

# 深層学習入門

---

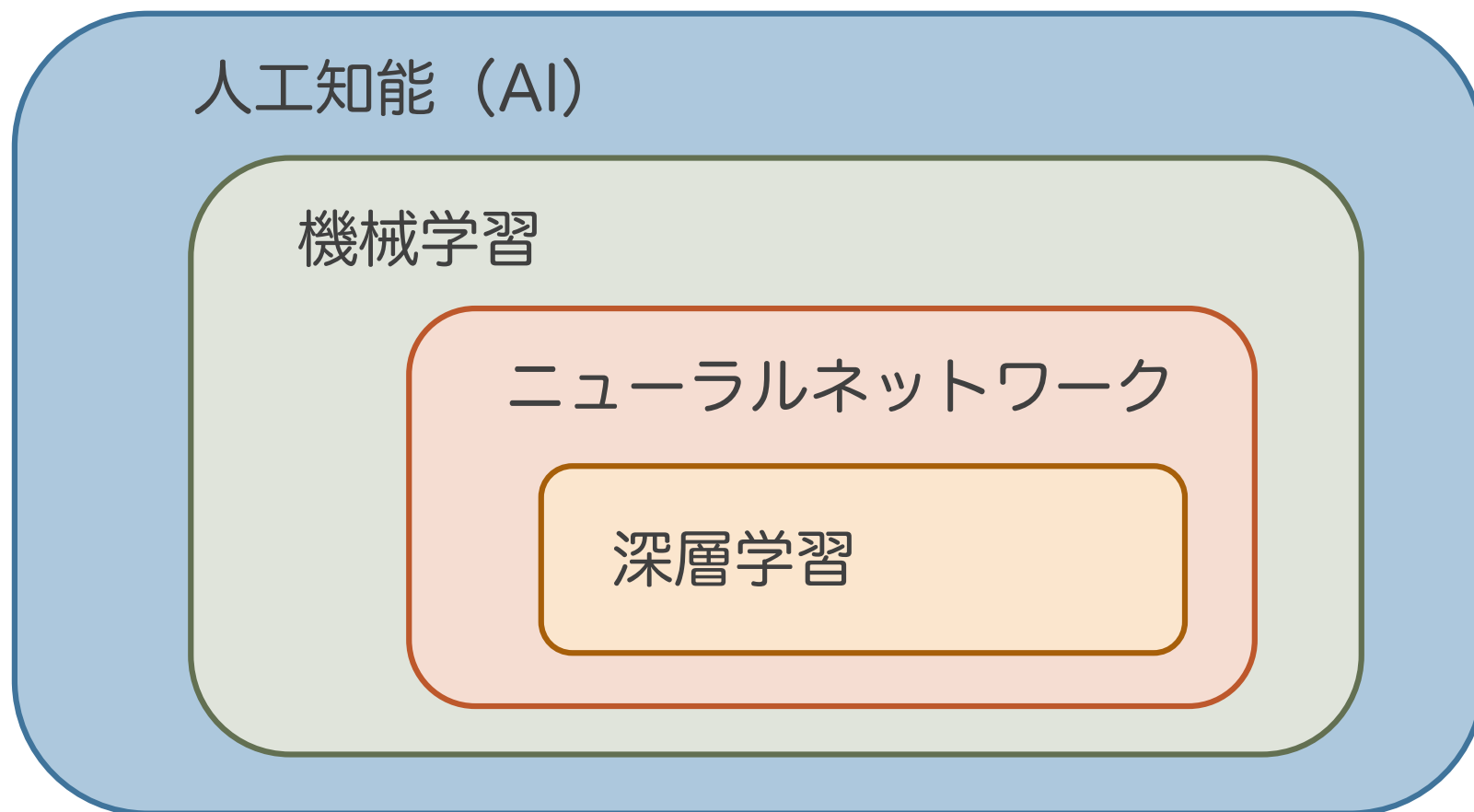
ニューラルネットワークの概要

一関高専 未来創造工学科 情報・ソフトウェア系

小池 敦

# 概要

---



# ニューラルネットワーク

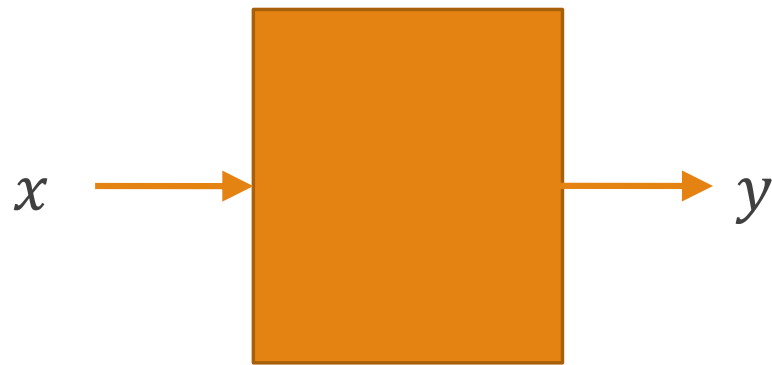
---

- 脳の神経細胞（ニューロン）のネットワークを参考に作られた数理モデル
- 2010年ごろまではそれほど高性能ではなかった
- 2012年の画像認識コンテストILSVRCでディープラーニングを使用したチームが圧勝し、その頃から広く活用されるようになった

# ニューラルネットワークの基礎

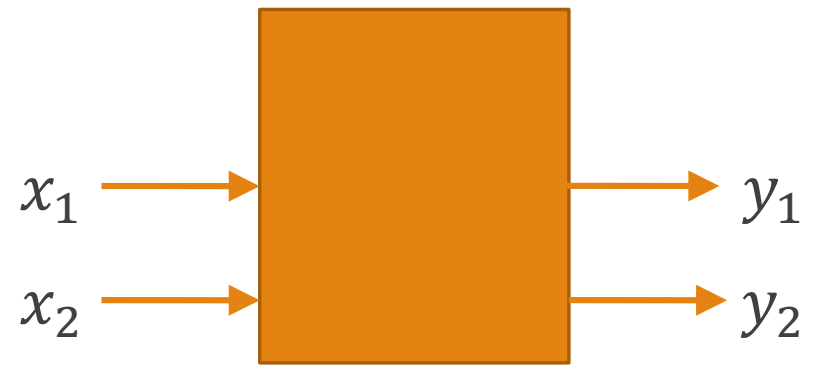
- 1ネットワークで入力 $x$ と出力 $y$ の関係を表す
  - 基本的には $y = f(x)$ のような関数になる

$$y = f(x)$$



1入力1出力

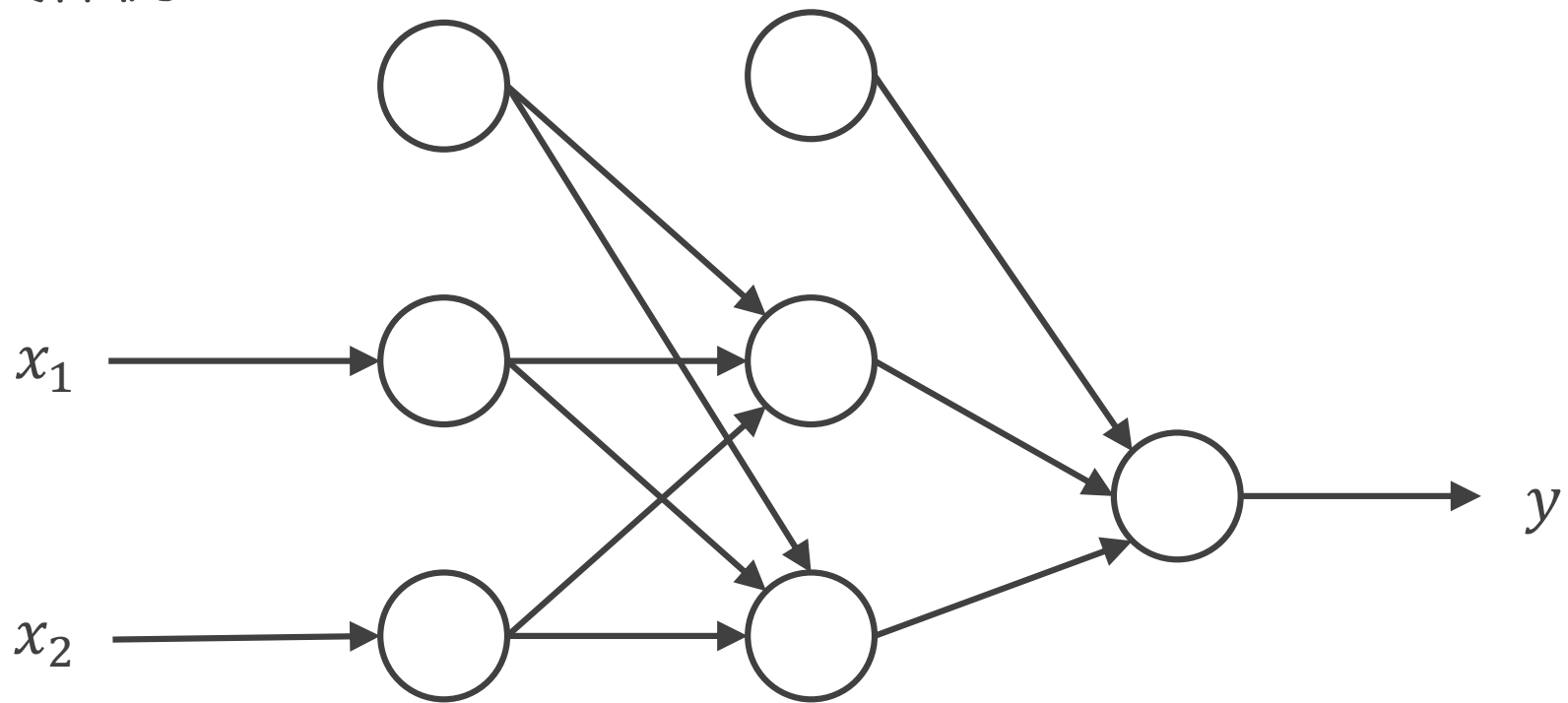
$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = f \left( \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right)$$



