

6. 識別 - ニューラルネットワーク -

6.1 識別関数法

- 識別関数法とは

- 確率の枠組みにはとらわれず、

$$f_{Positive}(\boldsymbol{x}) > f_{Negative}(\boldsymbol{x})$$

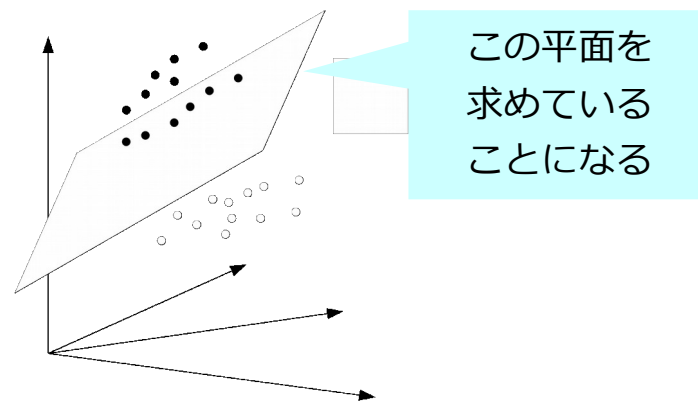
ならば \boldsymbol{x} を Positive と判定する関数 f を推定する

- 単層パーセプトロン

最も単純な識別関数法の実現

- 識別関数として 1 次式 (= 直線・平面) を仮定

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$$

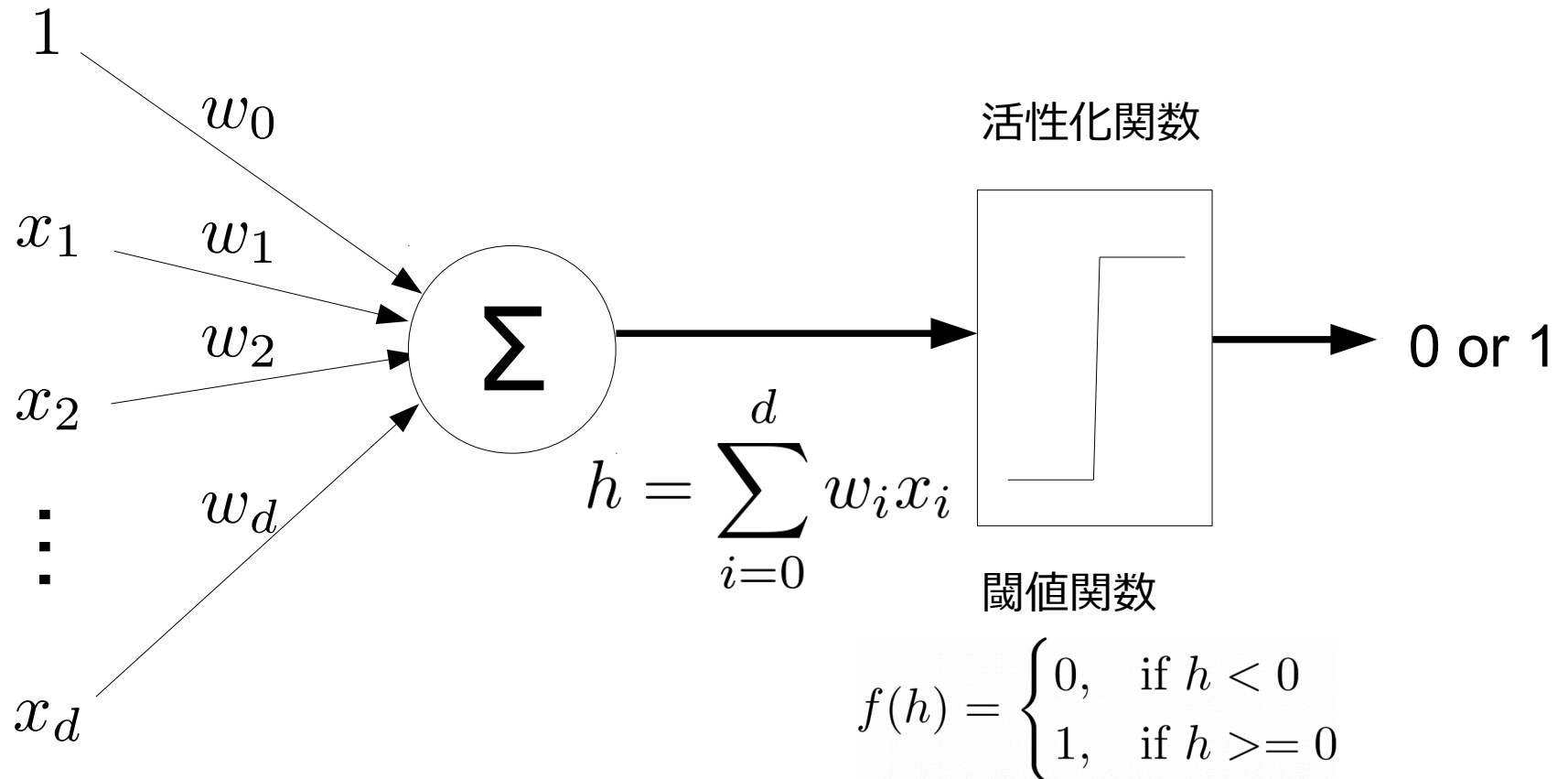


6.2 誤り訂正学習

- 単層パーセプトロンの定義

以後、 w は w_0 を含む

- $w^T x = 0$ という特徴空間上の超平面を表現



6.2 誤り訂正学習

- パーセプトロンの学習規則

データが線形分離可能な場合はパーセプトロンの学習規則で学習可能

1. w の初期値を適当に決める
2. 学習パターンからひとつ x を選び、 $g(x)$ を計算
3. 誤識別が起きたときのみ、 w を修正する

