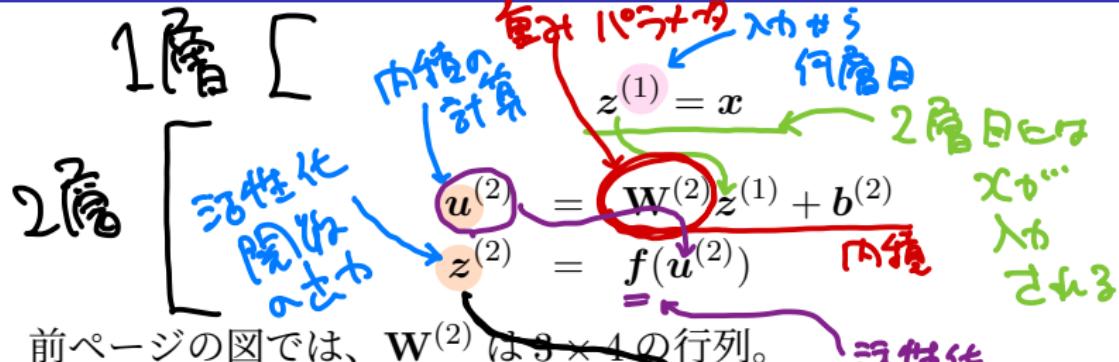
# 活性化関数

Name	Plot	Equation	Derivative
Identity		$f(x) = x$ <span style="font-size: 2em;">回帰の式</span>	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ <span style="font-size: 2em;">2段階式</span>	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Sigmoid)		$f(x) = \frac{1}{1 + e^{-x}}$ <span style="font-size: 2em;">2段階式</span>	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ <span style="font-size: 2em;">線形化</span>	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

# 層を増やす



# 層を増やす



3層 [ 活性化関数の計算 ]

$$u^{(3)} = \mathbf{W}^{(3)} z^{(2)} + b^{(3)}$$
$$z^{(3)} = f(u^{(3)})$$

前ページの図では、 $\mathbf{W}^{(3)}$  は  $2 \times 3$  の行列。

多層ニューラルネットワークの  $(l+1)$  番目の層は、直前の  $l$  層の出力  $z^{(l)}$  を受け取り次式を計算する

前ページの図では、 $\mathbf{W}^{(3)}$  は  $2 \times 3$  の行列。

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

層間の接続:

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

# 略記法

以下、バイアス項  $b$  も重みに組み込んで記述を見やすくする

- $x \leftarrow (x^\top, 1)$
- $w \leftarrow (w^\top, b)$
- $w^\top x \leftarrow w^\top x + b$

# ニューラルネットワークの学習

入力  
出力: 教師信号

## 目的

学習データの集合,  $\mathcal{X} = \{(x_1, d_1), (x_2, d_2), \dots, (x_N, d_N)\}$  を用いて、  
ニューラルネットワークが望ましい関数を表すように、各層の重み係数  
 $\mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \dots$  を求めること

最後の層  $L$  からの出力  $y = z^{(L)}$  は入力  $x$  と重み係数  
 $\mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \dots, \mathbf{W}^{(L)}$  の関数：

$$y = y(x; w) = y(x; \mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \dots, \mathbf{W}^{(L)})$$

(すべての重み係数を  $w$  で略記:  $w = \{\mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \dots, \mathbf{W}^{(L)}\}$ )

# 学習の方針

## 方針

データ  $x_n$  を入力したときの出力,  $y(x_n; w)$ , と  $d_n$  の誤差を  $L_n(w)$  で表す。全データに対する誤差の総和を小さくするように重み係数  $w$  を求める

$$w^* = \arg \min_w \sum_n L_n(w)$$

誤差の測り方は問題に応じて変える

誤差の測り方:  $L_n(w) = |y(x_n; w) - d_n|$

誤差の総和:  $\sum_n L_n(w)$

誤差の総和を最小化する重み係数  $w^*$  を求めること

$$y - d_n + \begin{cases} \text{正規分布} & \exp(-(y-d)^2/\sigma^2) \\ \log & (y-d)^2 \end{cases}$$

