情報システム実験実習II（後期）
機械学習によるデータ解析
2回目

情報システムコース 内田雅人

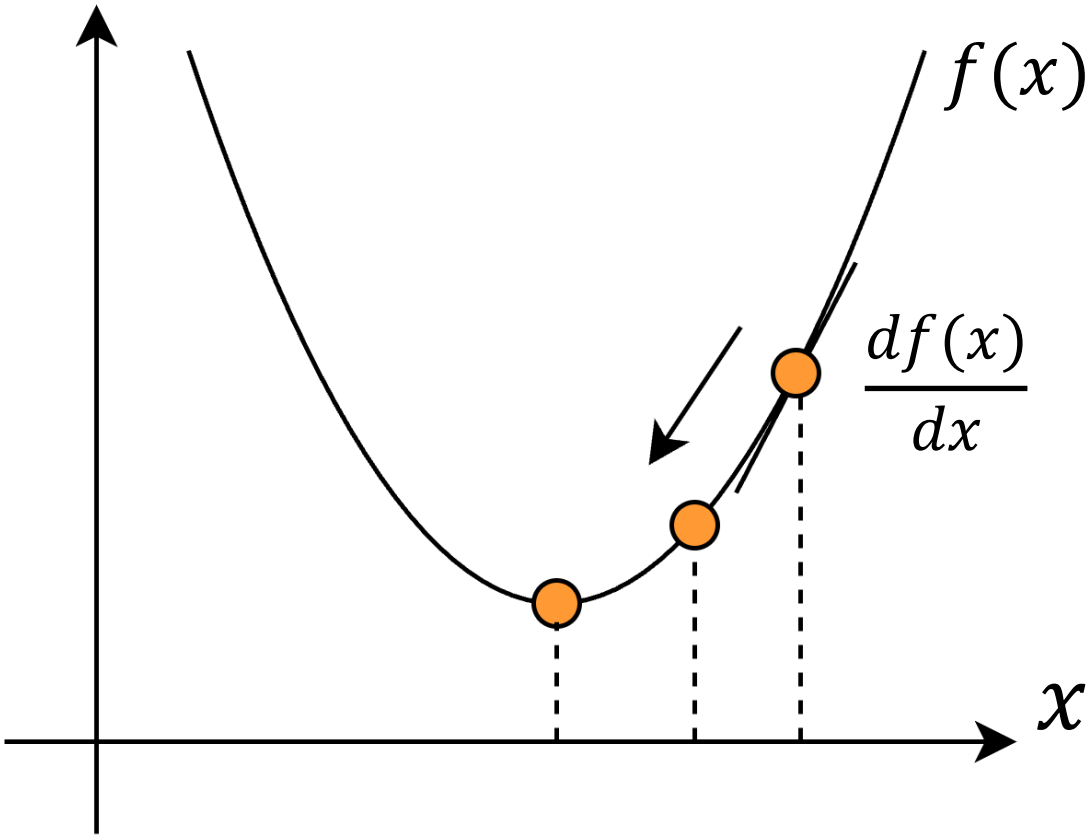
勾配降下法

勾配降下法

- 目的関数$f(x)$を最小にするxの値を数値計算により求める方法
  - ◆$\frac{df(x)}{dx} = 0$となる点を見つける
  - ◆最適化問題
    - 特定の値を最小もしくは最大にする問題
    - 機械学習の教師あり学習において基本的に誤差を最小にする問題
  - ※ 数値解（近似解）なので基本的に誤差がある

勾配降下法のイメージ

- $f(x)$の外形が不明でも微分は可能
  - ◆傾きのマイナス方向に進めば極小値へ辿り着く
  - ◆ちょっと動かして微分値求めて更新して…を繰り返す



関数の最小値を求める場合

- 目的関数$f(x)$の最小値を求める
  - ◆最小値を取るxは不明
  - ◆$f(x)$の勾配（微分値）はわかる
    $\rightarrow \frac{df(x)}{dx} = 0$を満たす$x$を求めれば良い

- xの初期値を乱数，ηを更新率として更新式は

$$x \leftarrow x - \eta \frac{df(x)}{dx} \quad \cdots (9)$$

左辺に右辺を代入

例：$f(x) = (x-1)^2$の最小値を求める

- 目的関数の微分値

$$\frac{df(x)}{dx} = \frac{d(x^2 - 2x + 1)}{dx} = 2x - 2$$

- 更新式に当てはめると

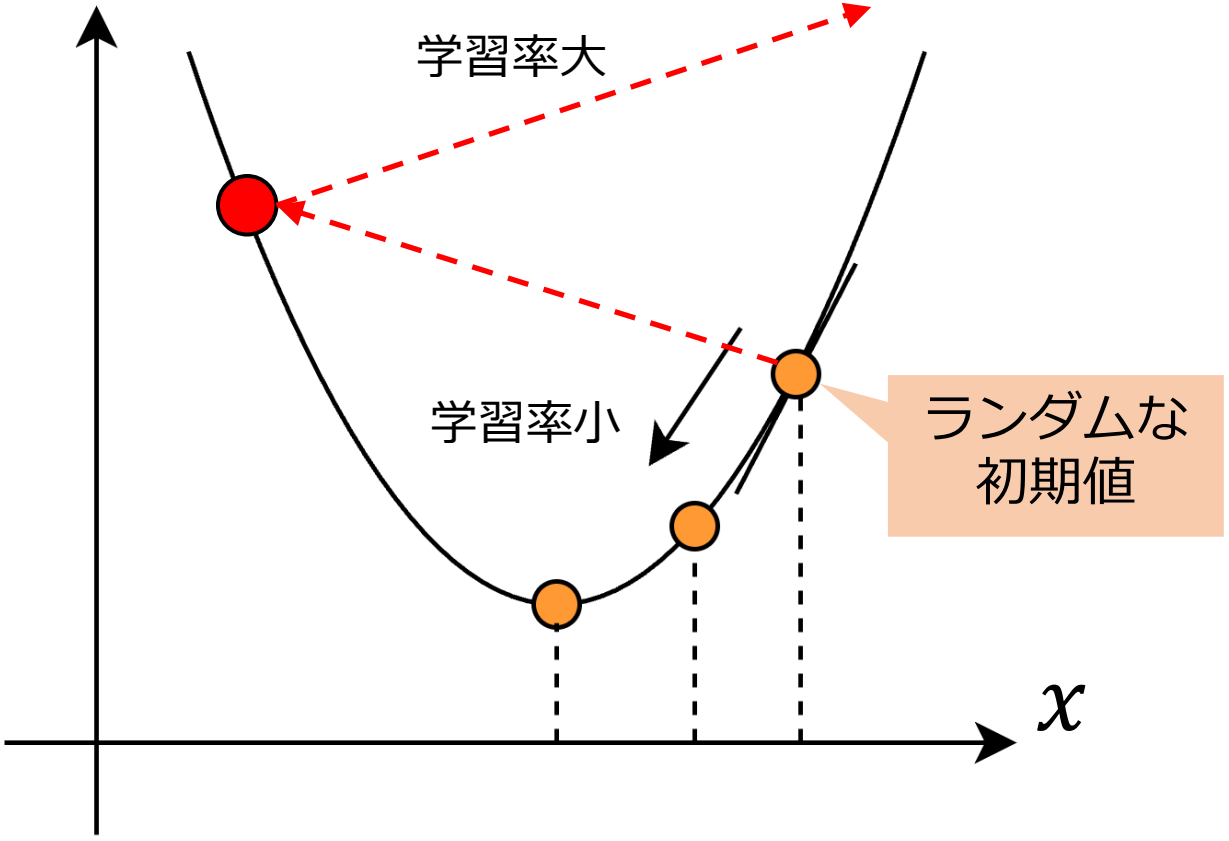
$$x \leftarrow x - \eta(2x - 2)$$

- 擬似コード

```
# 学習率
learning_rate = 0.1
# 繰り返し学習する回数
max_iteration = 100
# 勾配法により値を更新するループ
for i in range(max_iteration):
    # 勾配
    g_x = 2 * x - 2
    # 更新
    x = x - learning_rate * g_x
```

