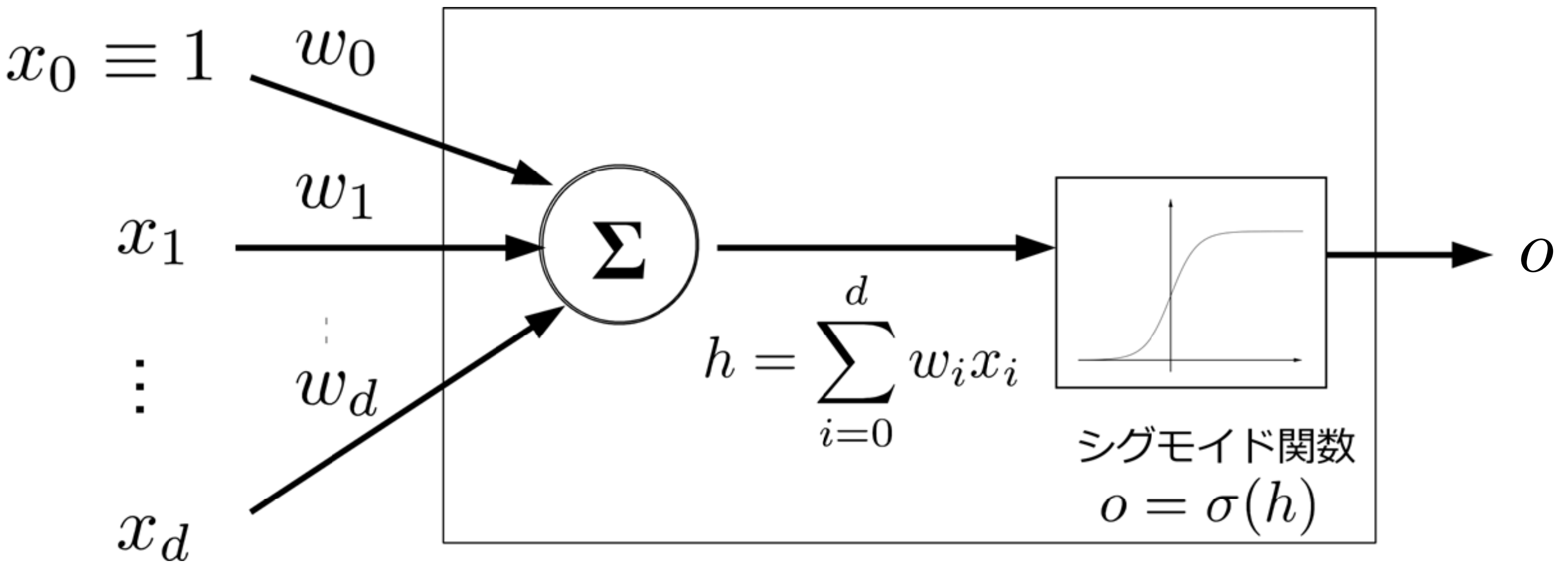
8. ニューラルネットワーク

8.1 ニューラルネットワークの計算ユニット

- ロジスティック識別と同じもの

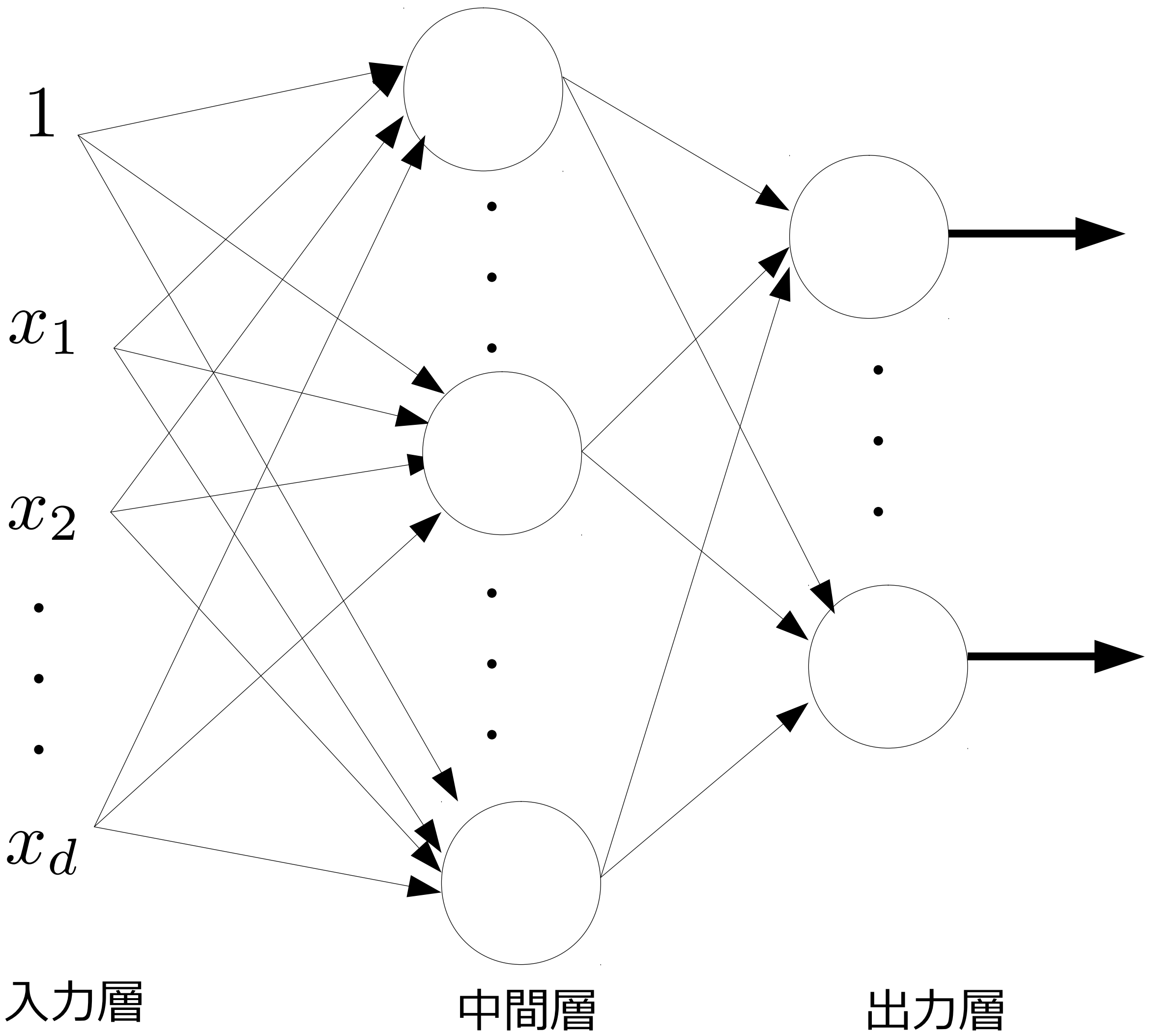