$\pi^d(1-\pi)^{1-d}$

$$\pi^d(1-\pi)^{1-d}$$

$\downarrow \log$

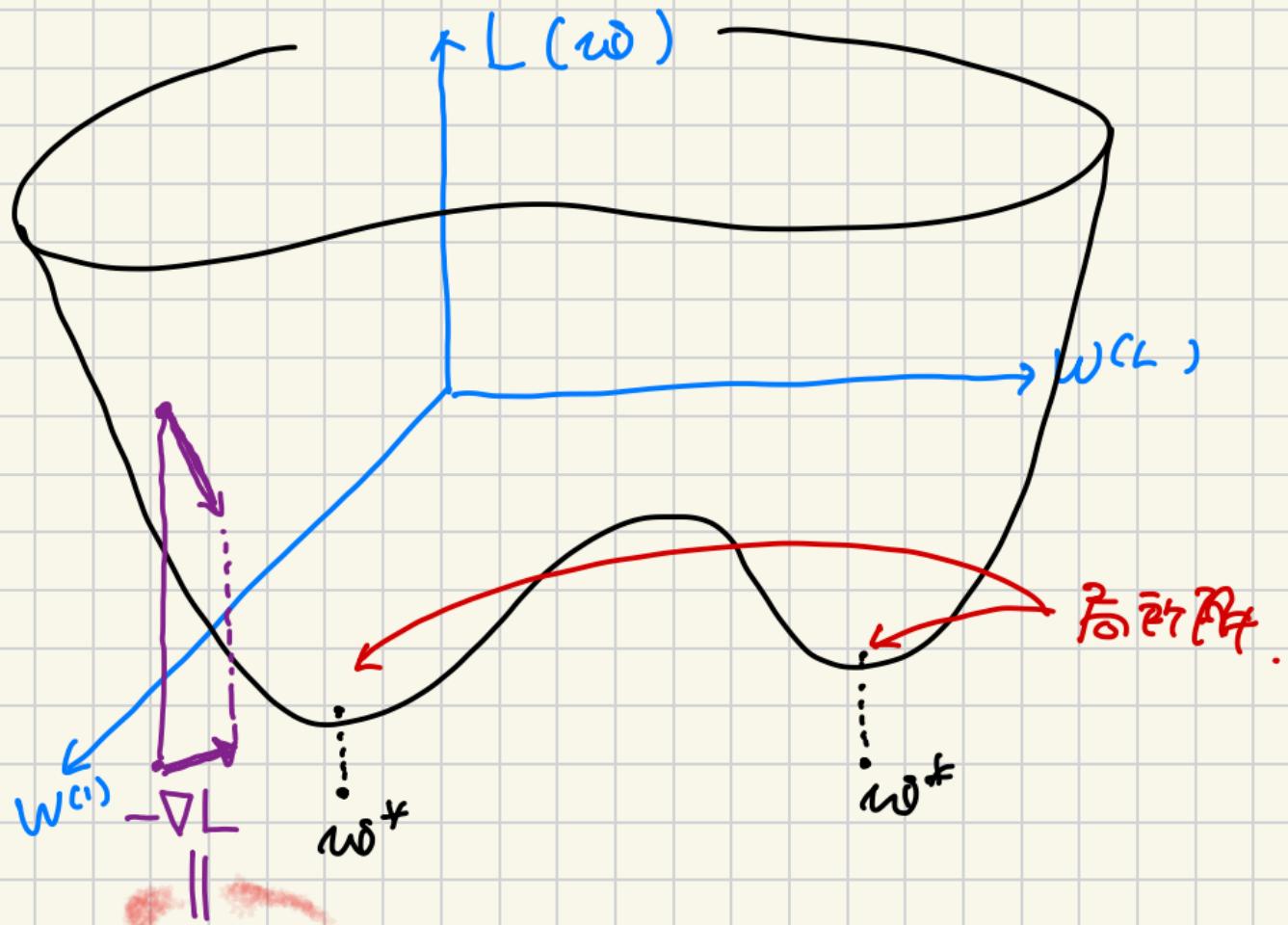
$$d \log \underline{\pi} + (1-d) \log (1-\underline{\pi})$$