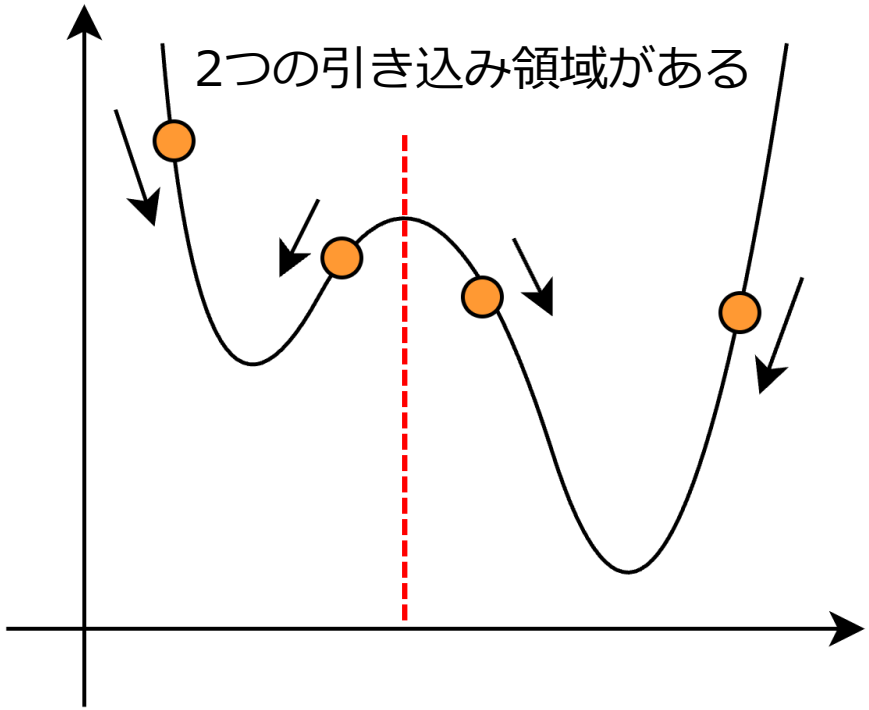
学習率による収束の違いのイメージ

- 学習率小：徐々に谷底へいく
- 学習率大：更新量が多すぎて発散



大域的最適解と局所的最適解

- 目的関数と初期値によっては最小値にいかず，別の極値に収束する場合がある
  - ◆大域的最適解（Global minimum）：全体の中での最小値
  - ◆局所的最適解（Local minimum）：ある区間での最小値
  - ◆引き込み領域：ある最適解に引き込まれる点の集合



一次関数のパラメータ推定(1)

- 例：関数$f(x) = wx + b$の係数を求める
  - ◆身長–体重のようなデータの組$\boldsymbol{x}$，$\boldsymbol{y}$があるとする

- 関数と真値yとの平均二乗和誤差を目的関数として以下のように定義（nはデータ数）

$$E = \frac{1}{n}\sum (f(x_i) - y_i)^2$$

- w，bの初期値を乱数，ηを学習率とすると更新式は

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}, \quad b \leftarrow b - \eta \frac{\partial E}{\partial b}$$

一次関数のパラメータ推定(2)

■ 証明は省略

$$\frac{\partial E}{\partial w} = \frac{\partial}{\partial w}\left(\sum (f(x_i) - y_i)^2\right)$$
$$= \frac{2}{n}\sum x_i(f(x_i) - y_i)$$

$$\frac{\partial E}{\partial b} = \frac{2}{n}\sum (f(x_i) - y_i)$$

※ 複数のデータ（バッチ）で実行するので，バッチ勾配降下法ともいう

※ 細かい微分の途中過程は省略

演習(3)：勾配降下法

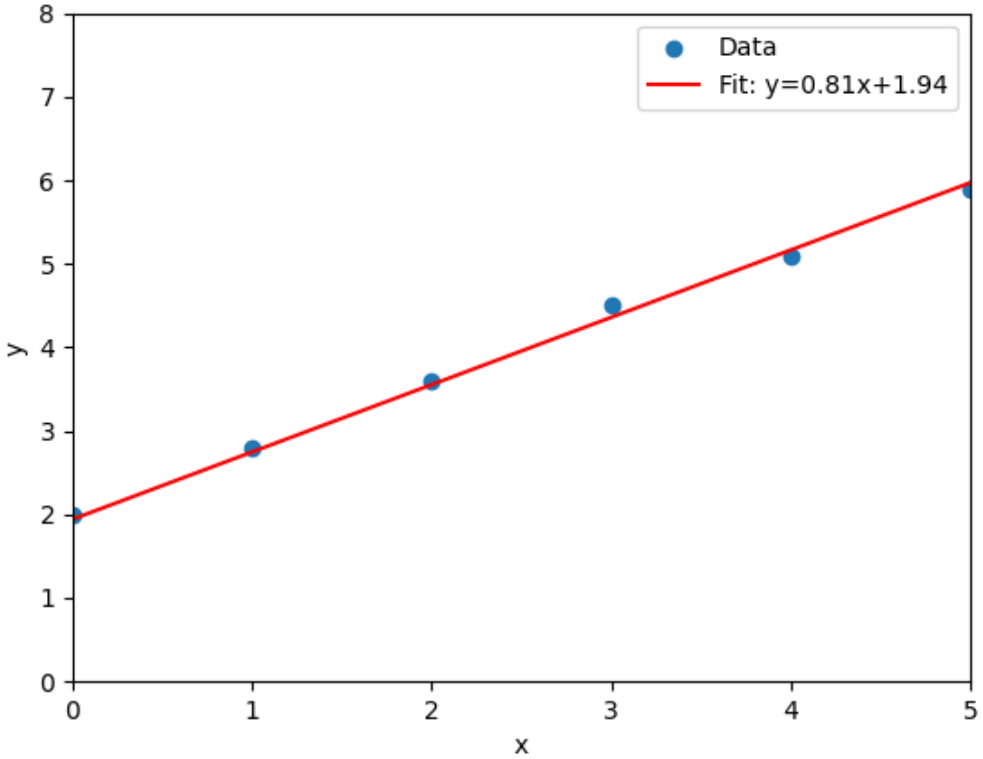
- 勾配降下法により一次関数$y = wx + b$の係数を求めるプログラムを完成させよ.

- 演習で使って良いnumpyの関数など
  - ◆ガウス分布で乱数生成：rng.standard_normal()
  - ◆平均：np.mean()
  - ◆累乗：np.pow()，あるいは「変数名 ** 2」

- プログラムの実行（/scripts内で実行）
  $ uv run gradient_method.py

毎回，前回の演習と結果が異なる

- 前回：$a = 0.780000,\ b = 2.033333$
- 今回(例)：$w = 0.805403,\ b = 1.943190$
  （小数点以下6桁で打ち切り示してある）



見た目はほとんど変わらない

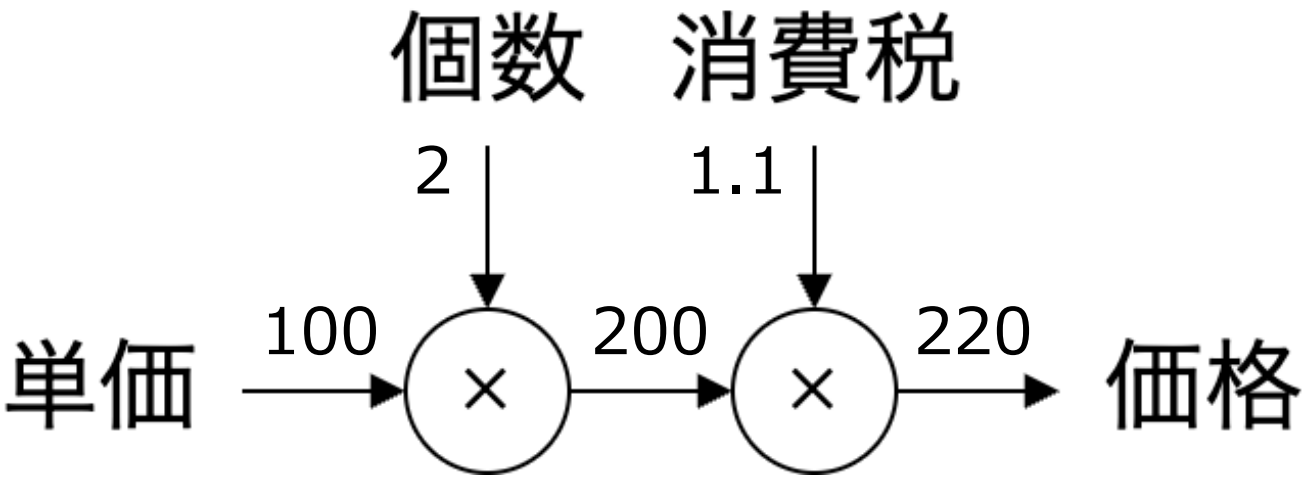
ディレクトリ構造

```
02_perceptron
├ outputs
┃        └ gradient_method.png（出力結果）
├ scripts
┃        ├ gradient_method.py（演習）
┃        └ .py
├ pyproject.toml
└ requirements.txt
```