$$w' = w + \rho x \quad (\text{クラス 1 のパターンをクラス 2 と誤ったとき})$$

$$w' = w - \rho x \quad (\text{クラス 2 のパターンをクラス 1 と誤ったとき})$$

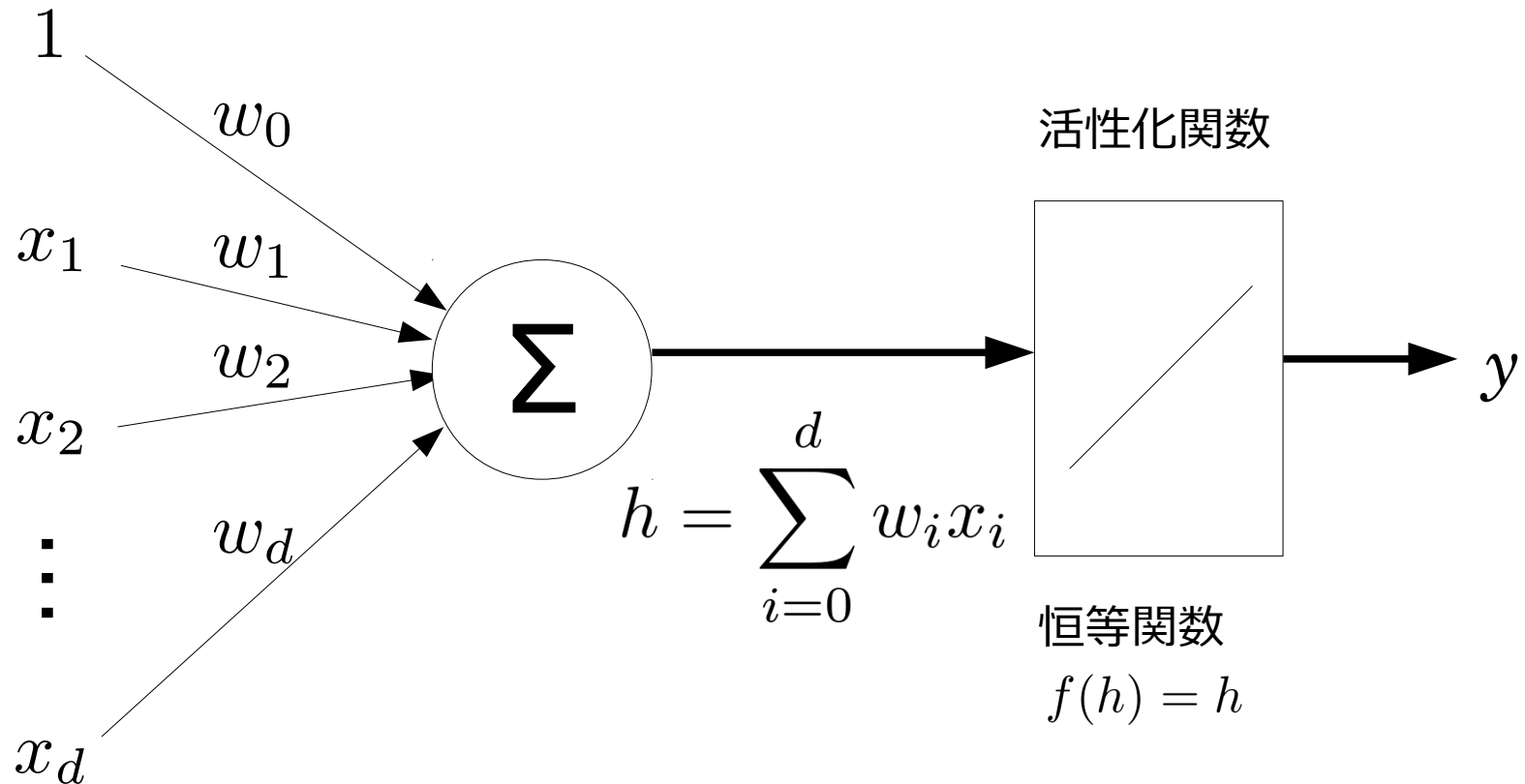
学習係数

4. 2,3 を全ての学習パターンについて繰り返す
5. すべて識別できたら終了。そうでなければ 2 へ

6.3 最小二乗法による学習

- 最小二乗法に用いる計算ユニット

- $w^T x$ の値を計算



6.3 最小二乗法による学習

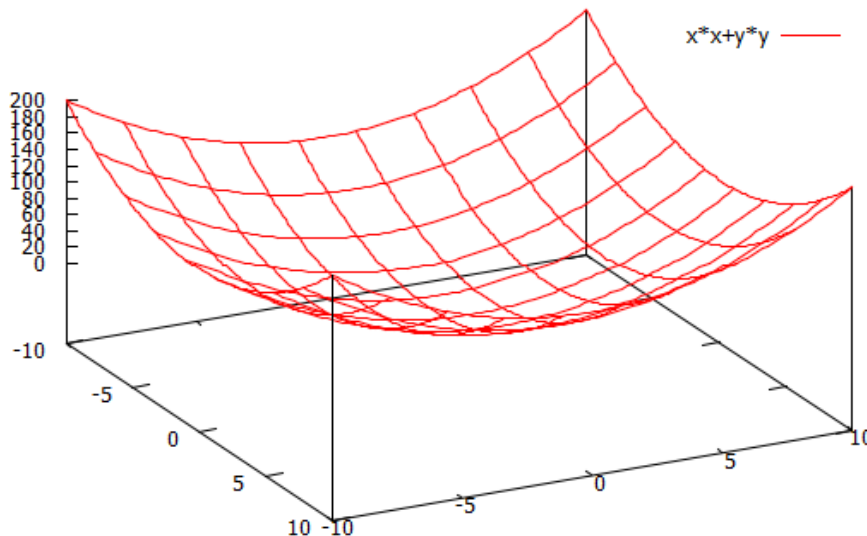
- エラーの定義

- 二乗誤差 $J(\boldsymbol{w}) \equiv \frac{1}{2} \sum_{\boldsymbol{x}_i \in D} (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2$

全データに対する
正解と関数の出力
との差の2乗和

- J は \boldsymbol{w} の関数

- \boldsymbol{w} を J の勾配方向へ一定量だけ動かすことを繰り返して、最適解へ収束させる (→最急降下法)