given

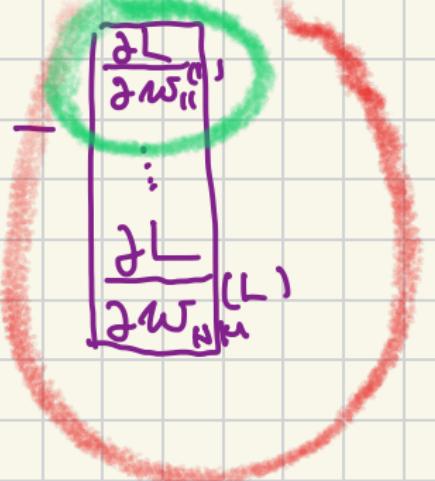
N.N. about



method  
of 2nd order Taylor

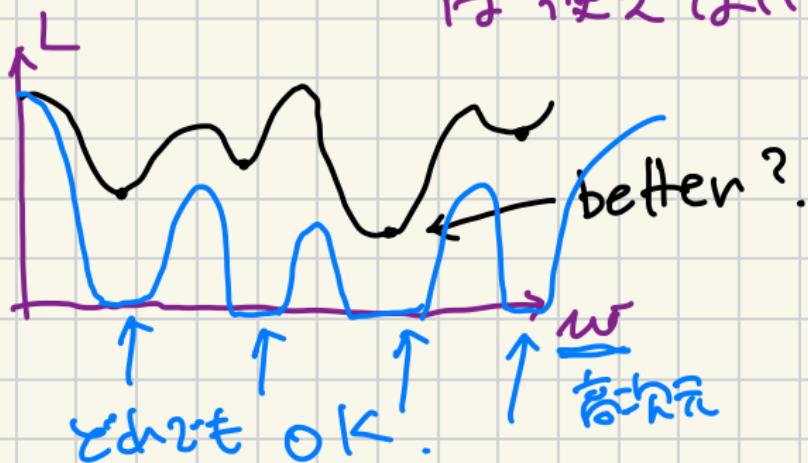


◎  $w$  が高次元では?  
Newton 法 (2 階行列)



$$\begin{bmatrix} \nabla^2 L \end{bmatrix} \quad \text{(2 階 + $N$ 次)}$$

は便でない



# 係数の求め方

## コスト関数の勾配

$$\nabla L(\mathbf{w}) = \frac{\partial L}{\partial \mathbf{w}} = \left[ \frac{\partial L}{\partial w_1}, \dots, \frac{\partial L}{\partial w_M} \right]^\top$$

## 勾配降下法によるコストの更新

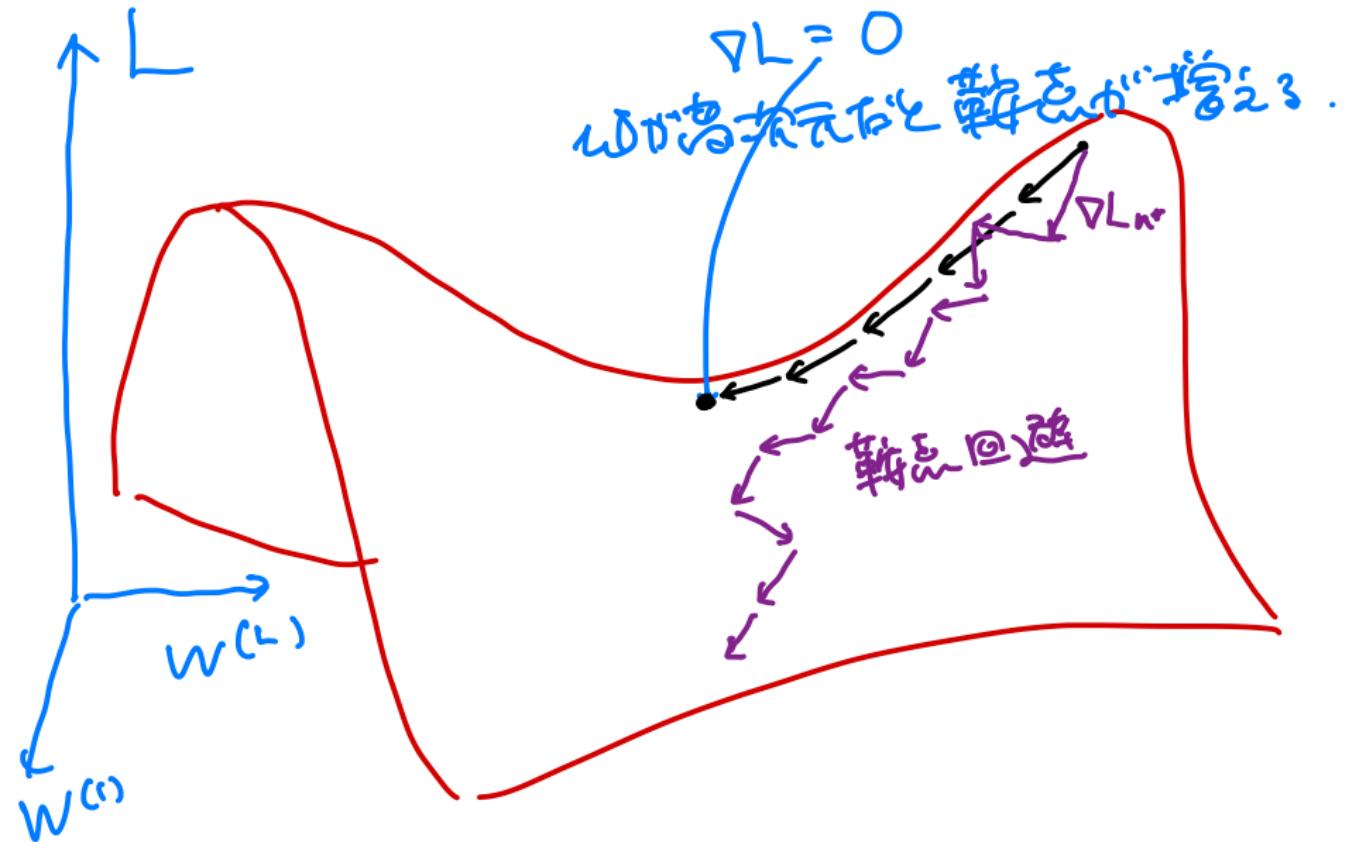
- ①  $\mathbf{w} = \mathbf{w}^0$  で初期化する
- ②  $\mathbf{w}^{t+1} = \mathbf{w}^t - \epsilon \nabla L$  による更新を繰り返す。