計算グラフ
順伝播と逆伝播

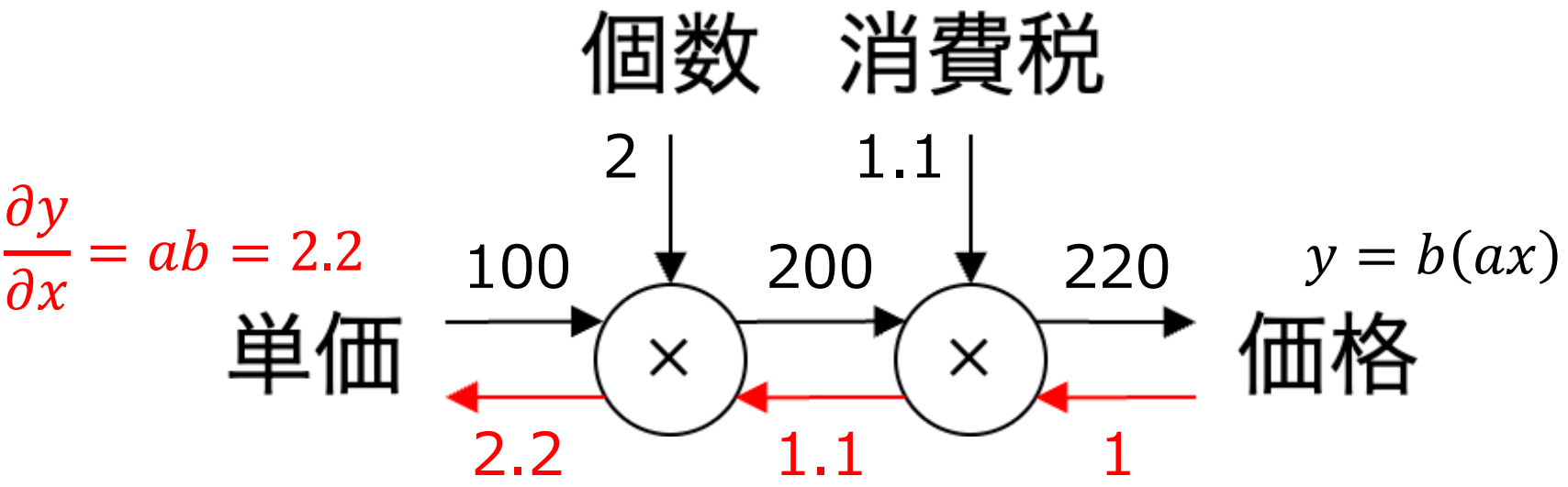
計算をノードとエッジで表現してみる

- 例として「価格＝単価×個数×消費税」をノードとして表す
  - ◆数字は具体的な数字例



入力の変化は出力にどう影響する？

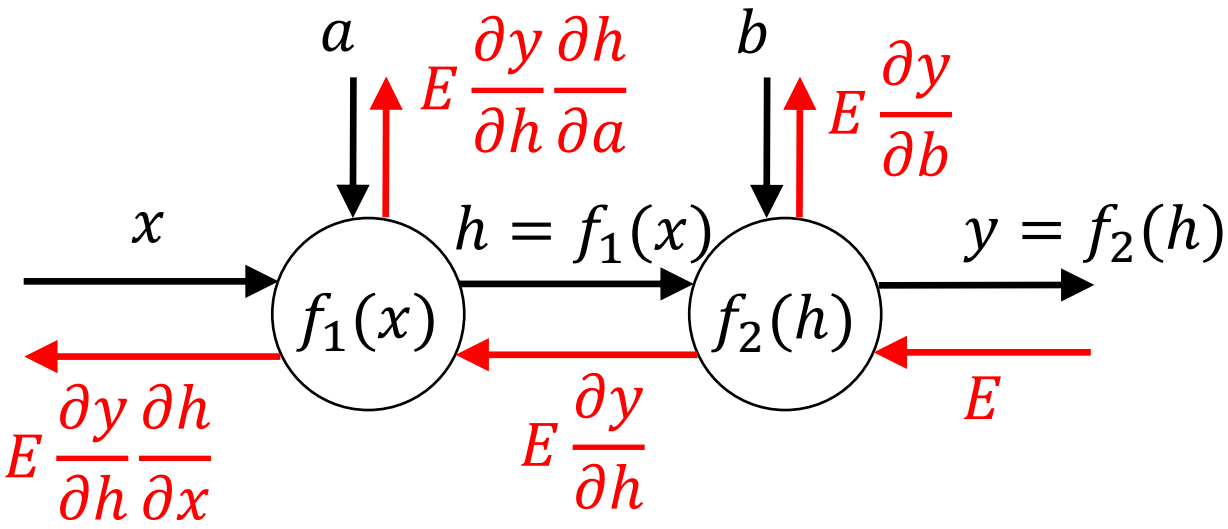
- 「最終的な価格への影響するか」は
  「単価の値段に関する支払い額の微分」
  - ◆単価をx，個数をa，消費税をb，価格をyとする
  - ◆数字は具体的な数字例，赤字は微分値



単価が変わると
2.2倍の影響がある

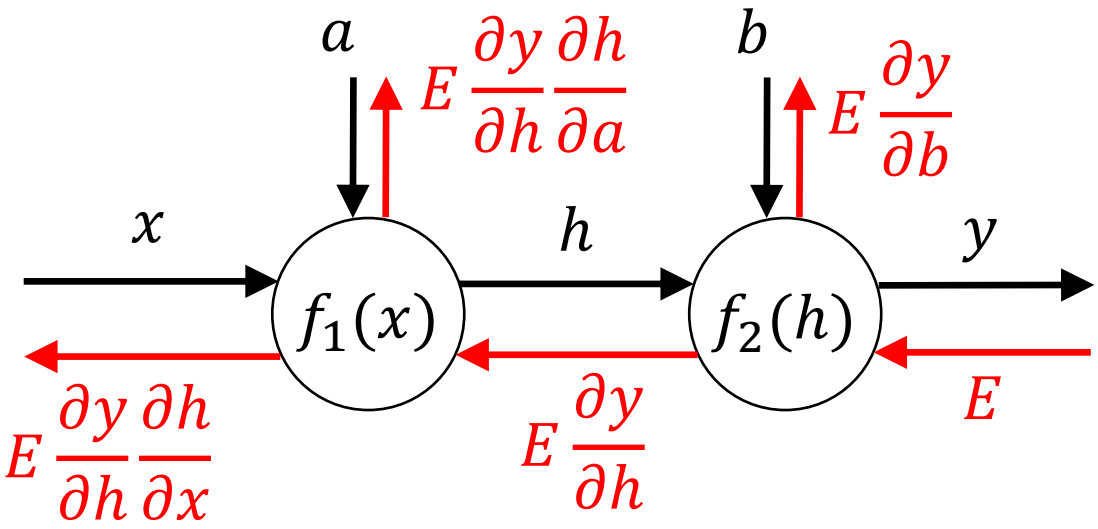
計算グラフの一般化(1)