2入力2出力  
(入出力とも2次元ベクトル)

# ニューラルネットワークの基礎

具体例

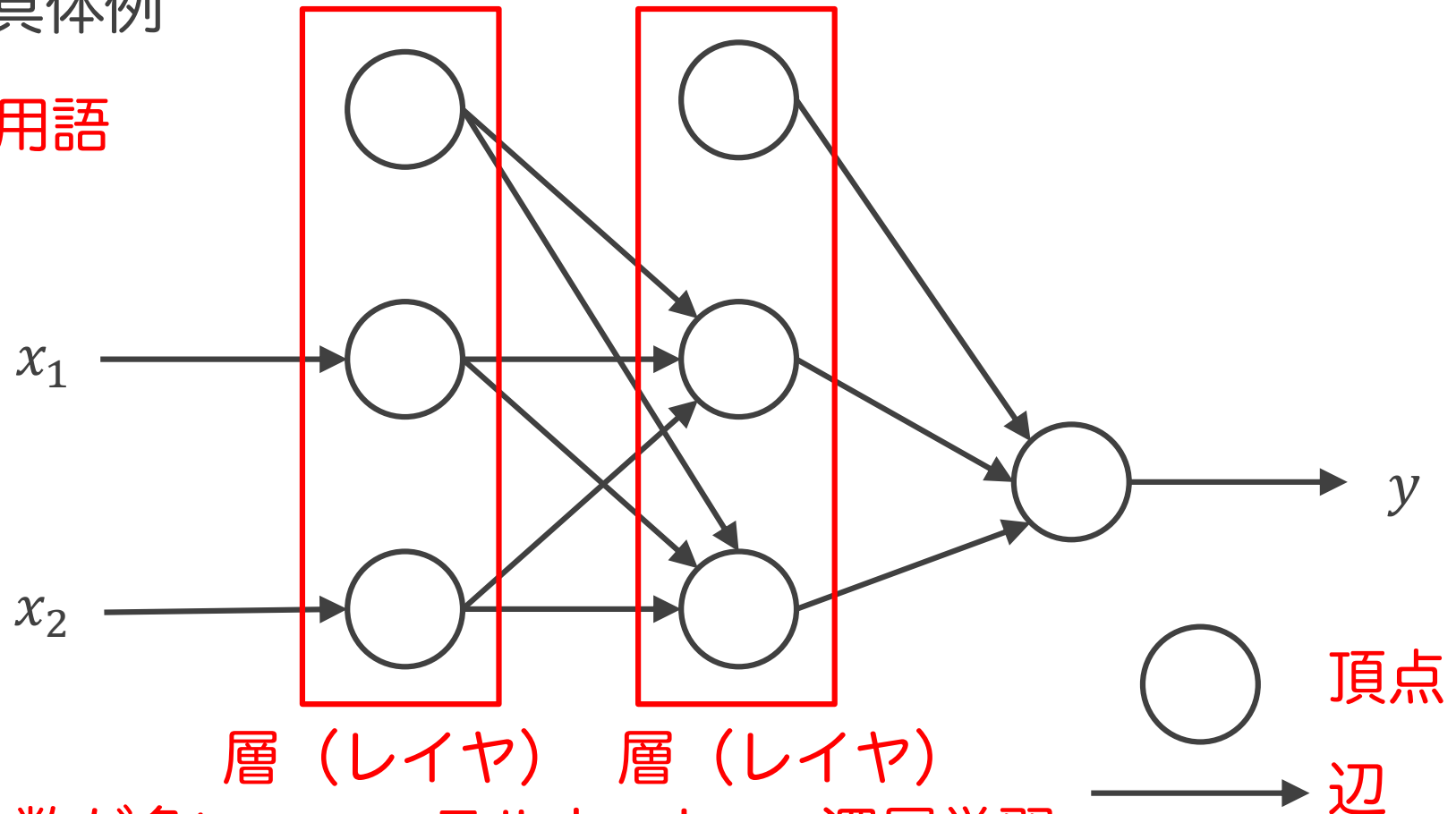


これからこの意味を説明する

# ニューラルネットワークの基礎

具体例

用語

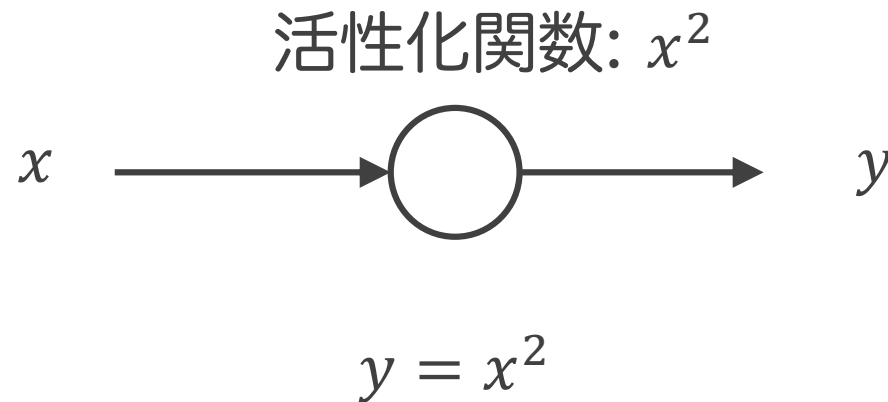


層 (レイヤ) 層 (レイヤ)  
層の数が多いニューラルネットワーク ⇒ 深層学習

# ニューラルネットワークの基礎

---

頂点は関数を表す（活性化関数と呼ぶ）  
活性化関数は1引数，1出力



※ 実際にはこんな活性化関数は使わない

# ニューラルネットワークの基礎

---

頂点への入力がない時は定数1を返す関数  
(バイアス項と呼ばれる)