鞍点でよく停まる

現在

勾配の  
逆(逆) (学習率)

## コスト関数の鞍点



# 確率的勾配降下法



学習データの一部のみで勾配を求めて係数  $w$  を更新する

## Stochastic Gradient Descent: SGD

- ①  $N$  個の学習データの内の 1 つ,  $x_{n^*}$ , をランダムに選択
- ②  $w^{t+1} = w^t - \epsilon \nabla L_{n^*}$  により係数を更新。

鞍点を回避しやすい

特定のデータを複数の凸面

$$\begin{aligned}\nabla L &= \nabla \sum_n L_n \\ &= \sum_n (\nabla L_n) \rightarrow \nabla L_{n^*}\end{aligned}$$

図の説明:  $\nabla L = \nabla \sum_n L_n$  は複数の凸面  $L_n$  の和である。この和を計算する際、 $n^*$  における  $\nabla L_{n^*}$  のみが選択され、他の項は無視される。これは「特定のデータを複数の凸面」と表現されている。

# 注：学習率

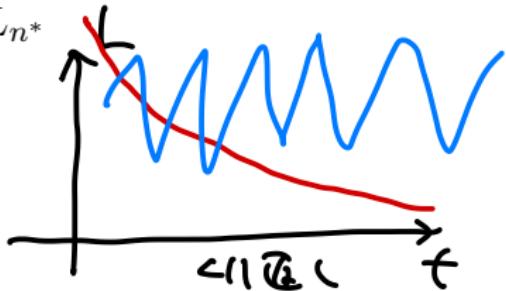
$$w^{t+1} = w^t - \epsilon \nabla L_n^*$$