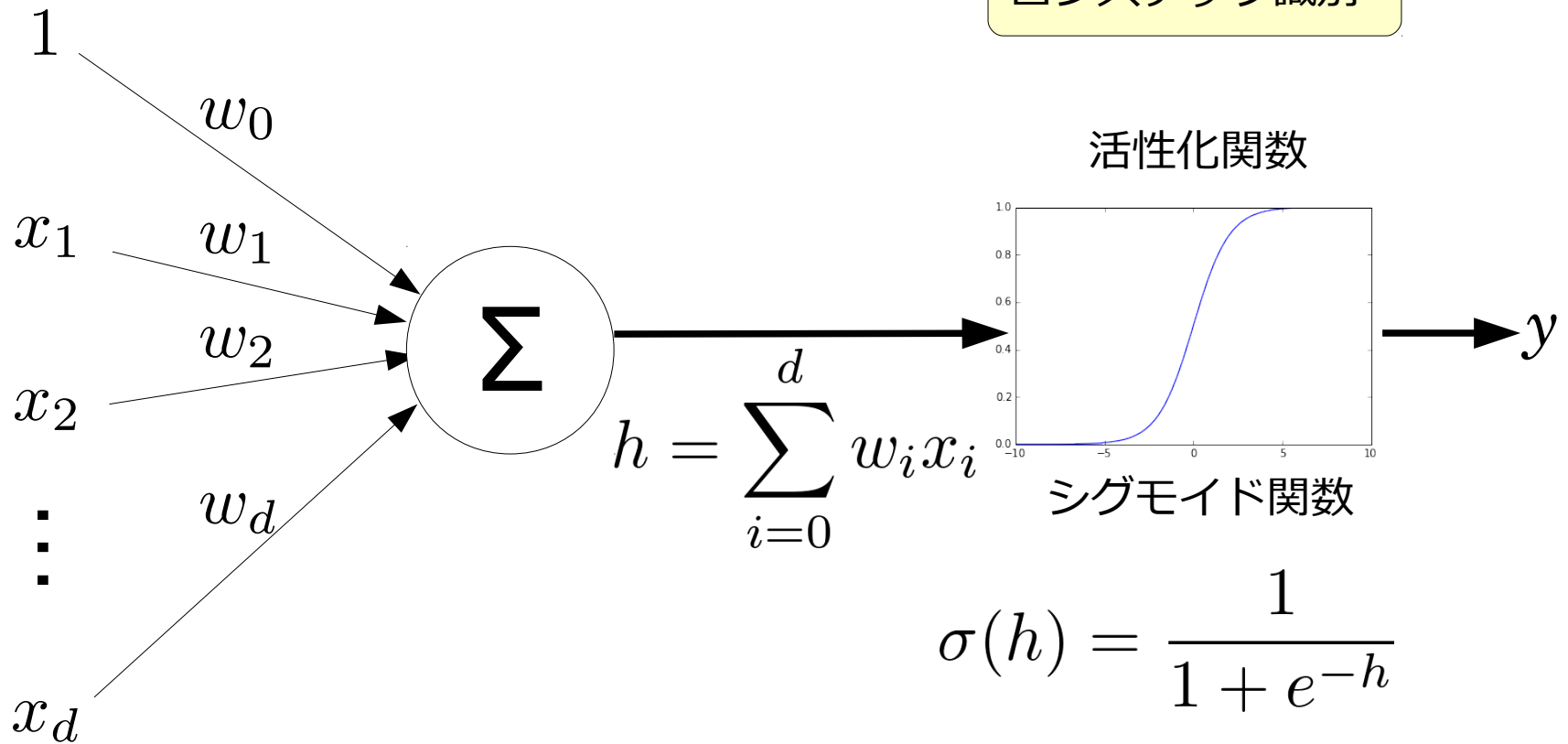


$$\begin{aligned} \boldsymbol{w}' &= \boldsymbol{w} - \rho \frac{\partial J}{\partial \boldsymbol{w}} \\ &= \boldsymbol{w} - \rho \sum_{p=1}^n (\boldsymbol{w}^T \boldsymbol{x}_p - b_p) \boldsymbol{x}_p \end{aligned}$$