- 出力に対する各パラメータの変化量は微分で表現可能計算過程は全て関数として表現
  - ◆出力に対する各パラメータの微分は合成関数の微分＝連鎖律で表現



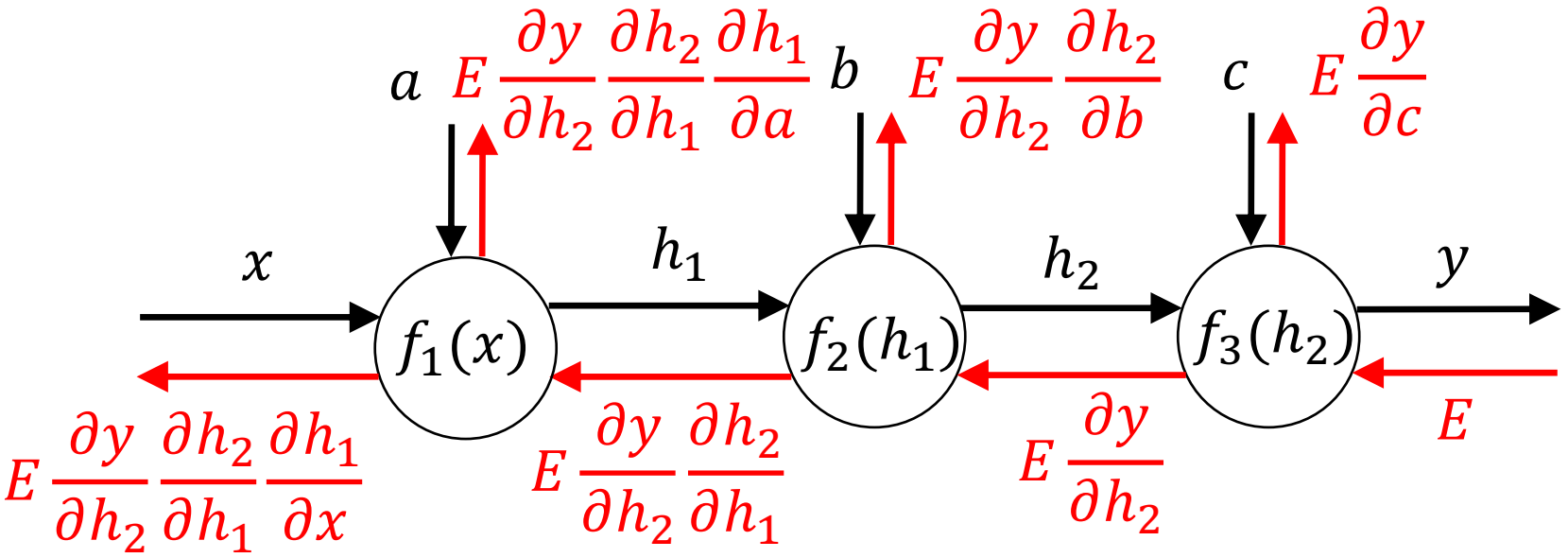
計算グラフの一般化(2)

- 出力yに対する各パラメータの偏微分は下記の図で表現可能
  - ◆出力から各ノードで発生する偏微分



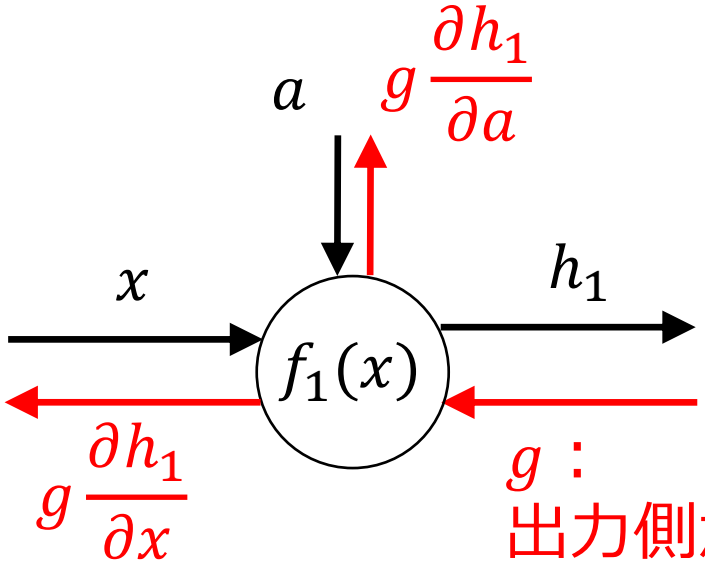
計算グラフの一般化(3)

- 計算グラフはノードが何個繋がっていても偏微分の値を入力側に伝播するだけでいい
  → 逆伝播という



計算グラフの一般化(4)

- 一つのノードに着目すると，出力側からの偏微分値に自身の偏微分値を伝えれば良い




```
class Function1():
    def __init__(self):
        self.a = # 乱数
        self.lr = # 学習率

    def forward(self, x):
        return # h1の計算

    def backward(self, g):
        self.g_a = # 偏微分値の計算
        self.a = # 更新式
        return # g*g_h1を返す
```

伝播してきた微分値に
自身の偏微分をかけて入力側に伝える

ニューラルネットワーク
単純パーセプトロン

表記の注意

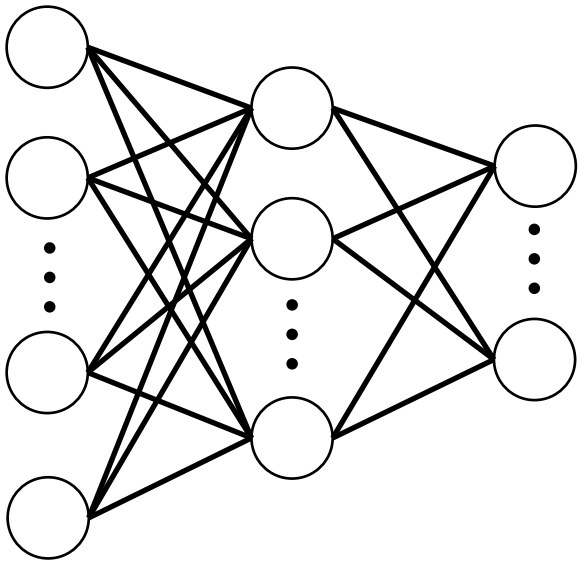
- 内積演算（積和演算）
  - $\boldsymbol{x} = [x_0, x_1, \cdots x_n]^T$, $\boldsymbol{w} = [w_0, w_1, \cdots w_n]^T$の各要素の積の総和
  - 機械学習では内積が頻出

$$y = \sum w_i x_i = [x_0, x_1, \cdots x_n] \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} = \boldsymbol{x}^T \boldsymbol{w}$$

太字で書いてあるので注意

ニューラルネットワーク

- 脳の神経細胞（ニューロン）と
  シナプスによる結合（神経回路網）による
  ネットワークを数理モデルで表現したもの
- 現代のAI技術の多くがニューラルネットワーク