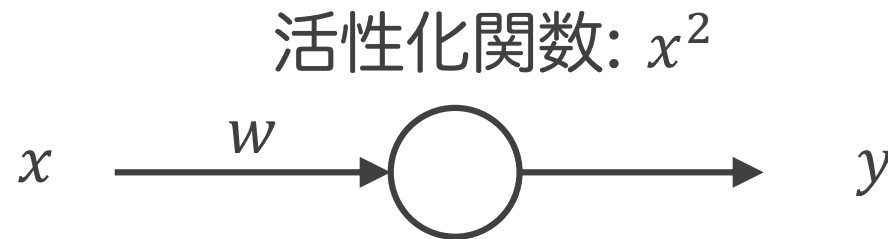
$$y = 1$$



# ニューラルネットワークの基礎

---

辺には重みを設定できる  
重みの値で定数倍される

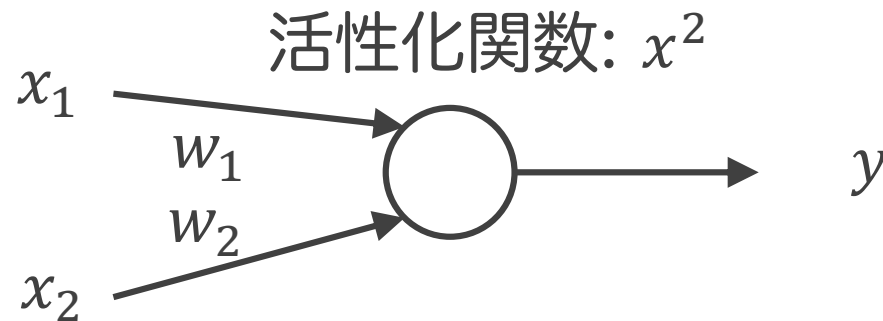


$$y = (wx)^2$$

# ニューラルネットワークの基礎

---

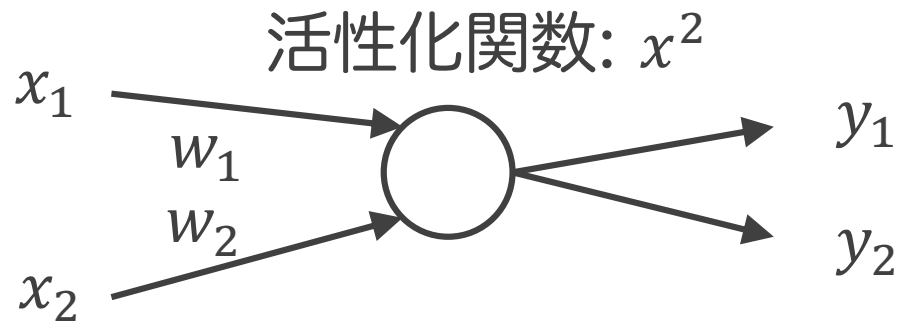
頂点への入力複数ある時、それらは加算される



$$y = (w_1x_1 + w_2x_2)^2$$

# ニューラルネットワークの基礎

頂点からの出力が複数ある時, それらは同じ値になる



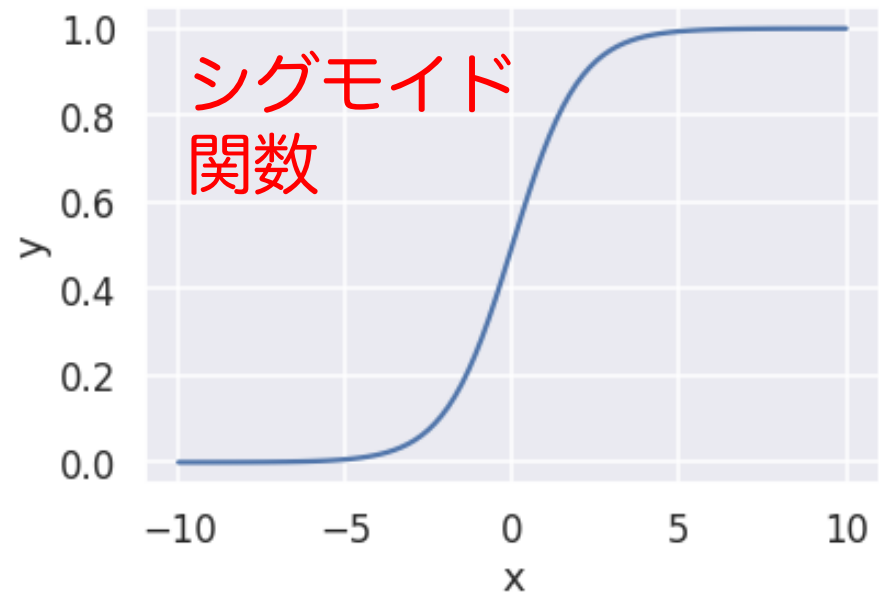
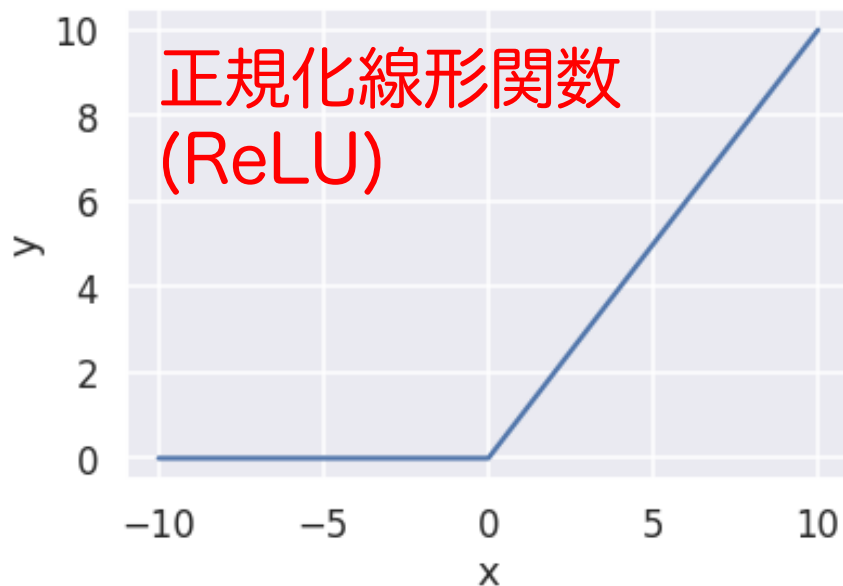
$$y_1 = (w_1x_1 + w_2x_2)^2$$

$$y_2 = (w_1x_1 + w_2x_2)^2$$

# ニューラルネットワークの基礎

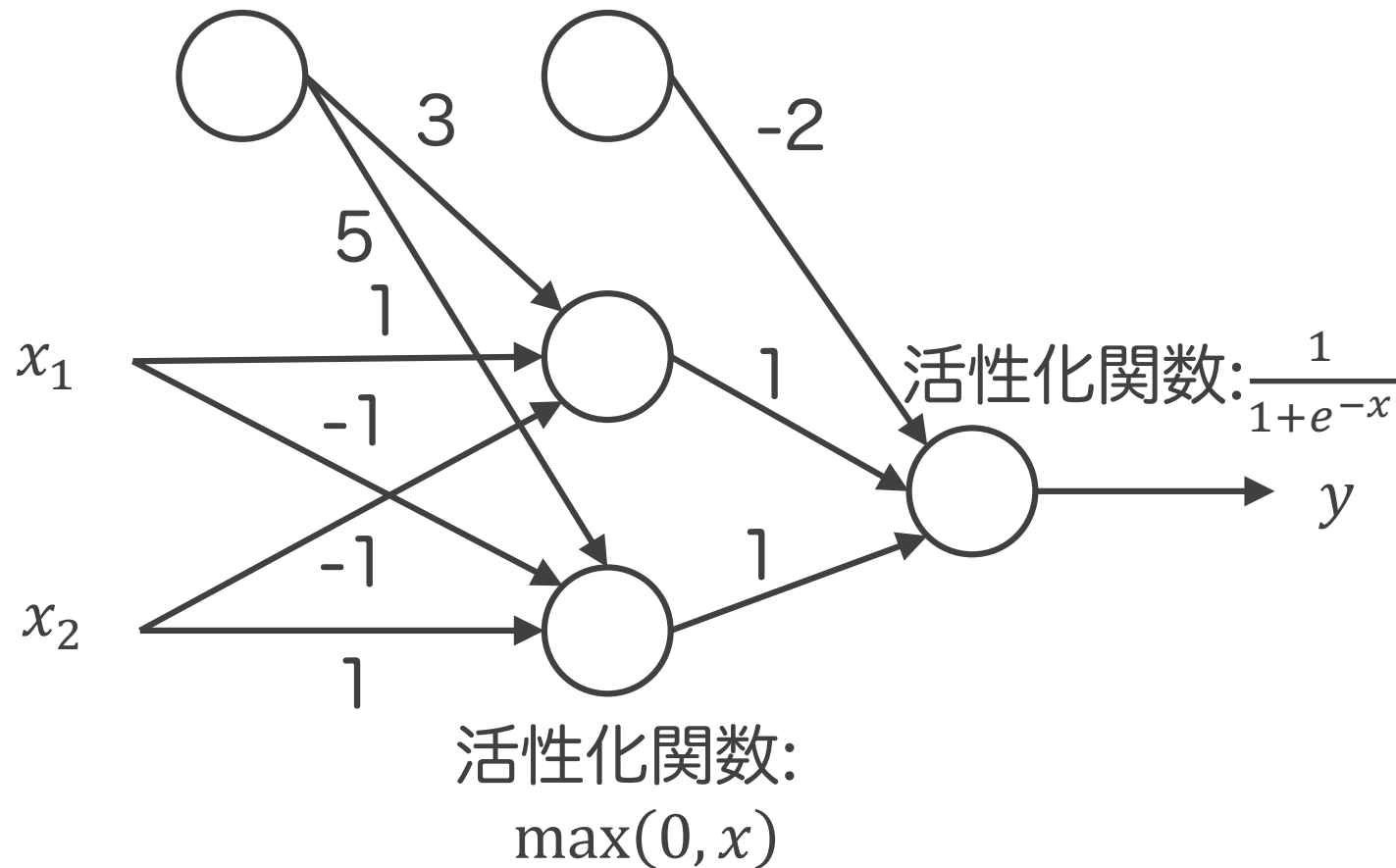
## 活性化関数

- 同一層の頂点は、基本的に同じ活性化関数を使う
- 活性化関数には正規化線形関数がよく使われる  
ReLU:  $y = \max(0, x)$
- シグモイド関数 (sigmoid) も使われる