係数  $\epsilon$  を学習率と呼ぶ。

振動する

- 小さすぎると収束に時間がかかる
- 大きすぎると局所解を通り過ぎる

実験により適切な値を定める必要がある



## 最適化法について

# 局所解と大域解

Neural Network の学習で得られる解は（ほぼ全て）局所解

- 局所解：近傍でコスト最小
  - 複数の局所解が存在
  - 初期値に依存して得られる解は変化
- 大域解：定義域全体の中でコスト最小

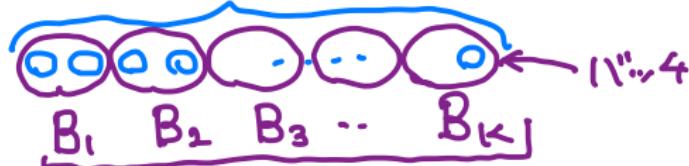
# 勾配降下法

- 確率的勾配降下法 (SGD): 訓練データをランダムに一つずつ利用

学習データ N 回

$$\nabla L_{n^*}$$

- ミニバッチ勾配降下法: 訓練データの一部を複数個利用



$$\sum_{n \in B_k} \nabla L_n$$

- 勾配降下法: 訓練データの全てを毎回利用

$$\sum_{n=1}^N \nabla L_n$$

# ミニバッチとエポック

- ミニバッチ：訓練データを複数の「ミニバッチ」に分割
  - 訓練データをランダムに並べ替えて  $M$  分割  $(\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_M)$
- エポック：全てのデータを 1 回ずつ更新に使うこと
  - 全データを  $M$  個のバッチに分割して， 全てのバッチ  $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_M$  を 1 回ずつ更新に使うと 1 エポック

# モメンタム

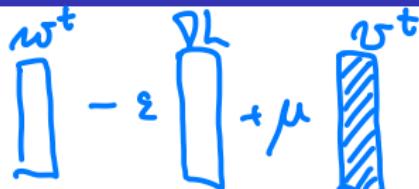
各種勾配降下法：更新時のパラメータの値における勾配のみ利用

- 収束に時間がかかることが多い

モメンタムを利用する更新法：過去の履歴を参照

- momentum = 「慣性」
  - 過去の更新傾向を今回の更新に反映
  - 振動成分を無視して「傾向」を求める

# モメンタム SGD



モメンタム項を加えた SGD の更新式

$$w^{t+1} = \underbrace{w^t - \epsilon \nabla L_{n^*}}_{SGD} + \underbrace{\mu v^t}_{モメンタム項}$$

右辺第三項がモメンタム項（慣性項）

$$v^t = \underbrace{w^t - w^{t-1}}_{その量} \quad \text{モメンタム}$$

$$\begin{array}{c} \text{モメンタム} \\ = \end{array} \quad \begin{array}{c} \text{現在} \\ - \end{array} \quad \begin{array}{c} \text{前} \\ \end{array}$$

$$w^{t+2} = \underline{w^t} - \varepsilon \nabla L + \mu v^{t+1}$$

$$= w^t - \varepsilon \nabla L + \mu v^t - \varepsilon \nabla L + \mu v^{t+1}$$

w<sup>t</sup>                          v<sup>t+1</sup>

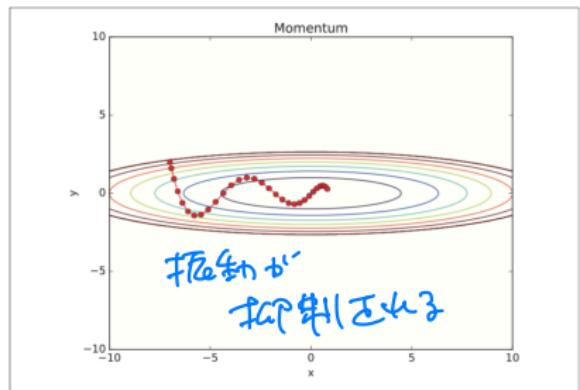
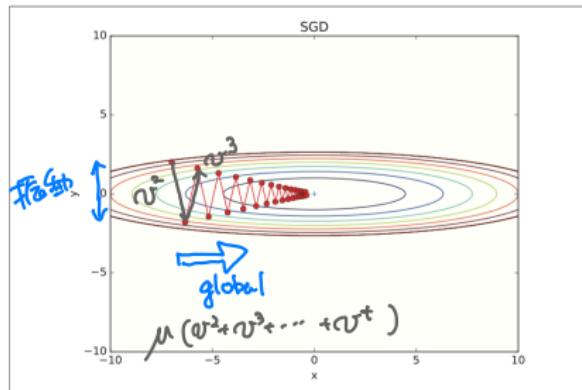
v<sup>t</sup>