入力層　　中間層
（隠れ層）　　出力層

：ニューロン（ユニット，ノード）
データ（値）が入っている

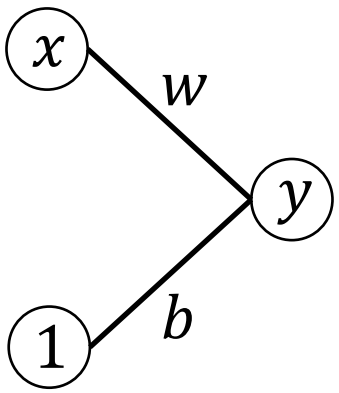
：シナプス（エッジ）
結合荷重に相当する

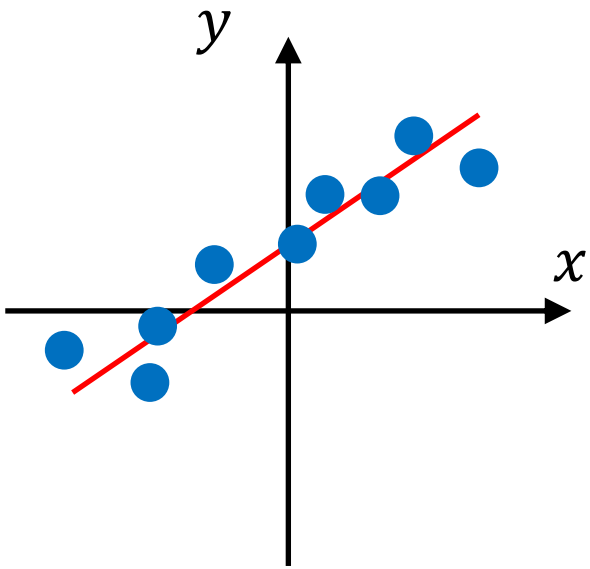
ニューラルネットワークの回帰

■ 例：1入力1出力を考える

$$y = wx + b \rightarrow \text{直線の式}$$

NNはw，bの値を学習(更新) → 傾きと切片の調整
回帰ではデータに当てはまる線を学習している





ニューラルネットワークの分類

■ 例：2入力1出力を考える

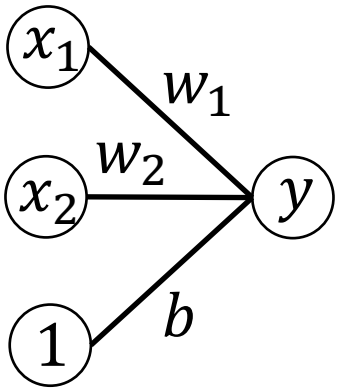
$$y = w_1 x_1 + w_2 x_2 + b$$

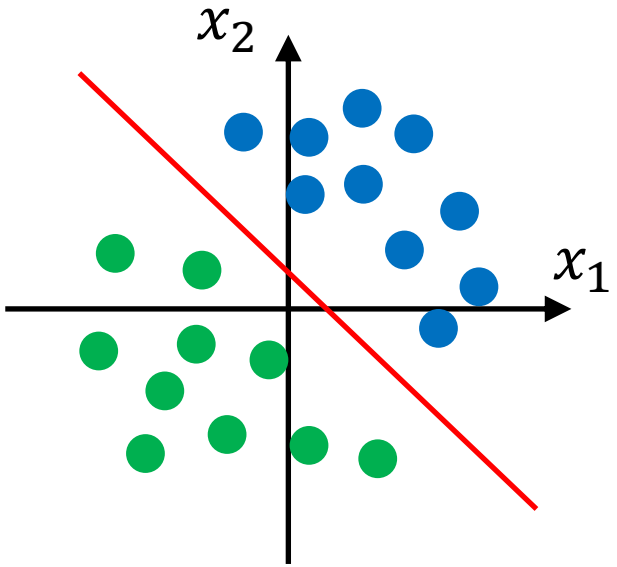
仮に$y = 0$として式を整理する → 直線の式

$$x_2 = \frac{w_1}{w_2} x_1 + \frac{1}{w_2} b$$

NNはw_i，bの値を学習(更新) → 傾きと切片の調整

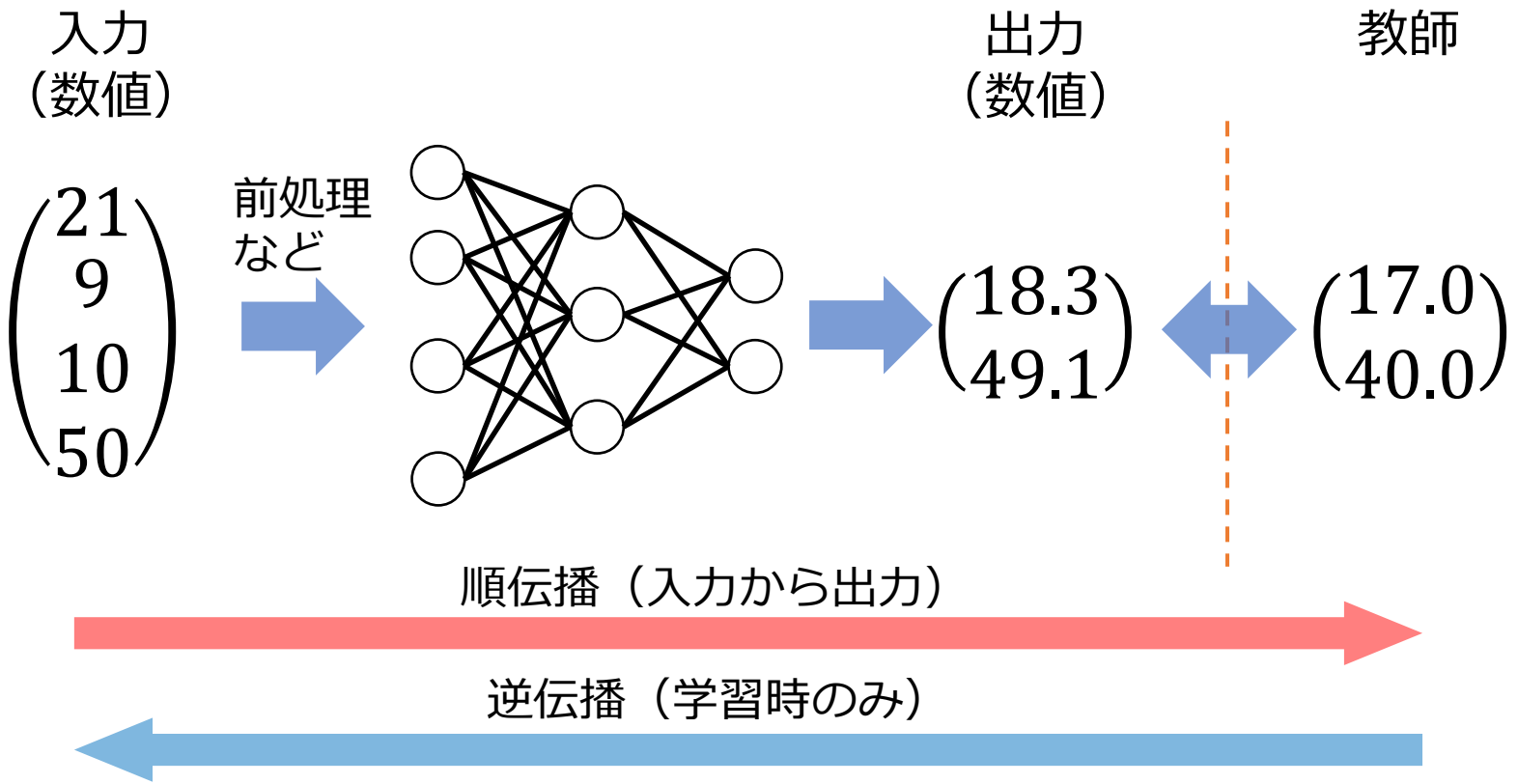
分類はデータを切り分ける線を学習





NNの入出力（回帰）

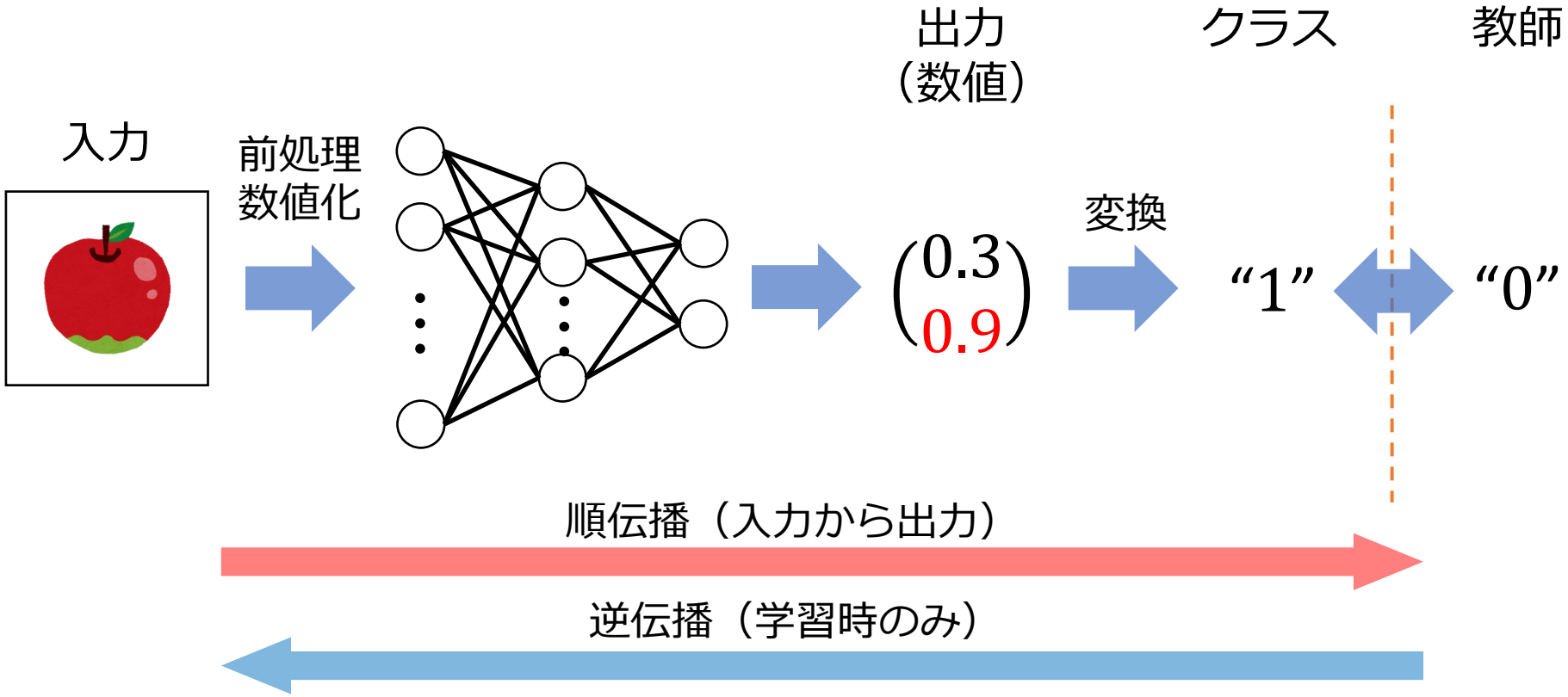
- 入力は今日の気象データ（数値），
  出力は次の日の気温と湿度（数値回帰）の例



※実際は前処理で正規化や標準化をする

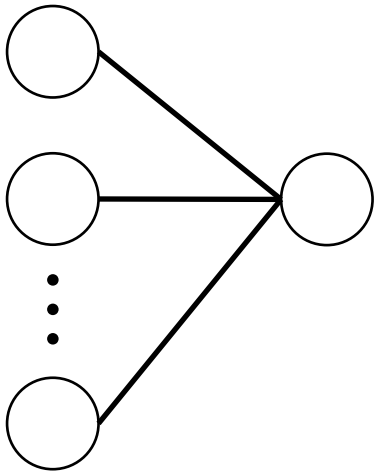
NNの入出力（分類）

- 入力は画像，出力はクラス分類の例
  （クラス”0”はりんご，“1”はみかん）



単純パーセプトロン

- 複数の入力と単一の出力を持つ
  - ◆線形分離可能な問題にしか対応できない
  - ◆出力1つで1つの直線→多出力なら非線形
- 入力を $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]^T$, バイアスを b
  重みを $\boldsymbol{w} = [w_1, w_2, \cdots, w_n]$とすると出力$y$は



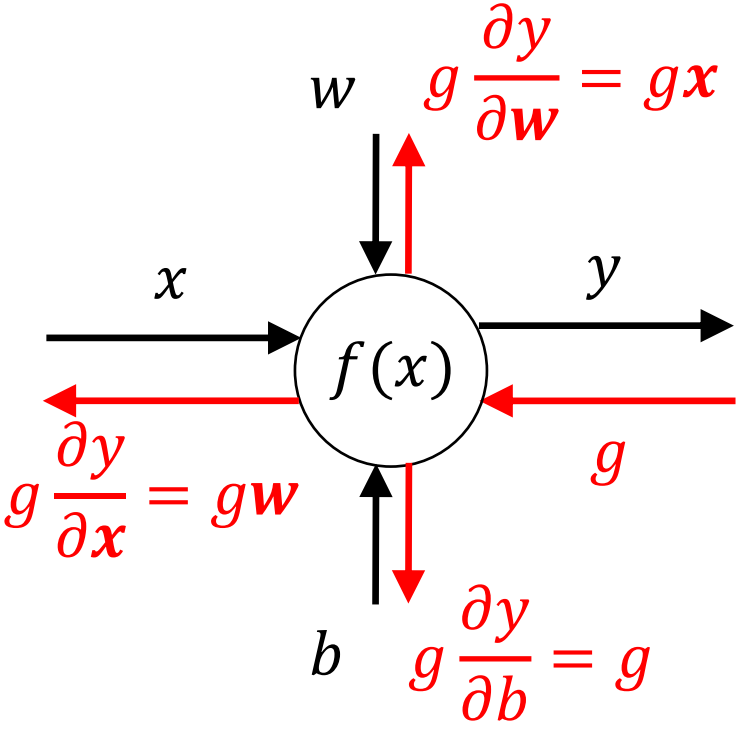
入力層　　出力層

$$y = [x_1, x_2, \cdots, x_n] \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} + b$$

$$= \boldsymbol{x}^T \boldsymbol{w} + b$$

順伝播の計算グラフ

■ $y = f(x) = \boldsymbol{wx} + b$の計算グラフ



計算グラフの実装イメージ

```
class Perceptron():