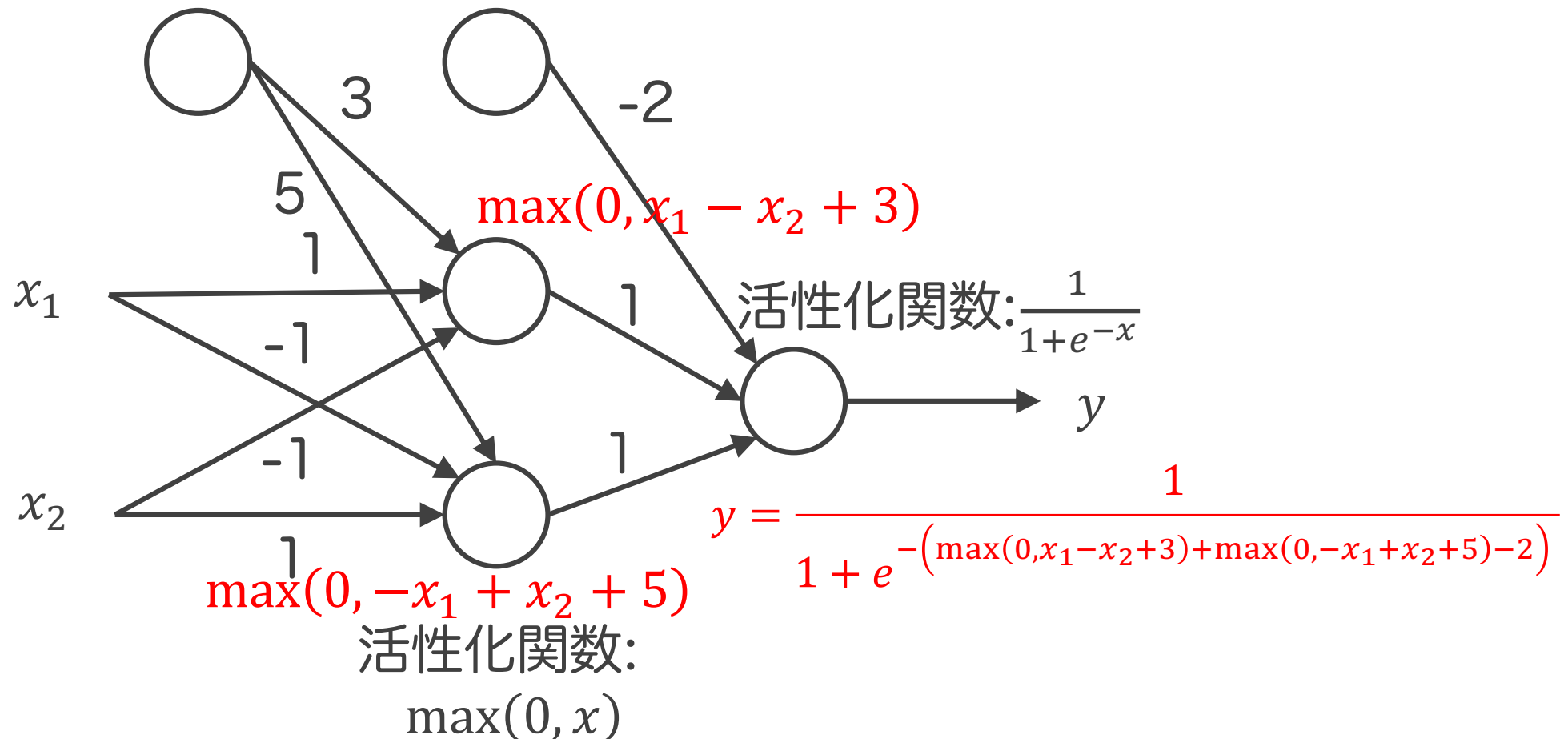
# ニューラルネットワークの基礎

練習問題：次のニューラルネットワークが表す関数は？

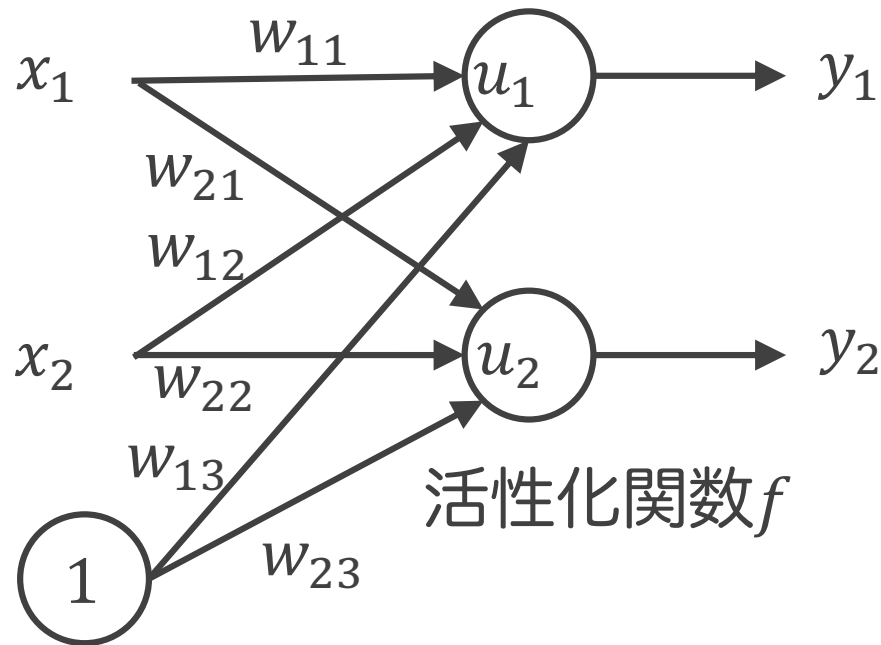


# ニューラルネットワークの基礎

練習問題：次のニューラルネットワークが表す関数は？



# ニューラルネットワークの 行列表現



$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

$$f\left(\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}\right) = \begin{bmatrix} f(u_1) \\ f(u_2) \end{bmatrix} \text{と定義すると}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = f\left(\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}\right)$$

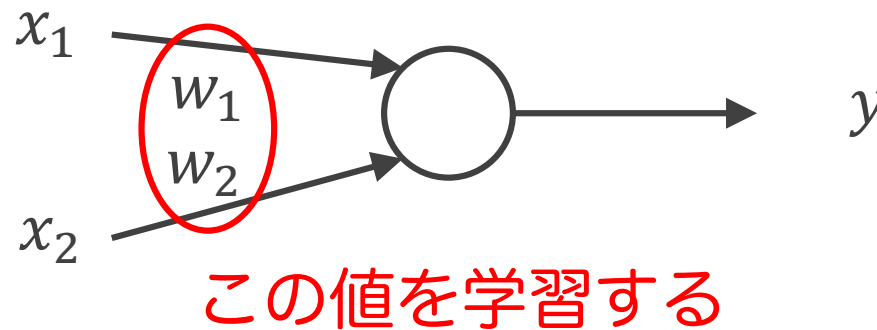
$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ は活性化関数  $f$   
への入力値

ニューラルネットワークは  
行列積と活性化関数の繰り返し  
として記述できる

# ニューラルネットワークの学習

---

- 辺の重みを学習する
- 活性化関数の部分は変えない
- 誤差最小を目指して繰り返し重みを変化させる
  - 1ターンのことをエポック(epoch)と呼ぶ





# 辺の重みの学習

---

- 全部の辺の重みを個別に調整する
  - 複数の辺で連携したりはしない
  - 重みの初期値は適当に決める
  - 誤差を辺の重みで偏微分し，その逆方向に変化させる
  - ある辺の重み $w$ の $t$ 回目の更新の値を $w_t$ とすると  
は以下のように計算される（勾配法の場合）

$$w_{t+1} \leftarrow w_t - \alpha \cdot \left. \frac{\partial E}{\partial w} \right|_{w=w_t}$$

- $E$ は誤差関数， $\alpha$ は学習率と呼ばれユーザが事前に決める値

# 誤差逆伝播法

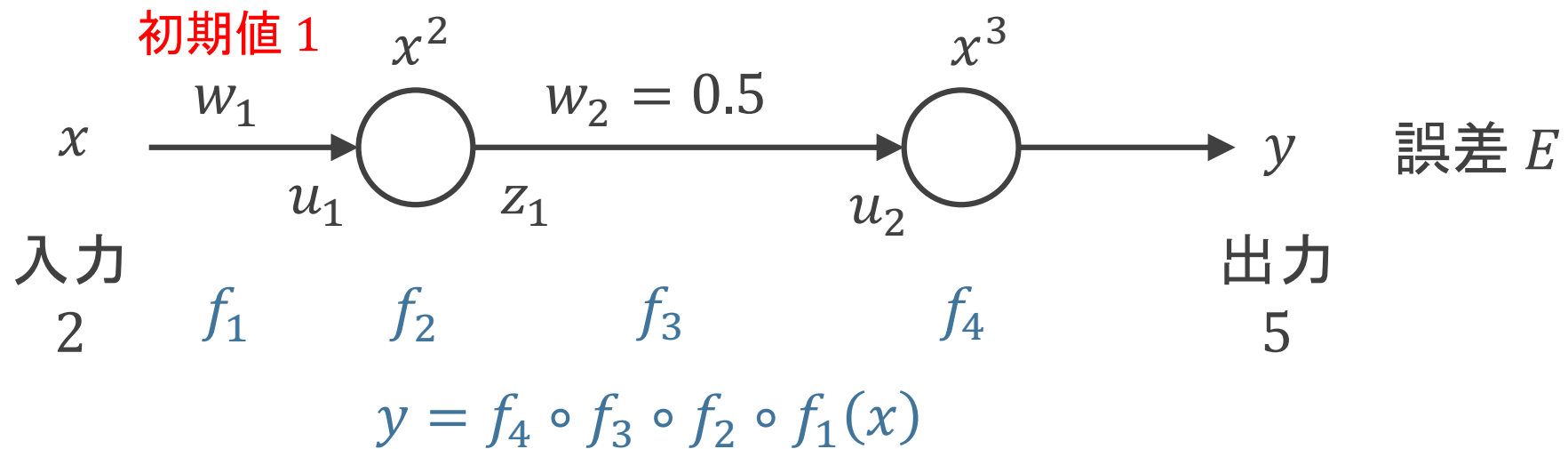
---

- 「誤差を各辺の重みで偏微分する」計算を高速に行う手法
- 出力層の誤差（誤差の偏微分）を入力層方向に伝播させる  
⇒ 誤差逆伝播法（バックプロパゲーション）と呼ぶ