$$\sigma(h) = \frac{1}{1 + e^{-h}}$$

8.2 フィードフォワード型ニューラルネットワーク

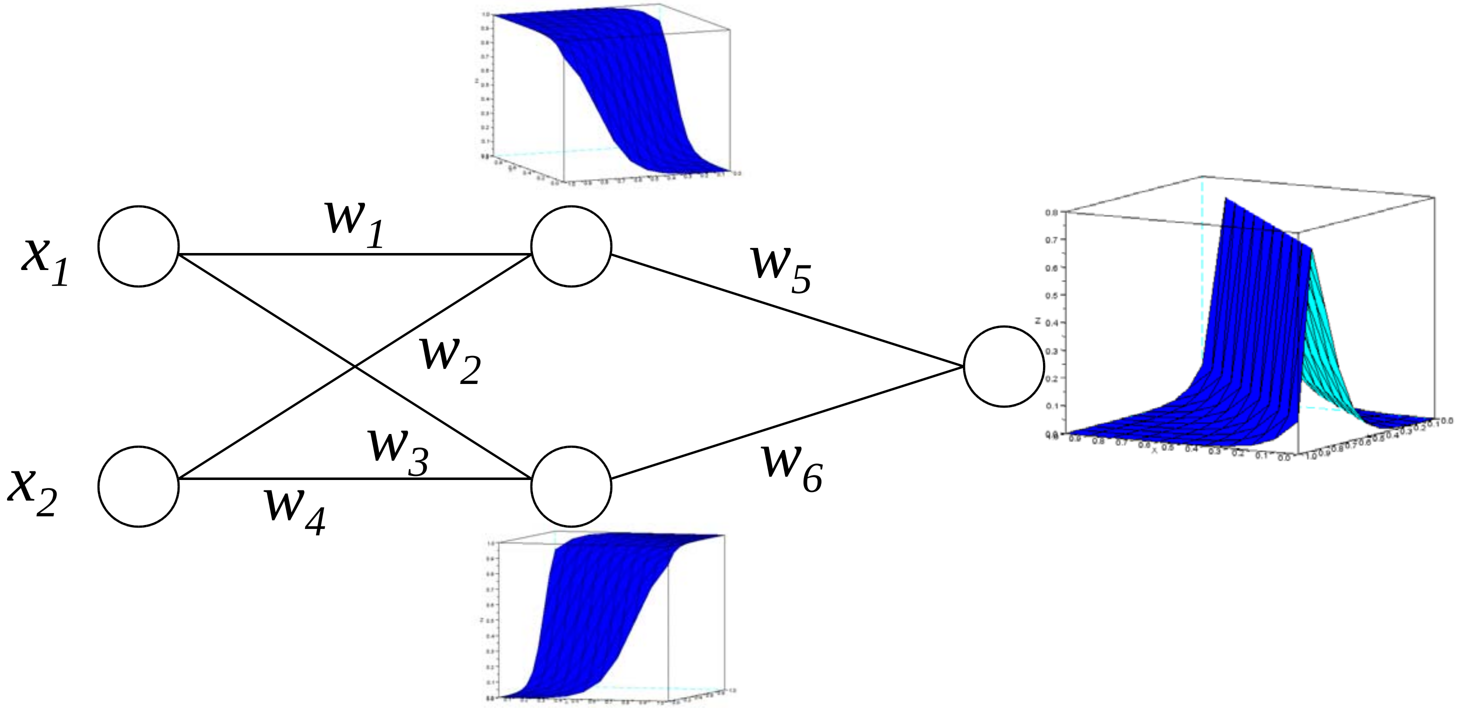
8.2.1 ニューラルネットワークの構成

- フィードフォワード型



8.2.1 ニューラルネットワークの構成

- 識別面の複雑さ
  - 中間層ユニットの個数に関係する
  - 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