6.3 最小二乗法による学習

- シグモイド関数の適用
 - 多層の誤差修正に対応するために、勾配計算の際に微分可能な活性化関数を用いる

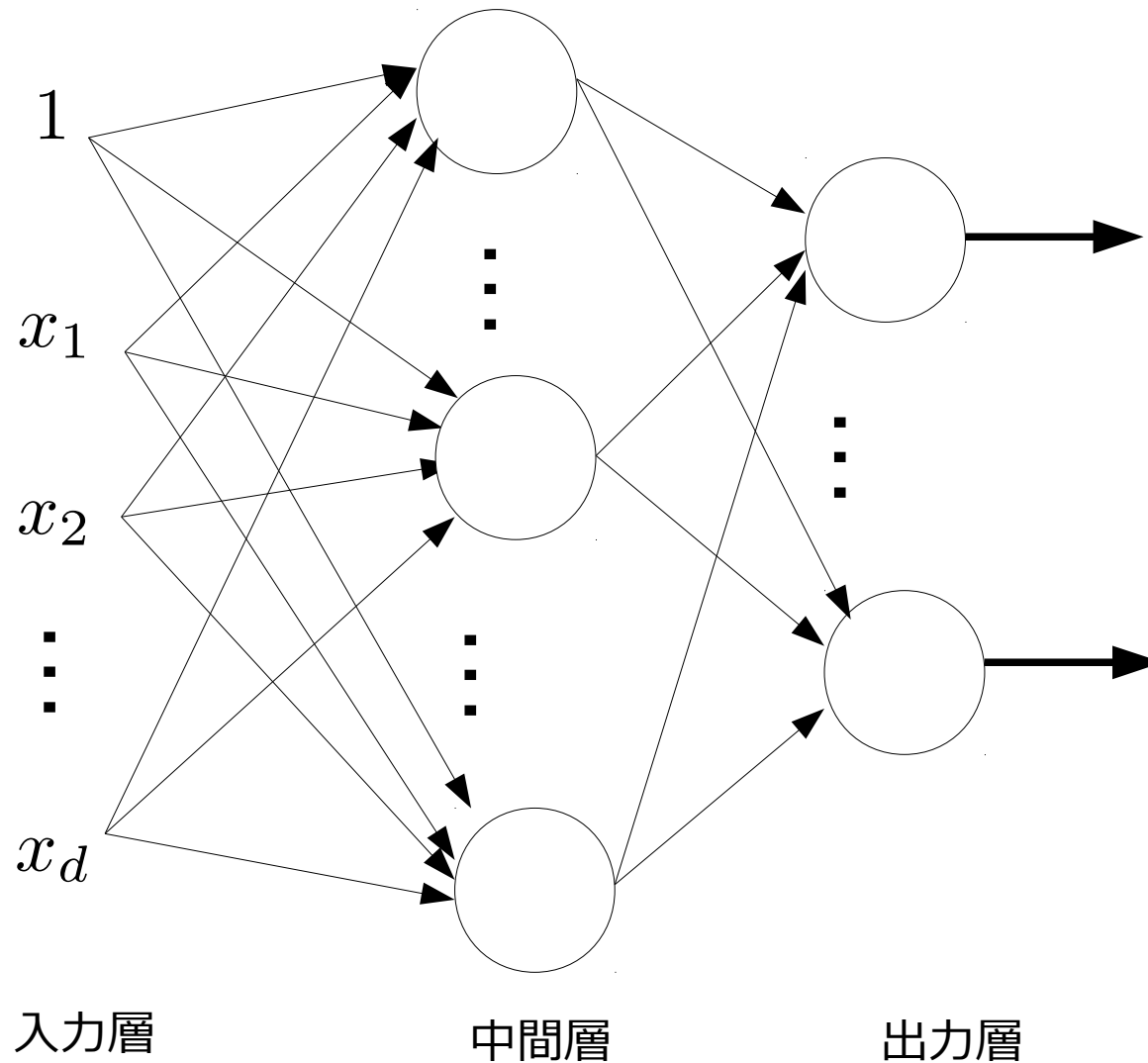
ロジスティック識別



$$\sigma'(h) = \sigma(h) \cdot (1 - \sigma(h))$$

6.4 ニューラルネットワーク

- 3層のフィードフォワードネットワーク



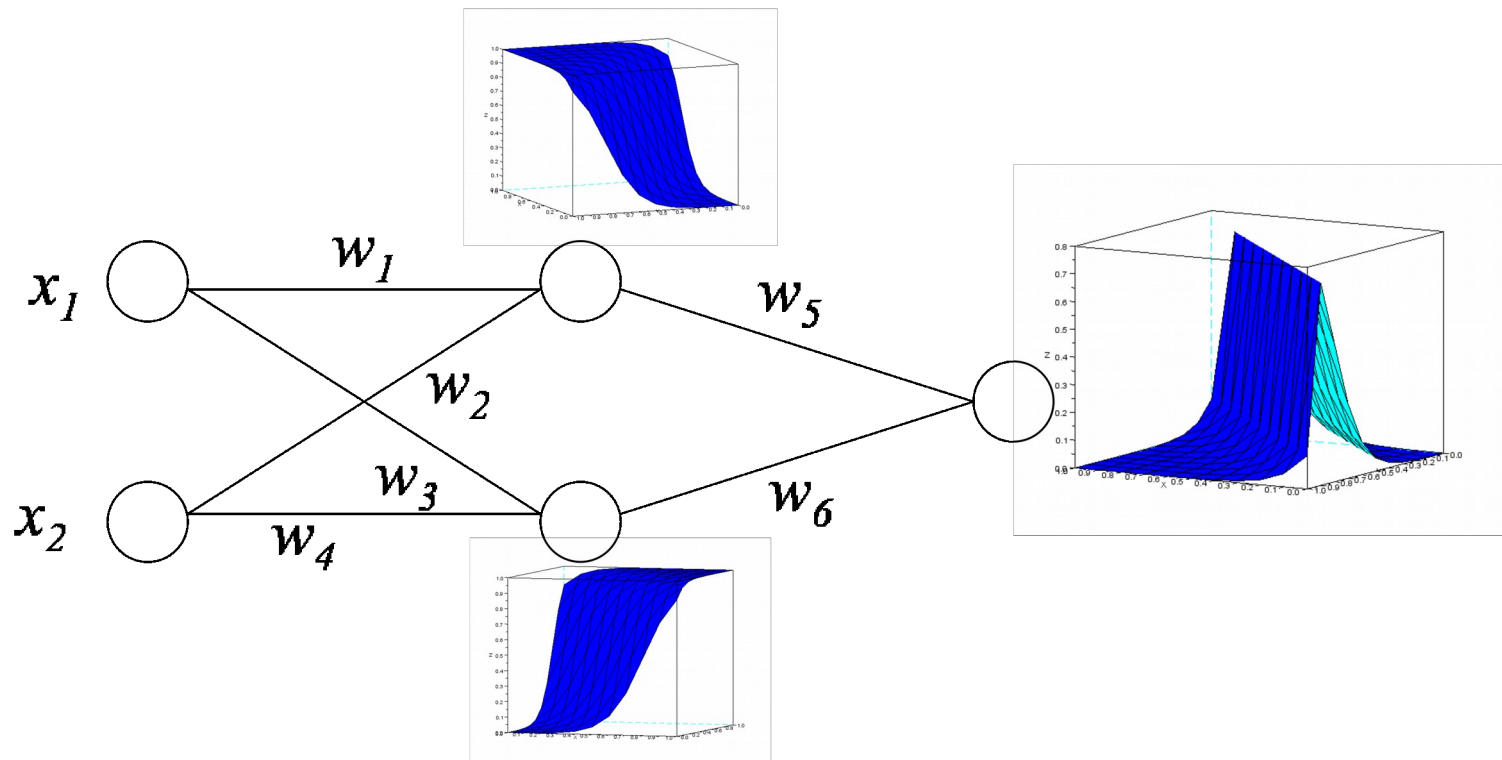
6.4 ニューラルネットワーク

- フィードフォワードネットワークのユニット