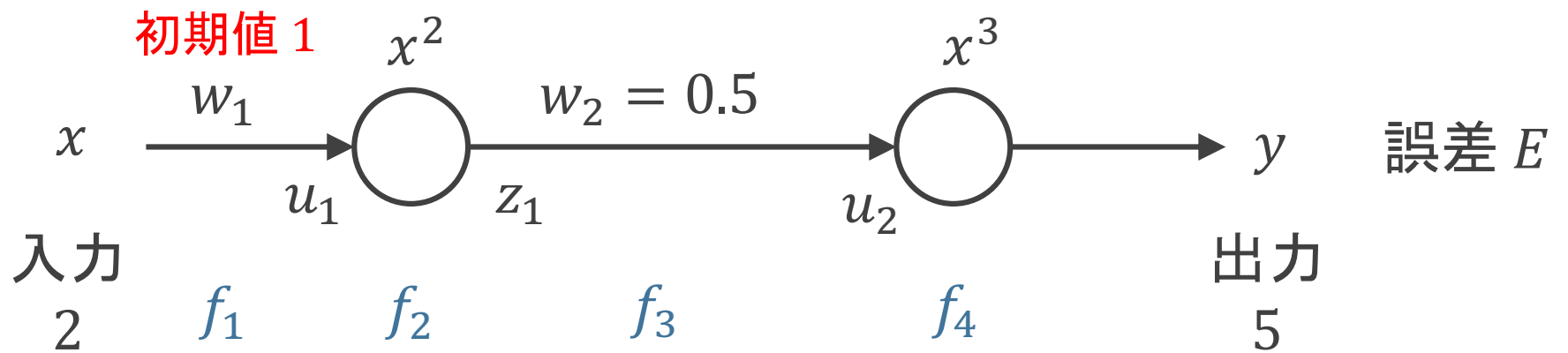
簡単な例題で計算してみる

# 例題

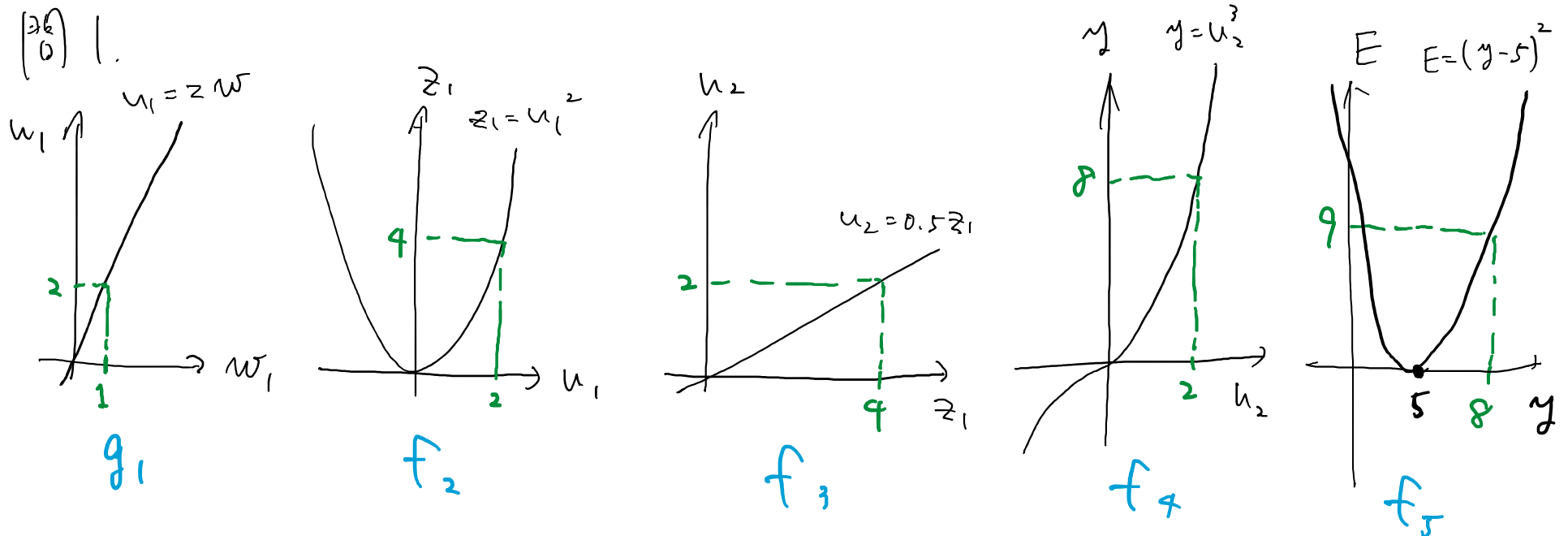
- $x = 2$  の時,  $y = 5$  となるように  $w_1$  を求めたい



- 問1.  $w = 1$  の時, 出力の二乗誤差を求めなさい
- 問2.  $w = 1$  の時,  $\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}$  を求めなさい



$$y = f_4 \circ f_3 \circ f_2 \circ f_1(x)$$

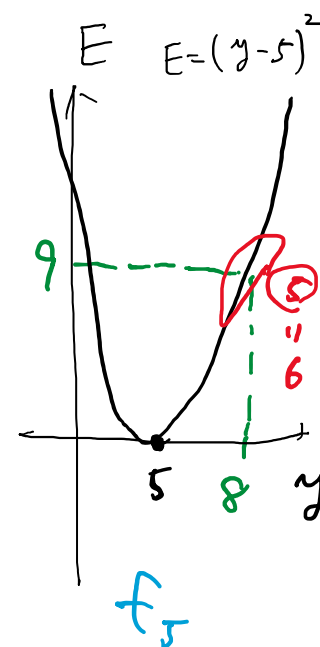
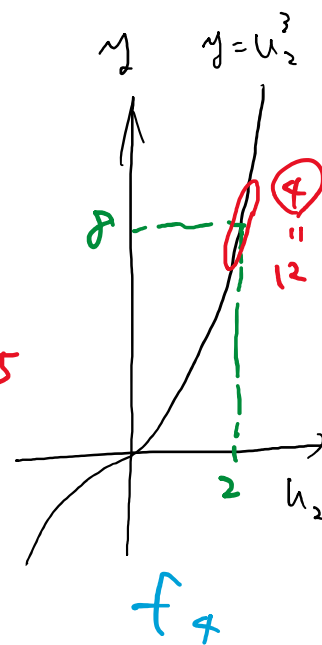
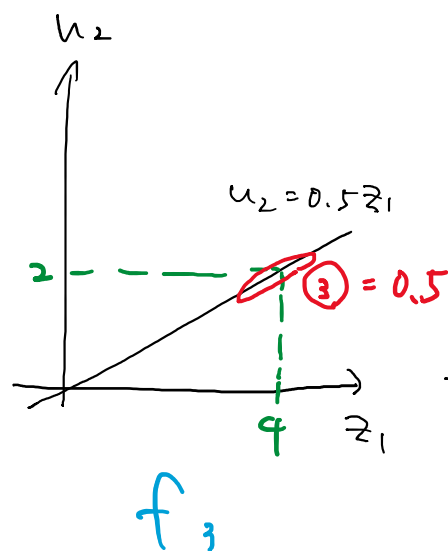
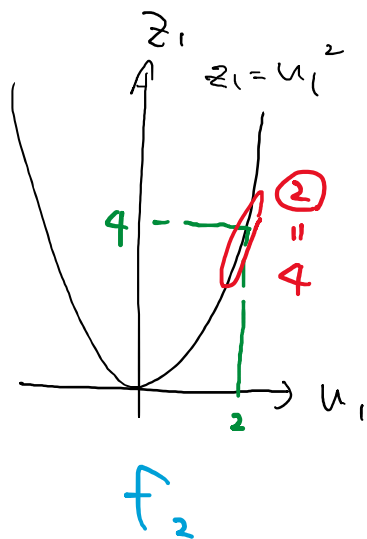
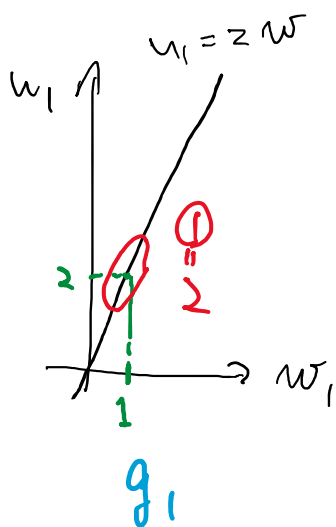


$$E = f_5 \circ f_4 \circ f_3 \circ f_2 \circ g_1(w_1) \quad w_1 = 1 \Rightarrow E = 9$$

問2. 合成関数の微分公式より、

$$\frac{\partial E}{\partial w_1} = \underbrace{\frac{\partial u_1}{\partial w_1}}_{(1)} \cdot \underbrace{\frac{\partial z_1}{\partial u_1}}_{(2)} \cdot \underbrace{\frac{\partial u_2}{\partial z_1}}_{(3)} \cdot \underbrace{\frac{\partial y}{\partial u_2}}_{(4)} \cdot \underbrace{\frac{\partial E}{\partial y}}_{(5)}$$

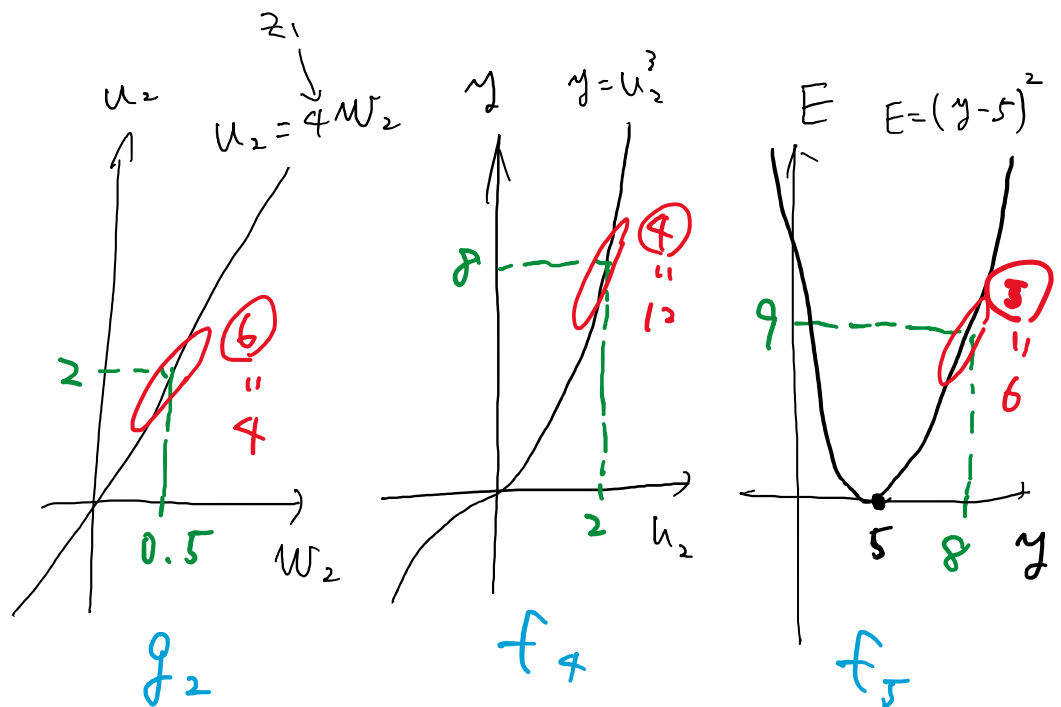
$$= 2 \cdot 4 \cdot 0.5 \cdot 12 \cdot 6 = 288$$