    def __init__(self): ← インスタンス生成時に
        self.w = # 乱数     呼び出される関数
        self.b = # 乱数
        self.lr = # 学習率

    def forward(self, x):
        self.x = x ← 後で勾配計算に使う
        return # wx + b

    def backward(self, g): ← 勾配算出と更新
勾配 self.g_w = # 偏微分値の計算
計算 self.g_b = # 偏微分値の計算
        g = # gwを計算
更新 self.w = # 更新式
     self.b = # 更新式
        return # gを返す ← 次のノードへ
```

self.変数名：メンバ変数クラス内で共通の値として参照可能

平均二乗誤差関数と逆伝播(1)

- 平均二乗誤差関数は回帰によく使う
  - ◆前回の表記を少し変える
  - ◆Nをデータ数，yを推定値，tを正解値とすると

$$E = \frac{1}{N}\sum_{i}(y_i - t_i)^2$$

- ある1つのデータiだけに注目すると

$$E_i = (y_i - t_i)^2$$

- 全体損失関数として平均二乗誤差は

$$E = (E_1 + E_2 + \cdots E_n)/N = \frac{1}{N}\sum E_i$$

平均二乗誤差関数と逆伝播(2)

- 重みもバイアスも全てのデータに対して共通
- パラメータ更新のため,
  全体損失関数Eの偏微分を考えると

$$\frac{\partial E}{\partial w} = \left(\frac{\partial E}{\partial E_1}\frac{\partial E_1}{\partial w} + \frac{\partial E}{\partial E_2}\frac{\partial E_2}{\partial w} + \cdots + \frac{\partial E}{\partial E_n}\frac{\partial E_n}{\partial w}\right)$$

- $\partial E/\partial E_i = 1$なので

$$\frac{\partial E}{\partial w} = \left(\frac{\partial E_1}{\partial w} + \frac{\partial E_2}{\partial w} + \cdots + \frac{\partial E_n}{\partial w}\right) = \sum \frac{\partial E_i}{\partial w}$$

→ 誤差関数から逆伝播する際は総和を計算せず,
パラメータの更新時に総和をとる
（総和ではなく平均でも良い）

平均二乗誤差関数と逆伝播(3)