過去の状態の  
「経験」

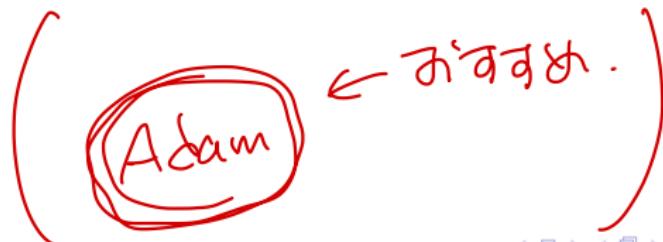
「学習する」

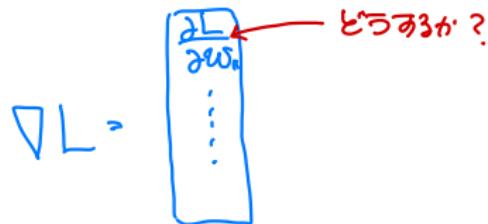
反復される。

# SGD vs Momentum SGD



<https://www.dragonarrow.work/articles/195>

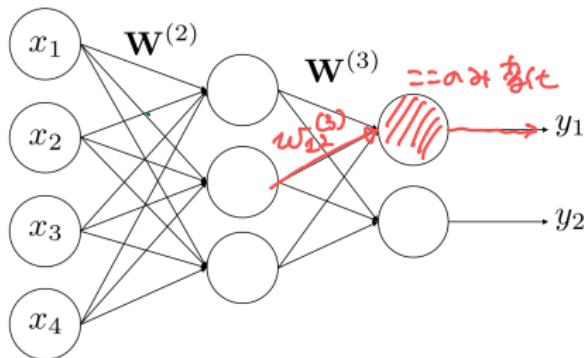




誤差逆伝搬

Back Propagation

# 誤差逆伝播法（準備）



回帰分析

$$(y_1 - d_1)^2 + (y_2 - d_2)^2$$

wの関数

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w}$$

$$= 2(y_1 - d_1) \frac{\partial y_1}{\partial w} \\ + 2(y_2 - d_2) \frac{\partial y_2}{\partial w}$$

2層ネットワークを題材に

入力は  $x = (x_1, x_2, x_3, x_4)^\top$ 。  $z_j^{(1)} = x_j$  とする。

次の層の出力は次のとおり

$$z_j^{(2)} = f(u_j^{(2)}) = f \left( \sum_i w_{ji}^{(2)} z_i^{(1)} \right)$$

出力層は次のとおり。回帰を想定して活性化関数は恒等写像

$$y_j(\mathbf{x}) = \underline{z_j^{(3)}} = \underline{u_j^{(3)}} = \sum_i w_{ji}^{(3)} z_i^{(2)}$$

出力層  
 最終層  
 ニューラルネット  
 ニューラルネットの出力  
 前の層のユニットへの入力  
 前からの出力

誤差関数は二乗誤差とする

$$L_n = \frac{1}{2} \| \mathbf{y}(\mathbf{x}_n) - \mathbf{d}_n \|^2 = \frac{1}{2} \sum_j (y_j(\mathbf{x}_n) - d_{nj})^2$$

ニューラルネットへの入力  
 教師信号

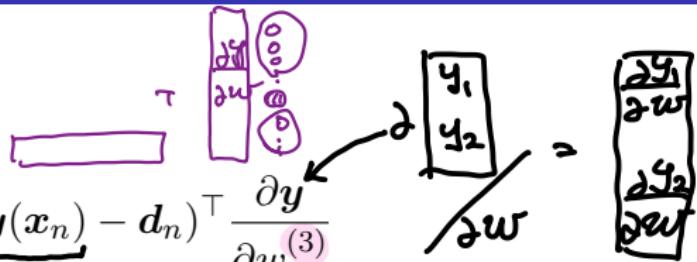
1, 2 (ネットユニットへの入力)  
 層の番号 (最終層)

内積計算式

回帰では二乗誤差関数を用いる  
 恒等関数  $f(u) = u$

# 誤差逆伝播法

まず出力層の重みで微分



$\partial y / \partial w_{ji}^{(3)}$  は第  $j$  成分のみ  $z_i^{(2)}$ 。  
= えーうしょんとくーぐの  
孟かがいに伝布の仕様

$$\frac{\partial y}{\partial w_{ji}^{(3)}} = [0, \dots, 0, \underset{\text{2層目}}{z_i^{(2)}}, 0, \dots, 0]$$

前から2層目  
前の層の  
2層目の出力

3層目の  
ユニット  
出力

2層目

3層目以外のユニットの  
出力は零でよい

$$\frac{\partial L_n}{\partial w_{ji}^{(3)}} = \underbrace{(y_j(x_n) - d_j) z_i^{(2)}}_{\substack{\text{3層目のユニットの出力} \\ \text{の誤差}}}$$