$$\frac{\partial E}{\partial w_2} = 12 u_2. \quad E = f_5 \circ f_4 \circ g_2(w_2) \quad \text{with } 1+2.$$

$$\therefore \frac{\partial E}{\partial w_2} = \underbrace{\frac{\partial u_2}{\partial w_2}}_{(6)} \cdot \underbrace{\frac{\partial y}{\partial u_2}}_{(4)} \cdot \underbrace{\frac{\partial E}{\partial y}}_{(5)}$$

$$= 4 \cdot 12 \cdot 6 = 288.$$



# 例題まとめ

---

- 問1は入力から出力に向けて順に計算していけば解ける  
⇒ この処理を順伝播と呼ぶ
- 問2は順伝播の後、各関数の偏微分を求める。  
その際 $w_1$ の計算には、 $w_2$ の計算結果が使える。  
⇒ 出力から入力に向けて計算することで、  
効率よく求めることができる  
⇒ 各辺の偏微分を求める上記の手順を  
誤差逆伝播法と呼ぶ

# もっと複雑な場合

---

- データが複数ある場合
  - データごとの誤差の偏微分を足し算すれば良い.
  - データを $N$ 個, データ $k$ での誤差を $E_k$ とすると誤差の偏微分は以下のように計算できる

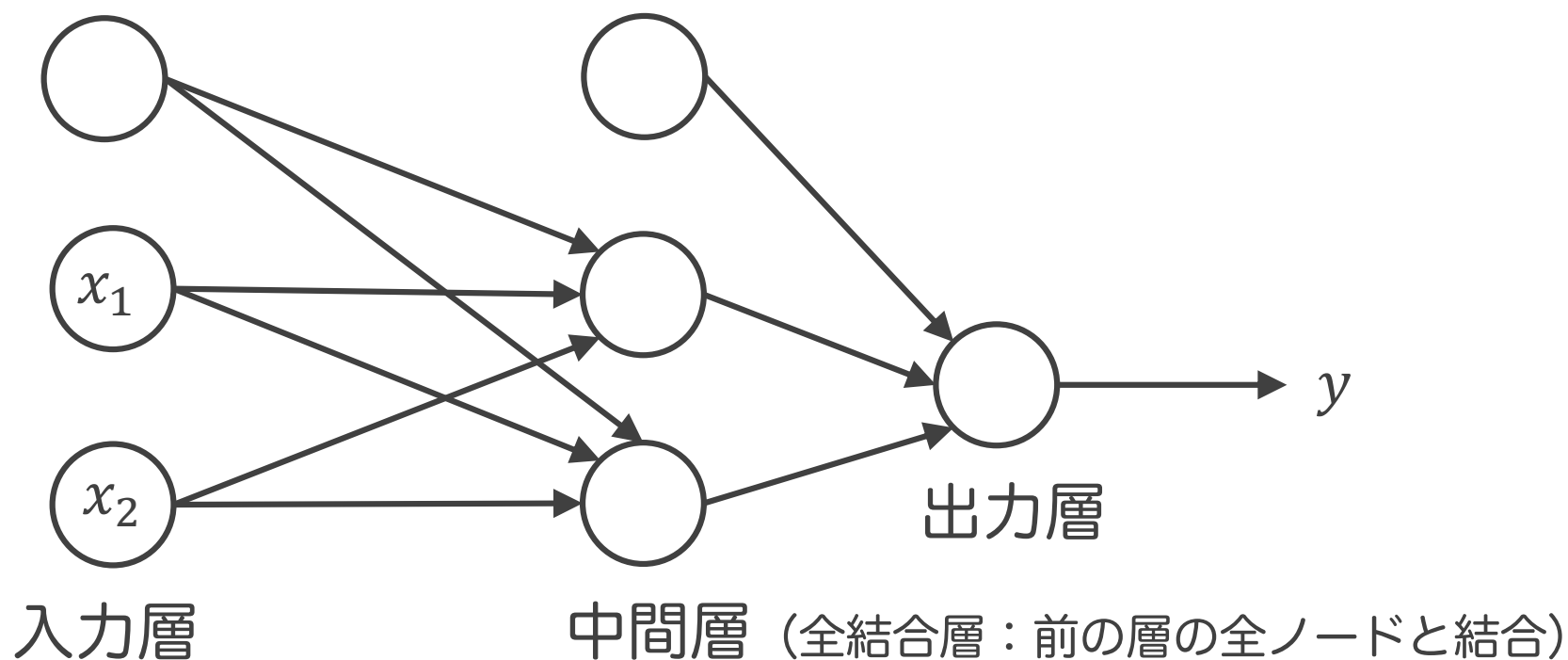
$$\frac{\partial \sum_{k=1}^N E_k}{\partial w} = \sum_{k=1}^N \frac{\partial E_k}{\partial w}$$

- 各レイヤのノードが複数ある場合
  - 多変数関数の偏微分公式を用いて偏微分を計算できる



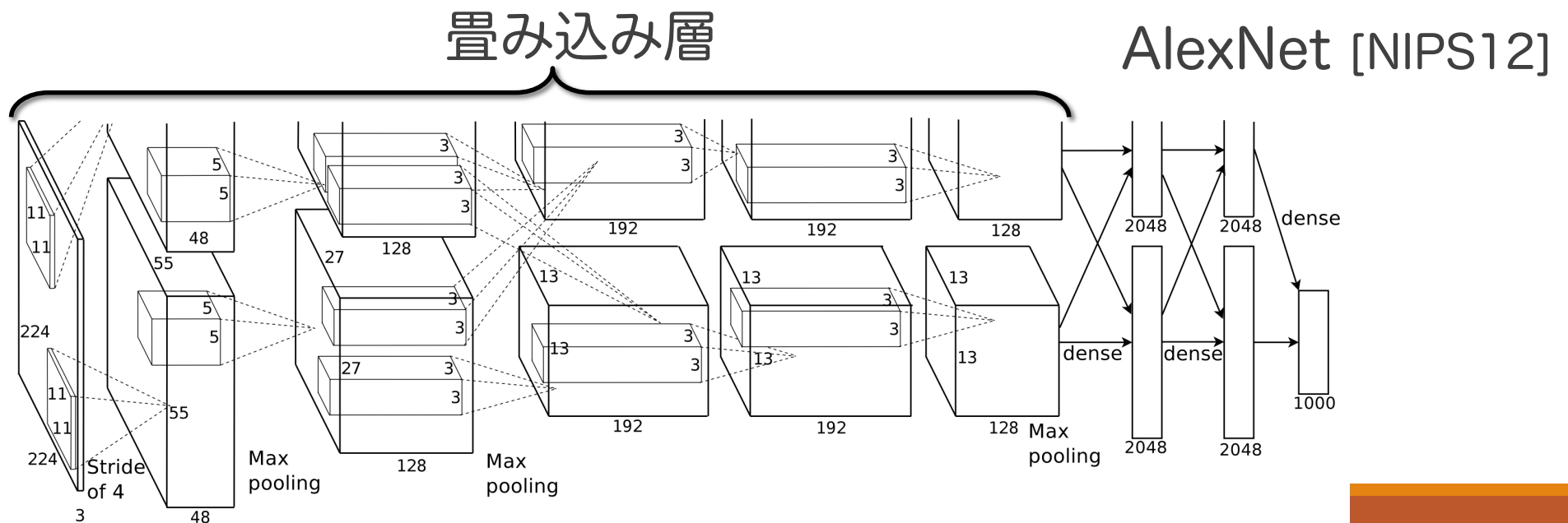
# 色々なネットワーク(1): 多層パーセプトロン (MLP)

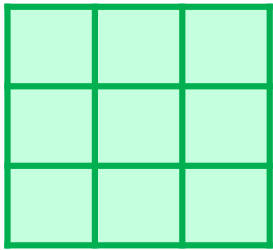
- 入力層, 出力層, (1つの) 中間層からなるニューラルネットワーク



# 色々なネットワーク(2)：畳み込みニューラルネットワーク (CNN)

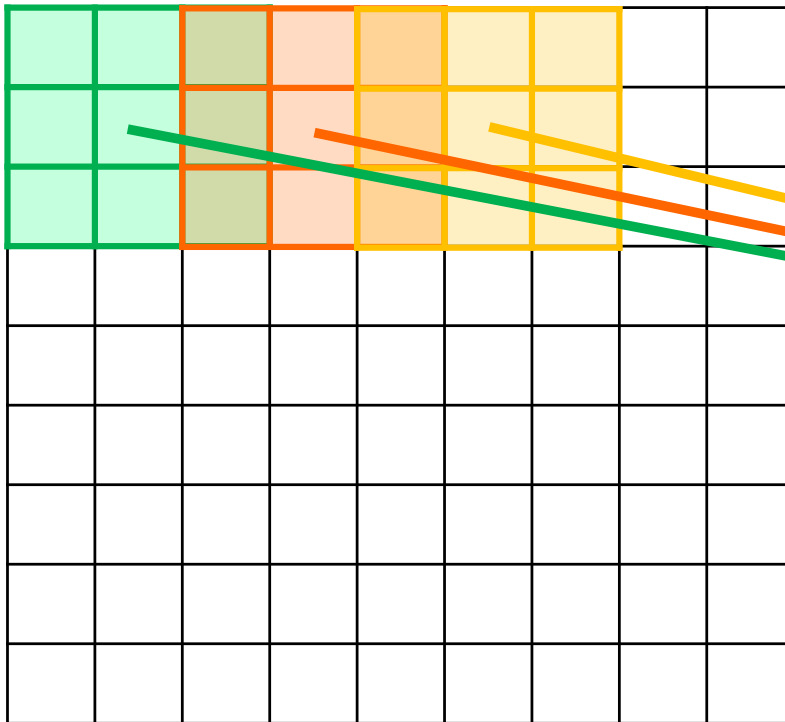
- 畳み込み層を備えたニューラルネットワーク
- 畳み込み層
  - 画像をスキャンして，画像のパターンを検出する