 - 中間層の活性化関数：シグモイド関数
 - 出力層の活性化関数：シグモイド関数または softmax 関数

$$f(h_i) = \frac{\exp(h_i)}{\sum_{j=1}^c \exp(h_j)}$$

6.4 ニューラルネットワーク

- 識別面の複雑さ
 - 中間層ユニットの個数に関する
 - シグモイド関数（非線形）を任意の重み・方向で足し合わせることで複雑な非線形識別面を構成



6.4 ニューラルネットワーク

- 誤差逆伝播法

1. リンクの重みを小さな初期値に設定

2. 個々の学習データ (x_i, y_i) に対して以下繰り返し

- 入力 x_i に対するネットワークの出力 o_i を計算

- a) 出力層の k 番目のユニットに対してエラー量 δ 計算

$$\delta_k \leftarrow o_k(1 - o_k)(y_k - o_k)$$

- b) 中間層の h 番目のユニットに対してエラー量 δ 計算

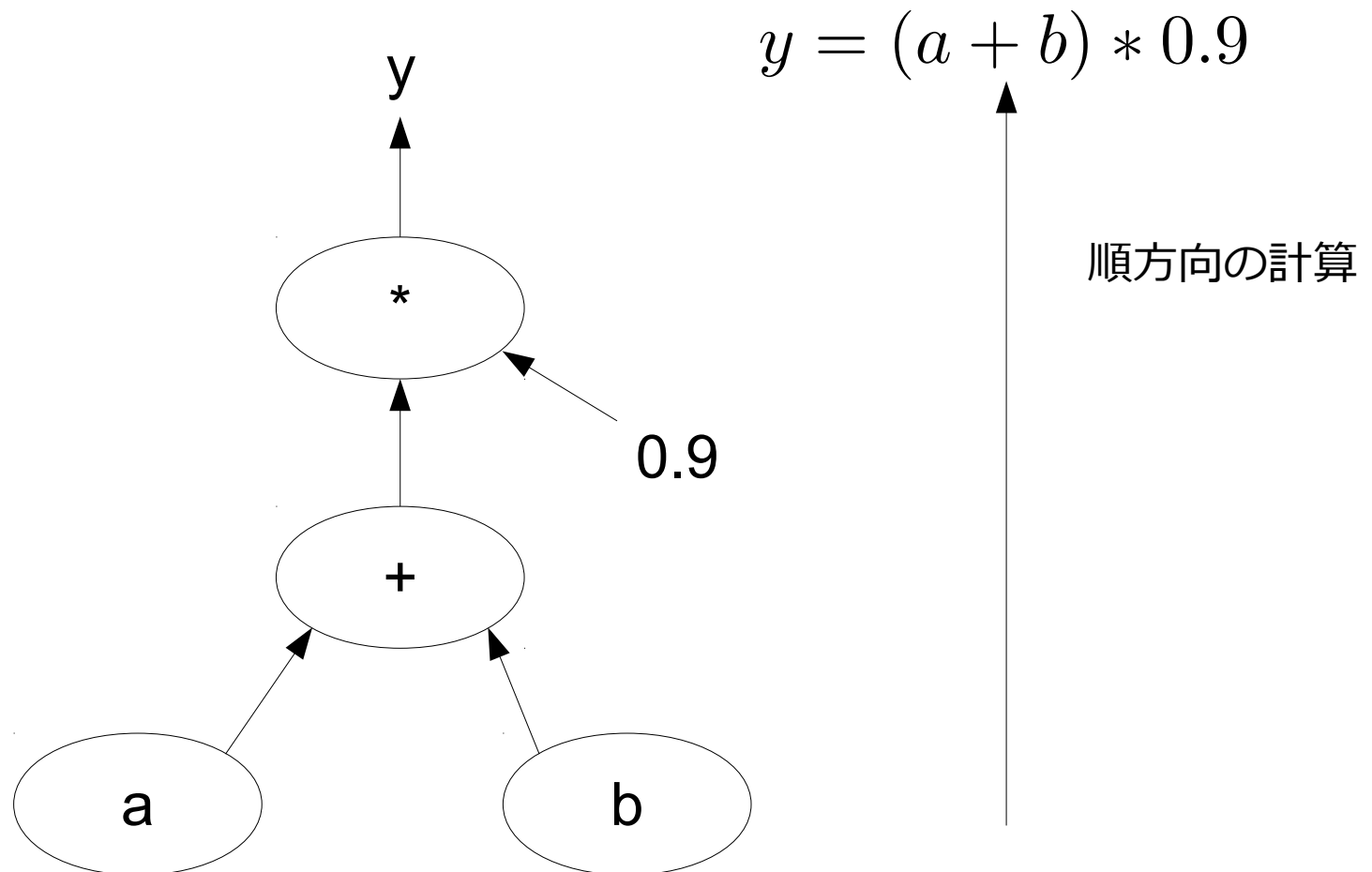
$$\delta_k \leftarrow o_k(1 - o_k) \sum_{k \in \text{outputs}} w_{kh} \delta_k$$