前の層へ  
出力

よって

# 誤差逆伝播法

あたかも逆に運転しているかのように思ってよい。

中間層の重みによる微分  $\partial L_n / \partial w_{ji}^{(2)}$  の計算。重み  $w_{ji}^{(2)}$  は

$$u_j^{(2)} = \sum_i w_{ji}^{(2)} z_i^{(1)}$$

↑  
2層目のユニットjの  
内積

右辺第二項は  $u_j^{(2)} = \sum_i w_{ji}^{(2)} z_i^{(1)}$  より次の通り

$$\frac{\partial L_n}{\partial w_{ji}^{(2)}} = \frac{\partial L_n}{\partial u_j^{(2)}} \frac{\partial u_j^{(2)}}{\partial w_{ji}^{(2)}}$$

$$\frac{\partial u_j^{(2)}}{\partial w_{ji}^{(2)}} = z_i^{(1)}$$

右辺第一項  $L_n / \partial u_j^{(2)}$  は、 $u_j^{(2)}$  の変化による  $L_n$  の変化を表現。 $u_j^{(2)}$  の変化は活性化関数を介して出力層の  $u_k^{(3)}$  を変化させる。 $L_n$  は  $u_k^{(3)}$  の変化に伴い変化する。

$$\frac{\partial L_n}{\partial u_j^{(2)}} = \sum_k \frac{\partial L_n}{\partial u_k^{(3)}} \frac{\partial u_k^{(3)}}{\partial u_j^{(2)}}$$

3層目の  
出力層に  
2層目の  
出力層に  
2層目の  
内積に

# 誤差逆伝播法

$$L_n = 1/2 \sum_k (y_k(\mathbf{x}_n) - d_k)^2 = 1/2 \sum_k (u_k^{(3)} - d_k)^2$$

$$\frac{\partial L_n}{\partial u_k^{(3)}} = u_k^{(3)} - d_k$$

2層目の出力

$$u_k^{(3)} = \sum_i u_{ki}^{(3)} f(u_i^{(2)})$$

3層目のユニット  
の内積

2層目の  
活性化関数

2乗誤差を行なう(2)  
1次の式に

活性化関数  
導関数

$$\frac{\partial u_k^{(3)}}{\partial u_j^{(2)}} = w_{kj}^{(3)} f'(u_j^{(2)})$$

以上より

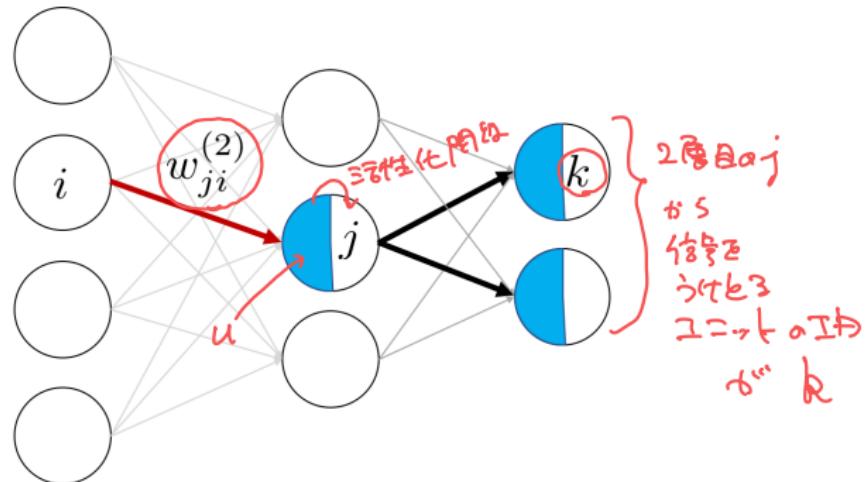
$$\frac{\partial L_n}{\partial w_{ji}^{(2)}} = \left( f'(u_j^{(2)}) \sum_k w_{kj}^{(3)} (u_k^{(3)} - d_k) \right) z_i^{(1)}$$

①

②

# 誤差逆伝播法

$$\frac{\partial L_n}{\partial w_{ji}^{(2)}} = \left( f'(u_j^{(2)}) \sum_k w_{kj}^{(3)} (u_k^{(3)} - d_k) \right) z_i^{(1)}$$