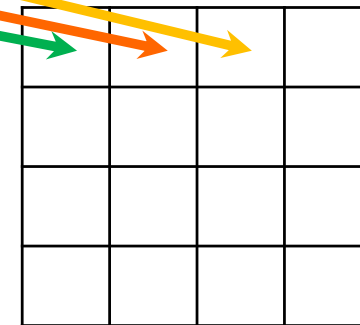
フィルタ（特定の画像パターンの検出用）

スキャン



画像

特徴マップ



様々な種類のフィルタ  
で特徴マップを作る

# フィルタ詳細

---

- フィルタにより得られる値

a	b
c	d

A	B
C	D



$$aA + bB + cC + dD$$

画像の画素値    フィルタ

この処理により画像のパターンを抽出できることを  
具体例で説明する

# 縦じまの検出

縦じま画像

1	0	1	0
1	0	1	0
1	0	1	0
1	0	1	0

1	-1	1	-1
1	-1	1	-1
1	-1	1	-1
1	-1	1	-1

→ 8

パターンが検出  
されると絶対値が  
大きな値が得られる

縦じま検出用  
フィルタ

横じま画像

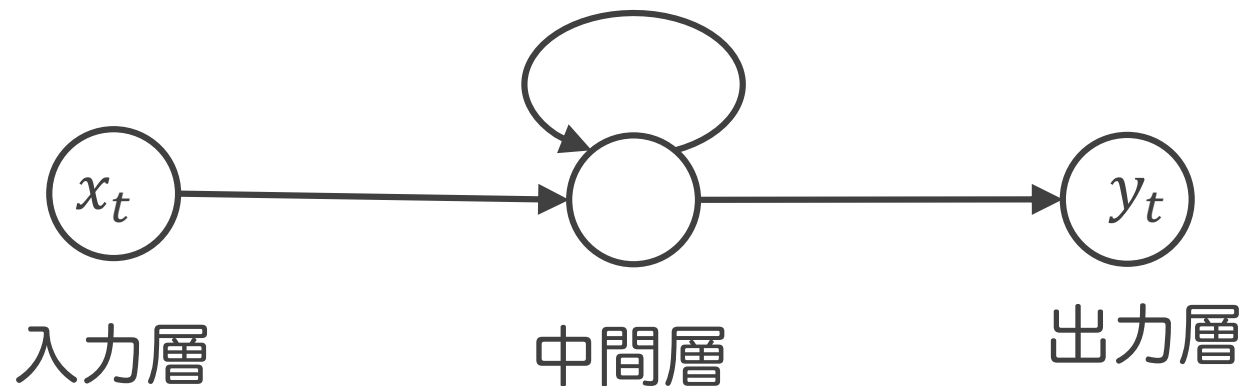
1	1	1	1
0	0	0	0
1	1	1	1
0	0	0	0

1	-1	1	-1
1	-1	1	-1
1	-1	1	-1
1	-1	1	-1

→ 0

# 色々なネットワーク(3)：再帰型ニューラルネットワーク (RNN)

- 時系列データを扱えるようにしたニューラルネットワーク
  - タイムステップ $t$ では $x_t$ が入力され,  $y_t$ が出力される
  - あるタイムステップでの中間層の出力が, 次のタイムステップに影響を与える



# 正則化

---


- 訓練データへの過度な適応（過学習）を防ぐ手法
- 教師あり学習の復習
  - 教師あり学習は入力と出力の関数を学習する
  - そのために関数の出力と実際の出力の間の誤差を最小化する
  - 誤差（損失関数）を $f_w(x)$ とすると、教師あり学習では $f_w(x)$ の最小化を試みる.
  - 関数のパラメータを集めたベクトルを $w$ とすると以下を高精度に計算することが目標

$$\min_w f_w(x)$$

# L1正則化, L2正則化

- 過学習防止のためのアイデア

- $w = (w_0, w_1, \dots, w_n)$ とした際,  $w_0, w_1, \dots, w_n$ 中に極端に値が大きいものがあると過学習が疑われる
- ということで $w_0, w_1, \dots, w_n$ が大きい値にならないようにした上で誤差を最小化したい

$\min_w f_w(x)$    $\min_w \left( f_w(x) + \lambda \sum_{k=1}^n w_k^2 \right)$  L2正則化

項追加

(パラメータの値が大きくならないようにする)

$\min_w \left( f_w(x) + \lambda \sum_{k=1}^n |w_k| \right)$  L1正則化

正則化項

$\lambda$  は適当な値



# L1正則化, L2正則化

---

- L2正則化とL1正則化の違い
  - L2正則化の方が計算が楽
  - L1正則化の結果の方が好まれる傾向がある
    - L1正則化では $w_0, w_1, \dots, w_n$ の中に0が出来やすい
    - (経験則として) より少ないパラメータで関数を構成した方が現実と適合することが多い
      - ⇒ 参考: オッカムのカミソリ, スパースモデリング

# ドロップアウト

---

- ニューラルネットワークにおける正則化手法
- 学習時ランダムにノードを無効化する
- 例えばある層のドロップアウト率が0.5なら
  - 層の出力ベクトルのうち、半分の要素は0
  - 残りの要素は倍の値
- なんでこんなことするの？
  - 開発者のHintonいわく、銀行の窓口の対応者をいつも変えることで不正が起きにくくなるのと一緒に

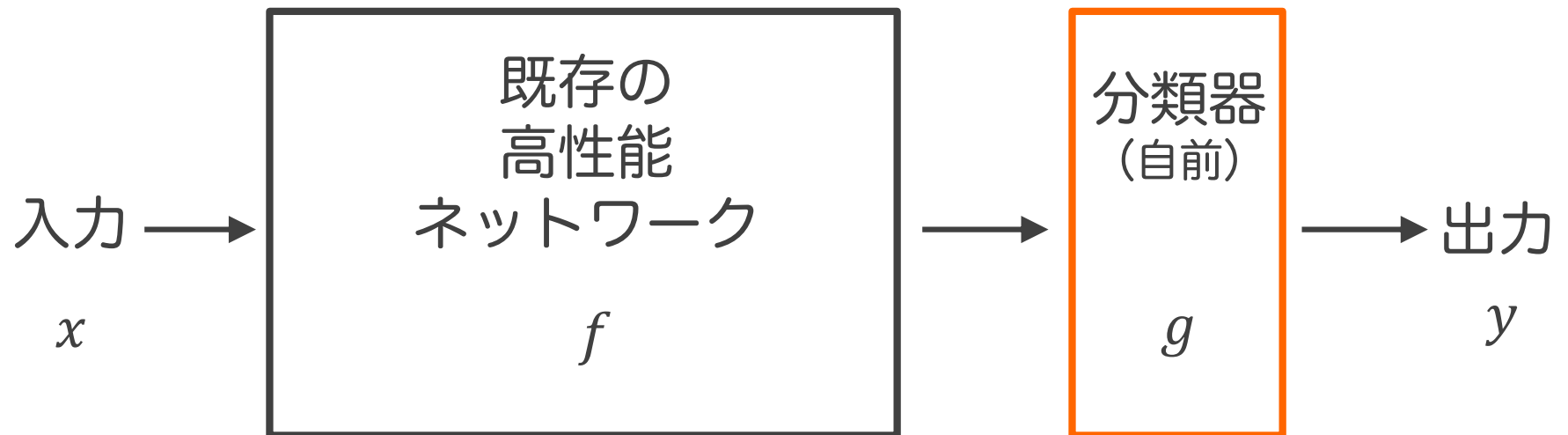
# 転移学習

---

- 学習済みのニューラルネットワークを活用して自分用のネットワークを作る
- 主に画像分類で使われる
- 概要
  - 画像分類では前半の層で画像のパターンを認識し、最終層付近で出力を計算する
  - 手持ちデータのラベルに応じて出力生成部分だけを取り替えるということがよく行われる