- E_iに対する出力yでの偏微分$\partial E_i / \partial y$は

$$\frac{\partial E_i}{\partial y} = 2(y_i - t_i)$$

- 実装イメージ

```
class MeanSquaredError():
    def __init__(self):
        self.diff = None
        self.loss = None

    def forward(self, y, t):
        self.diff = y - t ← 逆伝播で使う
        self.loss = # 誤差計算
        return self.loss

    def backward(self):
        return # 偏微分値を返す ← 総和や平均を取らない
```

self.変数名：メンバ変数クラス内で共通の値として参照可能

複数のデータに対応（バッチ対応）

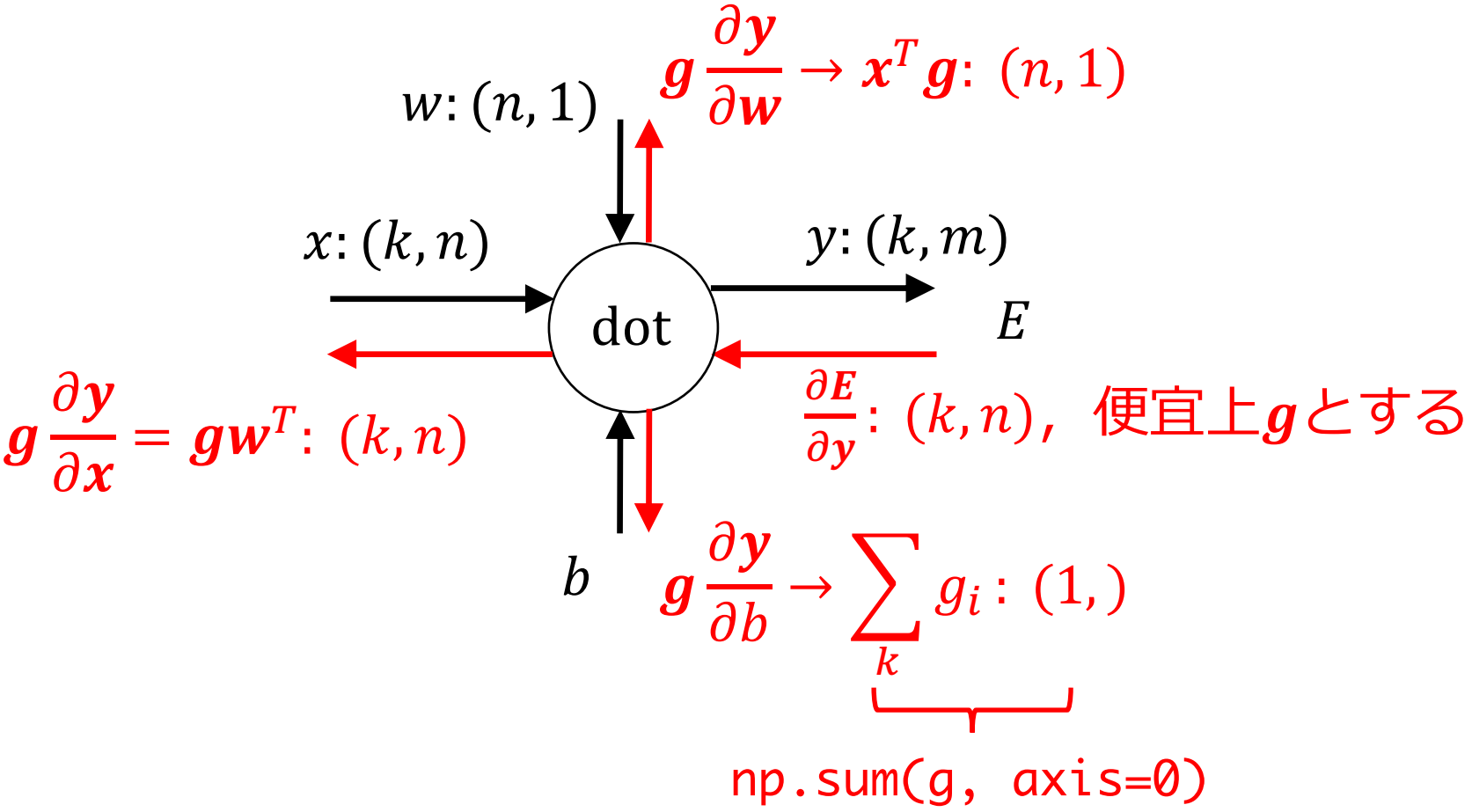
- 実際はデータセット，k個のデータがあるとする
  - ◆$\boldsymbol{x}$，$\boldsymbol{y}$の行方向に定義
  - ◆入力の次元数は(k, n)：(データ数, 入力数)
  - ◆重みの次元数は$(n, 1)$：(入力数, 出力数)
  - ◆出力の次元数は$(k, 1)$：(データ数, 出力数)
- 内積計算で表現

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} x_{1,1}, x_{1,2}, \cdots x_{1,n} \\ x_{2,1}, x_{2,2}, \cdots x_{2,n} \\ \vdots \\ x_{k,1}, x_{k,2}, \cdots x_{k,n} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} + b$$