- c) 重みの更新

$$w'_{ji} \leftarrow w_{ji} + \eta \delta_j x_{ji}$$

6.4 ニューラルネットワーク

- 計算グラフによる表現
 - 参考サイト) <http://postd.cc/2015-08-backprop/>

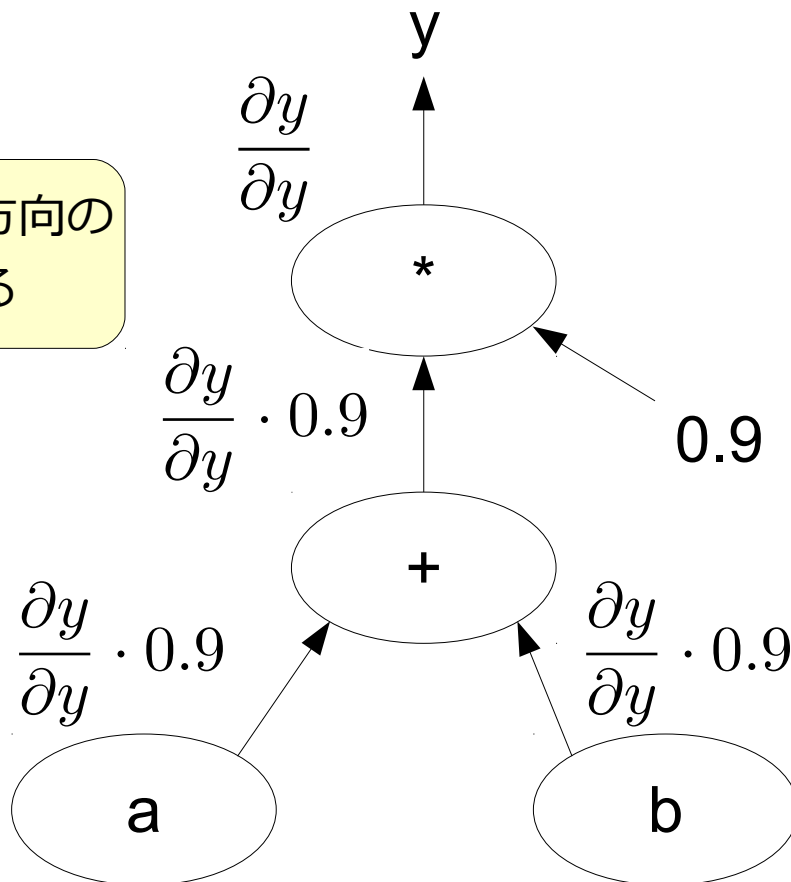


6.4 ニューラルネットワーク

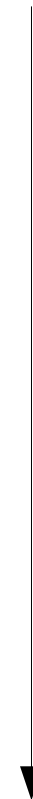
- 逆方向の微分

掛け算ノードは順方向の値を交換して掛ける

足し算ノードはそのまま流す

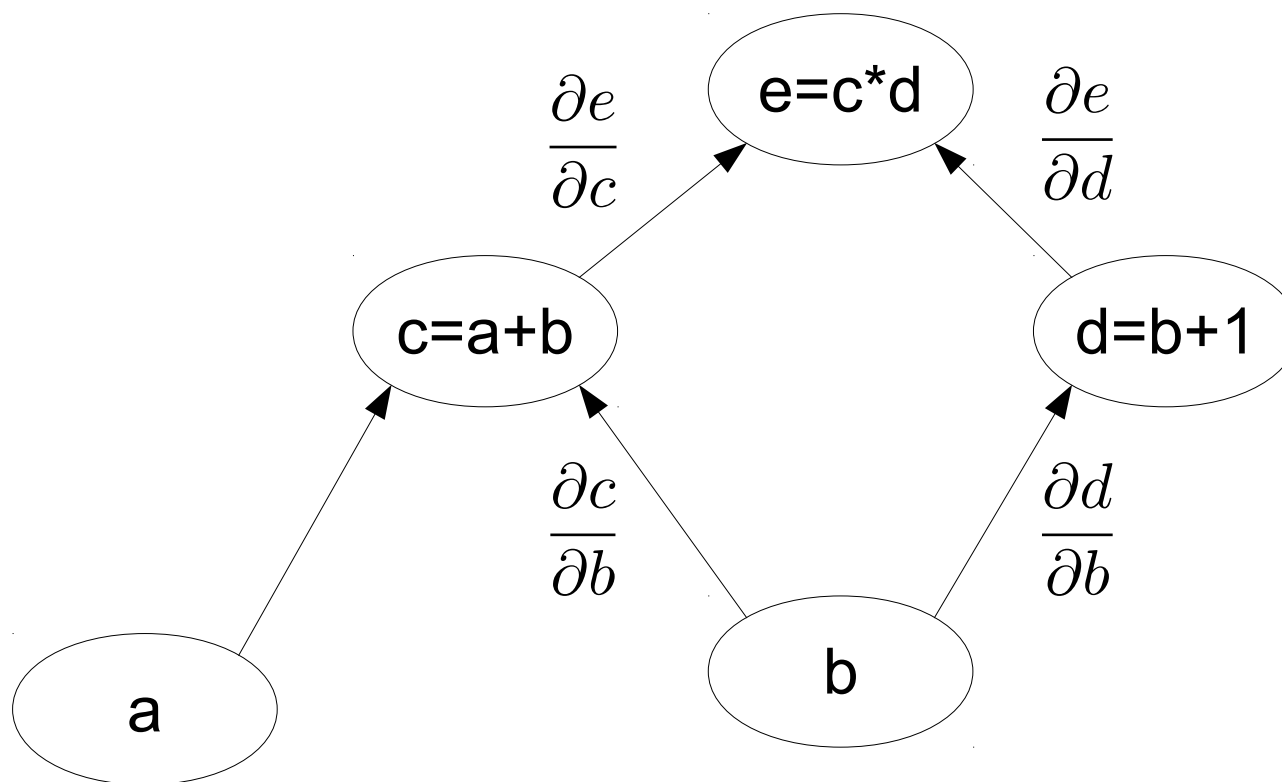


逆方向の微分



6.4 ニューラルネットワーク

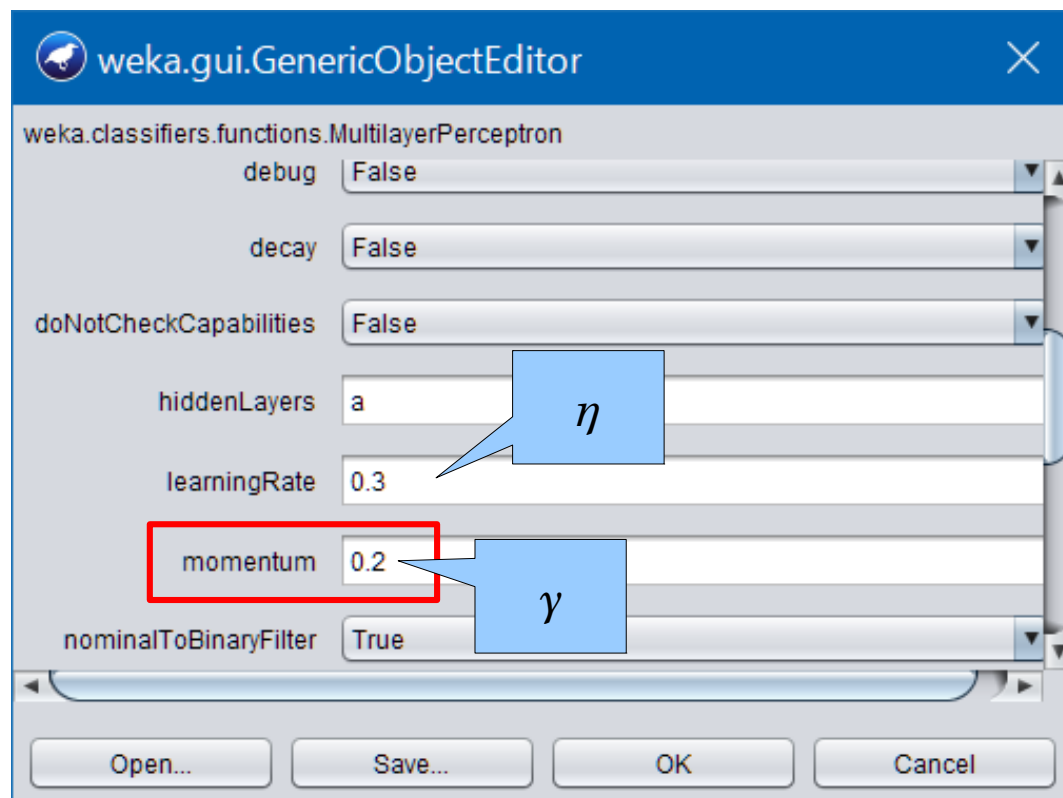
- 逆方向の微分



$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b}$$

6.4 ニューラルネットワーク

- Weka でのパラメータ調整
 - モーメントム（慣性） v
 - 更新の方向に勢いを付けることで収束を早め、振動を抑制する



$$v_t = \gamma v_{t-1} + \eta \frac{\partial E}{\partial w}$$

$$w' = w - v_t$$

6.4 ニューラルネットワーク

- sklearn の学習パラメータ (1/2)

```
MLPClassifier(  
    activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,  