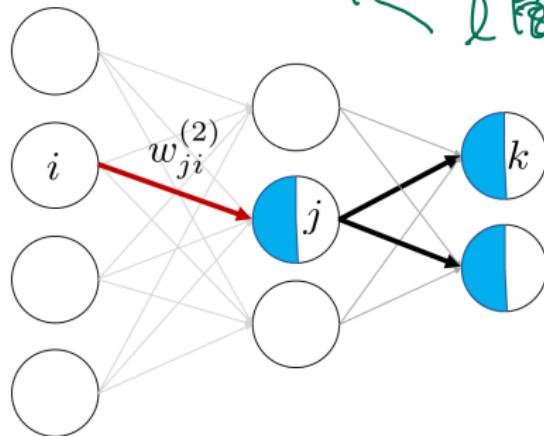
# 誤差逆伝播法

より一般的に

$$\frac{\partial L_n}{\partial w_{ji}^{(l)}} = \frac{\partial L_n}{\partial u_j^{(l)}} \frac{\partial u_j^{(l)}}{\partial w_{ji}^{(l)}}$$

$l$  層目の変化は

$l+1$  層目の  
変化  $(k)$   
ユニット  $k$   
への影響



右辺第一項は  $u_j^{(l)}$  の変化により  $L_n$  に生じる変化。 $u_j^{(l)}$  の変化は、次の  $(l+1)$  層の  $u_k^{(l+1)}$  を介して出力を変化させて、コスト関数に影響を与える。

# 誤差逆伝播法

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} \frac{\partial u}{\partial u}$$
$$\frac{\partial L_n}{\partial u_j^{(l)}} = \sum_k \frac{\partial L_n}{\partial u_k^{(l+1)}} \frac{\partial u_{k,l}^{(l+1)}}{\partial u_j^{(l)}}$$

l+1層目  
の  
 $\delta_k^{(l+1)}$ が分かれ  
 $\delta_j^{(l)}$ が計算可

ここで、第  $l$  層の第  $j$  番目のユニットについて、次式を定義する

$$\delta_j^{(l)} = \frac{\partial L_n}{\partial u_j^{(l)}}$$

l層目  
のユニットへ  
内層の値を  
伝搬

$$u_k^{(l+1)} = \sum_i w_{ki}^{(l+1)} f(u_i^{(l)}) \text{ より}$$
$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} \left( w_{kj}^{(l+1)} f'(u_j^{(l)}) \right)$$

$\delta_j^{(l)}$  は、ひとつ上の層の  $\delta_k^{(l+1)}$  から計算できる。

# 誤差逆伝播法

出力層から入力層に向かって  $\delta_j^{(l)}$  の計算を「伝播」させる。

$$\frac{\partial u_j^{(l)}}{\partial w_{ji}^{(l)}} = z_i^{(l-1)}$$
 より結局次式を得る

前の層の出力

$$\frac{\partial L_n}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)}$$

各計算にネットワーク全体を見る必要はない。局所的に計算可能。

出力層での微分は採用したコスト関数に依存して変わる。

出

$$\delta_j^{(L)} = \frac{\partial L_n}{\partial u_j^{(L)}}$$

出力層の  
計算は専

# 誤差逆伝播法

## 誤差逆伝播による勾配の計算方法

- 入力：学習データ  $(\mathbf{x}_n, \mathbf{d}_n)$
  - 出力：コスト関数  $L_n(\mathbf{w})$  の、各重み係数による微分,  $\partial L_n / \partial w_{ji}^{(l)}$
- ❶  $\mathbf{z}^{(1)} = \mathbf{x}_n$ 。各層への入力  $\mathbf{u}^{(l)}$  と出力  $\mathbf{z}^{(l)}$  を順に計算する
  - ❷ 出力層での  $\delta_j^{(L)}$  を計算する。普通は  $\delta_j^{(L)} = z_j - d_j$
  - ❸ 中間層での  $\delta_j^{(l)}$  を次式に従って逆向きに計算する ( $l = L, L-1, \dots$ )

$$\delta_j^{(l)} = \sum_k \delta^{(l+1)} \left( w_{kj}^{(l+1)} f'(u_j^{(l)}) \right)$$

- ❹ 各層  $(l)$  のパラメータによる微分を次式で計算する

$$\frac{\partial L_n}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)}$$