$$\boldsymbol{y} = \boldsymbol{x}\boldsymbol{w} + b$$

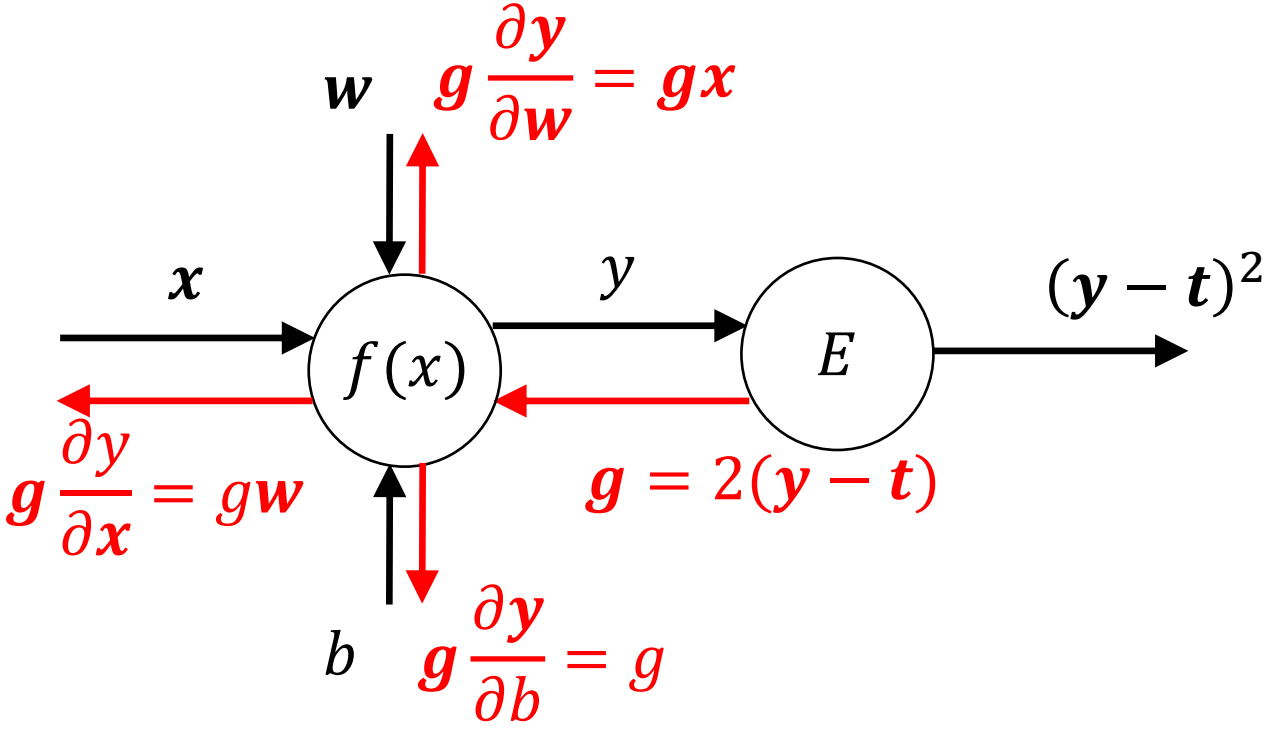
バッチ対応版の計算グラフ

■ ()は次元数



単純パーセプトロンの計算グラフ全体

■ 二乗誤差を使う場合



勾配法による更新の手順のイメージ

1. 順伝播→誤差計算
2. 誤差関数の逆伝播→パーセプトロンの逆伝播

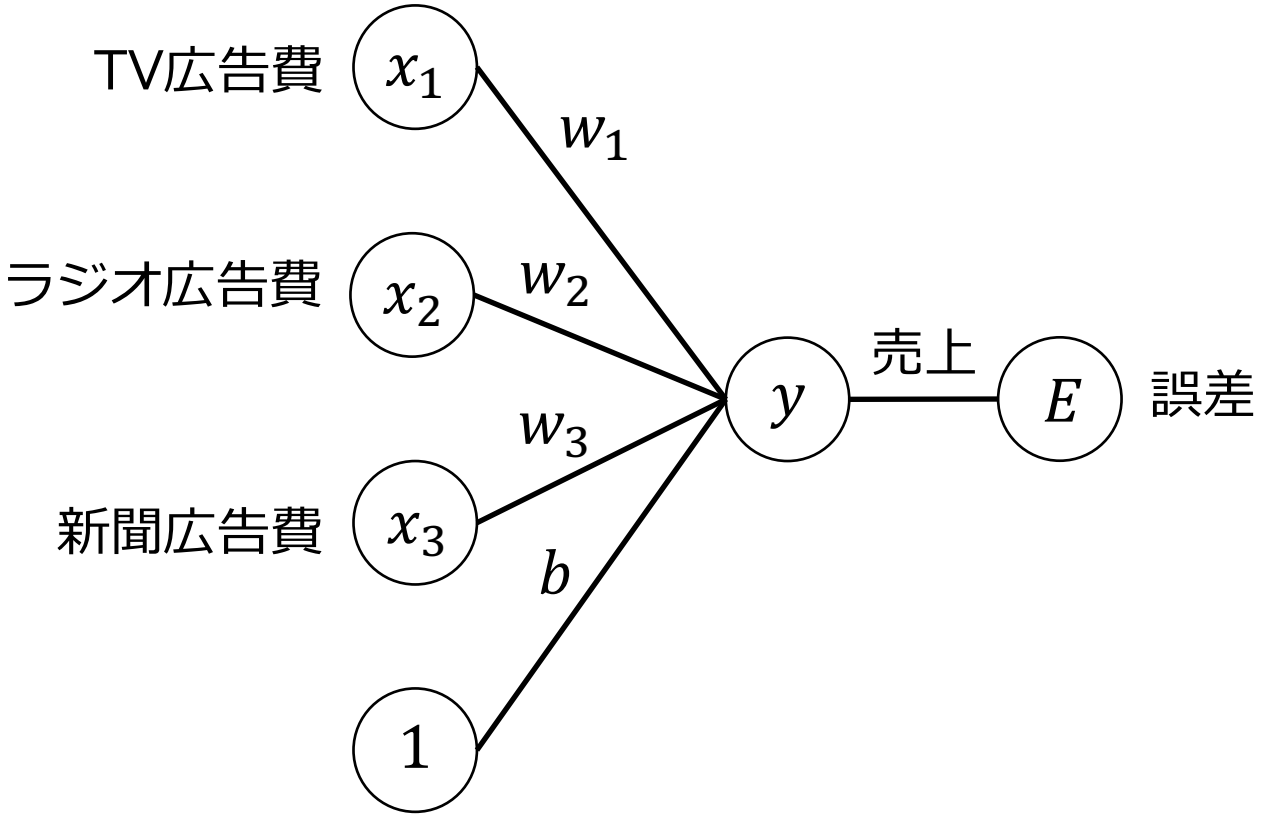
```
for i in range(max_iteration):
    # 順伝播
    y = perceptron.forward(x)

    # 誤差計算
    loss = loss_function.forward(y, t)

    # 誤差関数の勾配を計算
    g = loss_function.backward()

    # 誤差逆伝播で勾配を計算し更新
    g = perceptron.backward(g)
```

演習(4)：単純パーセプトロン

- 単純パーセプトロンにより，
  係数を求めるプログラムを完成させよ.
  - ◆データセットはdata/advertising.csvを使う
  - ◆説明変数：TV広告費、ラジオ広告費、新聞広告費
  - ◆目的変数：Sales（売上）
- 演習で使って良いnumpyの関数など
  - ◆ガウス分布で乱数生成（配列）：
    - 重み：rng.standard_normal((入力次元数, 出力次元数))
    - バイアス：rng.standard_normal(出力次元数)
  - ◆合計：np.sum()
  - ◆平均：np.mean()
  - ◆累乗：np.pow()，あるいは「変数名 ** 2」
- プログラムの実行（/scripts内で実行）
  $ uv run perceptron.py

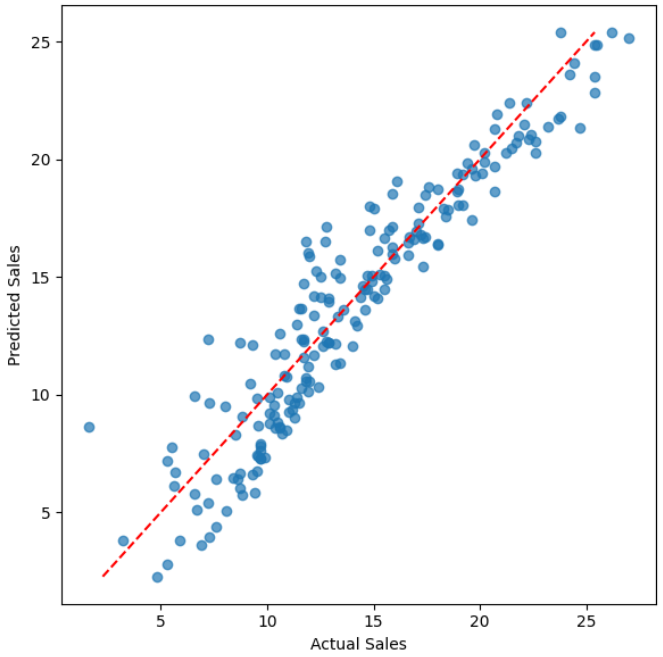
演習(4)のイメージ

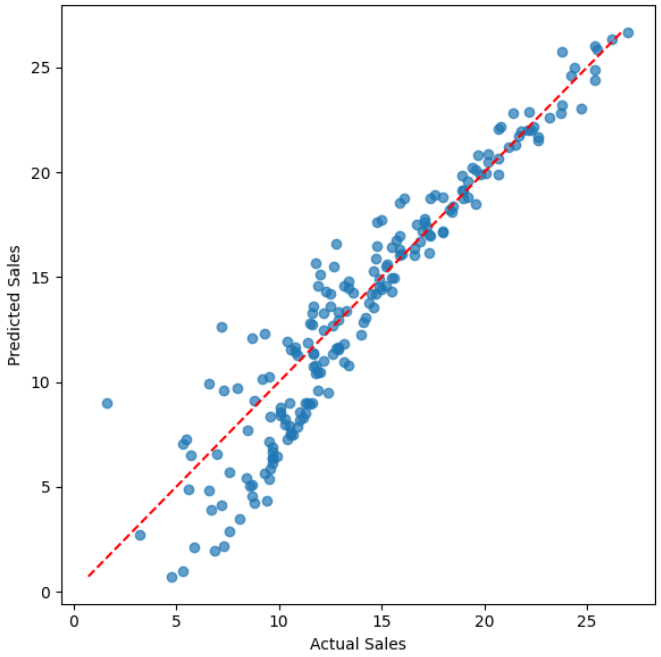
- パーセプトロンと誤差関数で
  それぞれのクラスを作成している



実行後の結果は毎回異なる

- 重みとバイアスの初期値が乱数のため，毎回収束した値が異なる
- 学習率や学習回数でも異なる





$$w = \begin{bmatrix} 0.05082344, \\ 0.17614651, \\ 0.02445359 \end{bmatrix}$$

$$b = [1.4407811]$$

$$b = -0.23855995 w = \begin{bmatrix} 0.05429334 \\ 0.22951969 \\ 0.01574611 \end{bmatrix}$$

課題・考察

正規化，標準化

■ 正規化（Normalization）

◆データを0～1に変換

- 最大値や取りうる最大値で割る

$$x_n = \frac{x}{x_{\max}}$$

- Min-Max法：最小値を0，最大値を1に変換

$$x_n = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

■ 標準化（Standardization）

◆データを平均0、標準偏差1に変換

$$x_n = \frac{x - \mu_x}{\sigma_x}$$

課題・考察(2)

- perceptron.pyにおいて，学習データxと教師データtを正規化か標準化できるように改良する．その後，正規化や標準化によって重み，出力結果，学習率，学習回数がどのように変化したか考察し述べよ．
  また，正規化や標準化がなぜ有効なのか調べよ．
  プログラムはex_02_学籍番号.pyとして作成せよ．

- ヒント：正規化や標準化にはnp.maxやnp.stdを用いると良い