    beta_2=0.999, early_stopping=False, epsilon=1e-08,  
    hidden_layer_sizes=(100,), learning_rate='constant',  
    learning_rate_init=0.001, max_iter=200, momentum=0.9,  
    nesterovs_momentum=True, power_t=0.5, random_state=None,  
    shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,  
    verbose=False, warm_start=False)
```

- activation: 活性化関数
 - 'identity': 同一値関数 $f(x) = x$
 - 'logistic': シグモイド関数 $f(x) = 1 / (1 + \exp(-x))$
 - 'tanh': 双曲線正接 $f(x) = \tanh(x)$
 - 'relu': ランプ関数 $f(x) = \max(0, x)$

6.4 ニューラルネットワーク

- sklearn の学習パラメータ (2/2)
- solver: 最適化手法
 - ‘lbfgs’: 準ニュートン法
 - 2 次微分を更新式に加える
 - ‘sgd’: 確率的最急降下法
 - ‘adam’: Adaptive Moment Estimation
 - モーメントの改良: 直前の値だけではなく、これまでの指数平滑移動平均を用いる
 - モーメントの拡張: 分散に関するモーメントも用いる
 - まれに観測される特徴軸に対して大きく更新する効果

データ数が多いときは adam 、
少ないときは lbfgs が
勧められている