# 転移学習

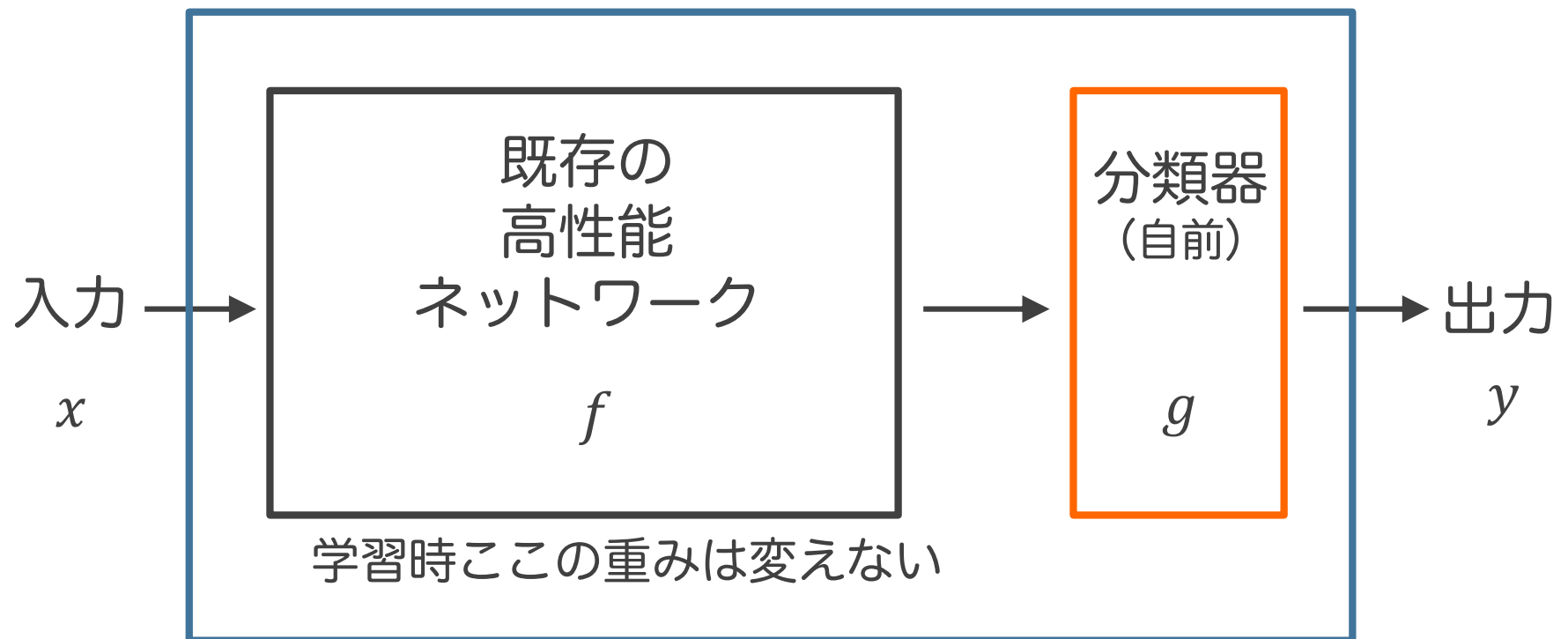
---



$$y = g \circ f(x)$$

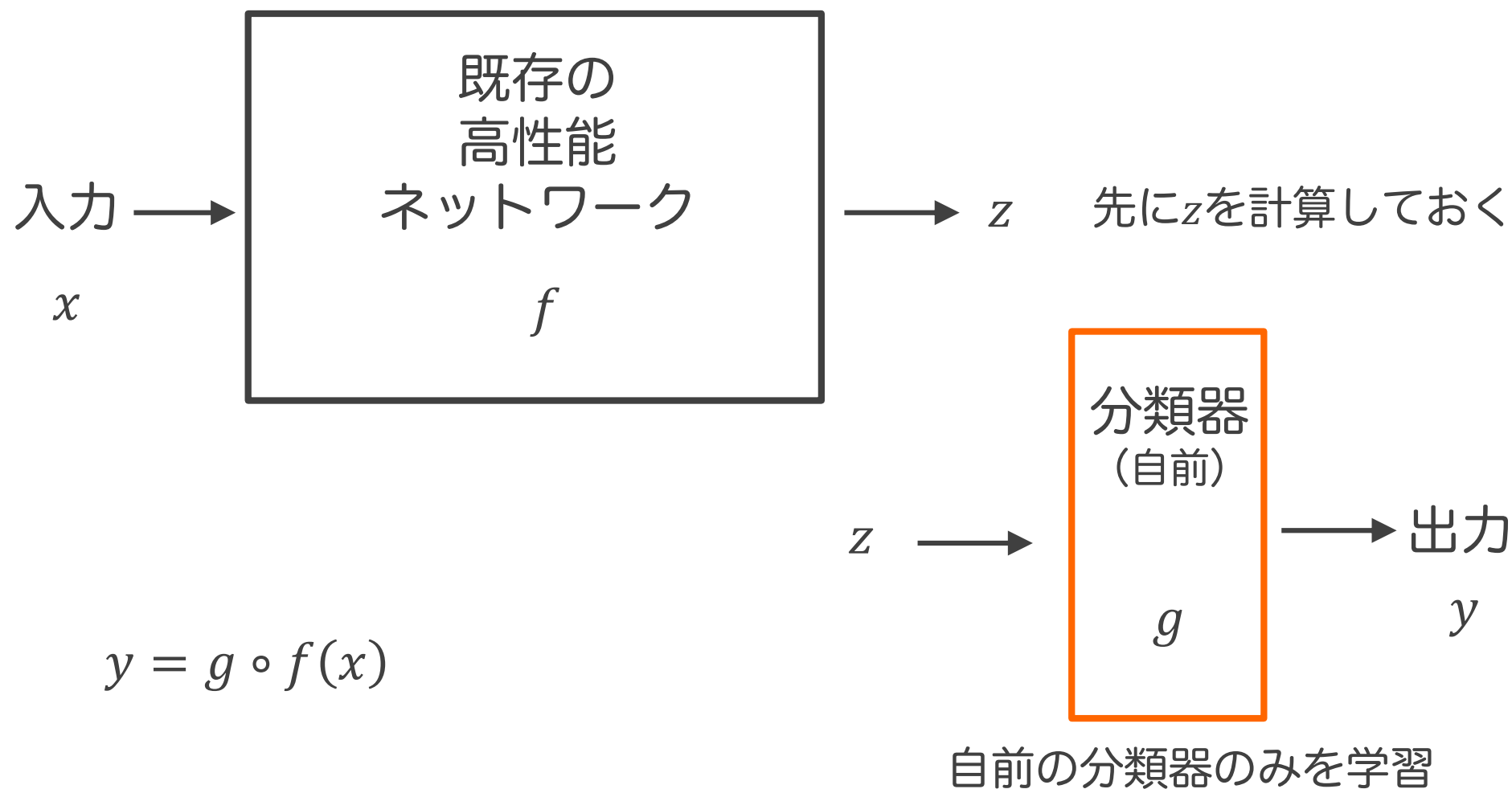
# 転移学習のやり方1

このようなネットワークを作る



$$y = g \circ f(x)$$

# 転移学習のやり方2 (こちらの方が高速)



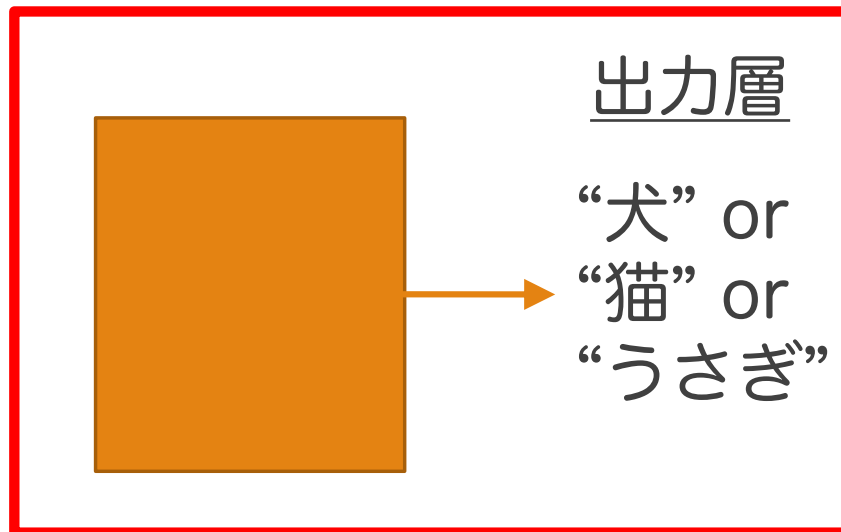
# ミニバッチ学習

---

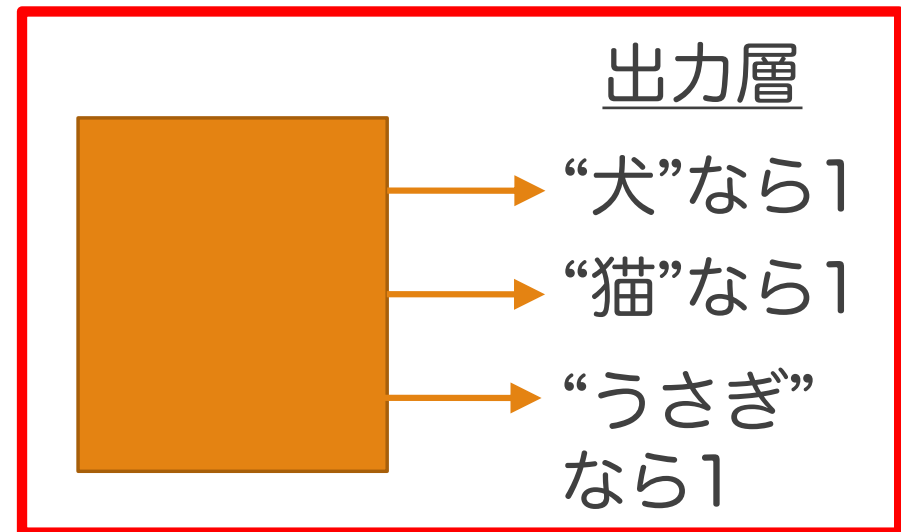
- 機械学習モデルの学習をする際、誤差を求めるには、すべてのサンプルに対する誤差を計算する必要がある
- これは時間がかかるのでやめる  
⇒ ミニバッチ学習
  - 一部のサンプルのみから求めた誤差を用いてパラメータを更新する
  - その際のサンプルのサイズをバッチサイズと呼ぶ

# One-hot表現

- 分類時, クラス名を出力するのではなく, 各クラスについてYes/No (0~1) を出力する



こうでなく



こうする