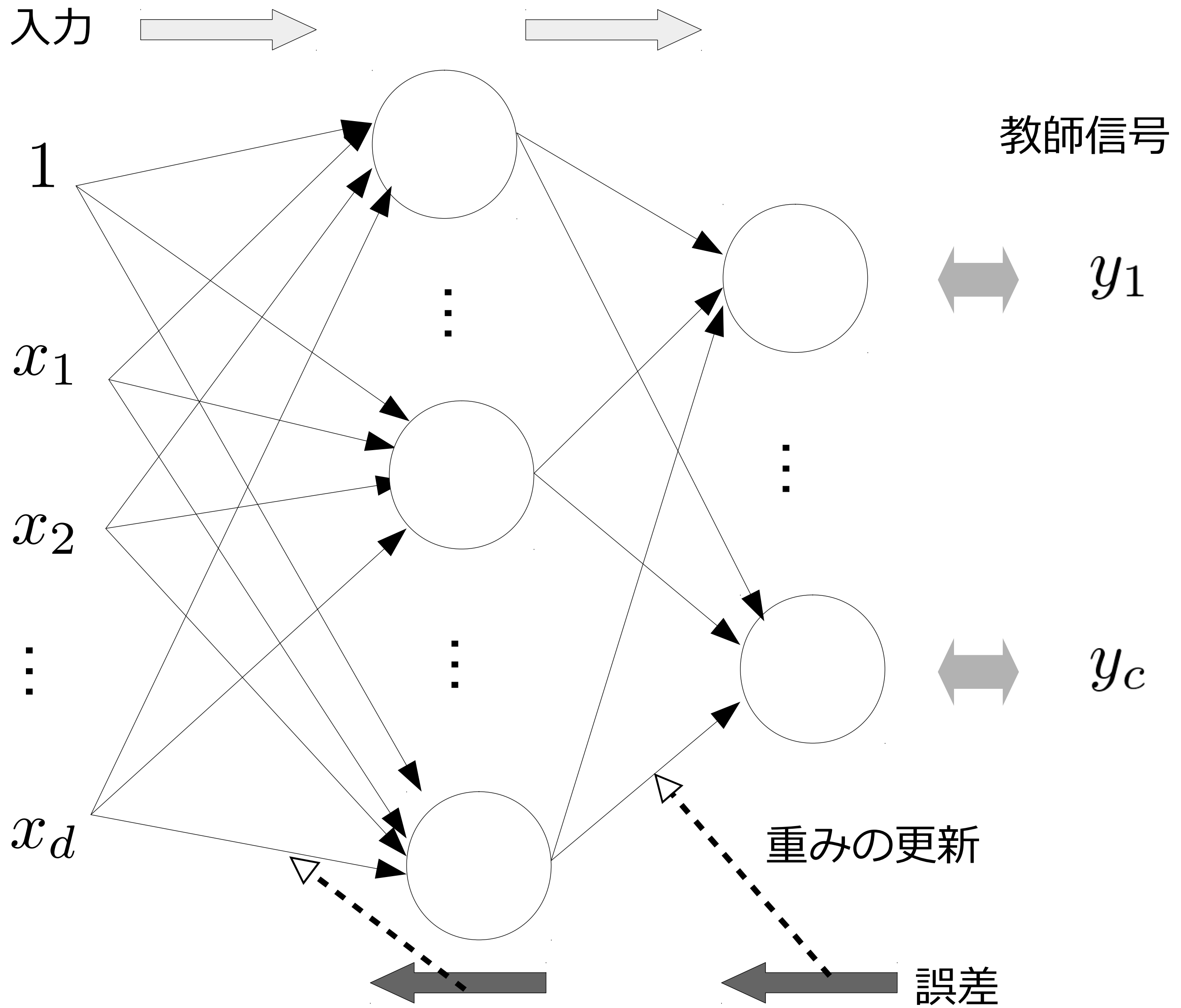
8.2.1 ニューラルネットワークの構成

- 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

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

8.2.2 誤差逆伝播法による学習

- 誤差逆伝播法の名前の由来



8.2.2 誤差逆伝播法による学習

- 誤差関数

$$E(\boldsymbol{w}) \equiv \frac{1}{2} \sum_{\boldsymbol{x}_i \in D} (y_i - o_i)^2$$

- 重みの調整式

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

ユニット j の重み
が変化すれば、
誤差も変化する

8.2.2 誤差逆伝播法による学習

- 誤差の変化量の計算

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{\boldsymbol{x}_j \in D}(y_j - o_j)^2 \\
&= \frac{1}{2}\sum_{\boldsymbol{x}_j \in D}\frac{\partial}{\partial w_i}(y_j - o_j)^2 \\
&= \sum_{\boldsymbol{x}_j \in D}(y_j - o_j)\frac{\partial}{\partial w_i}(y_j - o_j)
\end{aligned}
$$

ID: 0808

8.2.2 誤差逆伝播法による学習

- 出力値の微分の計算

$$\frac{\partial o_j}{\partial w_i} = \frac{\partial o_j}{\partial h_j}\frac{\partial h_j}{\partial w_i}$$

$$\frac{\partial o_j}{\partial h_j} = o_j(1 - o_j)$$

$$\frac{\partial h_j}{\partial w_i} = x_{ij}$$

- （出力層の）重みの更新式

$$w_i \leftarrow w_i + \eta \sum_{\boldsymbol{x}_j \in D} (y_j - o_j)o_j(1 - o_j)x_{ij}$$

8.2.2 誤差逆伝播法による学習

- 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 $(\boldsymbol{x}_i, y_i)$に対して以下繰り返し
   - 入力 $\boldsymbol{x}_i$ に対するネットワークの出力 $\boldsymbol{o}_i$ 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

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

   c)重みの更新

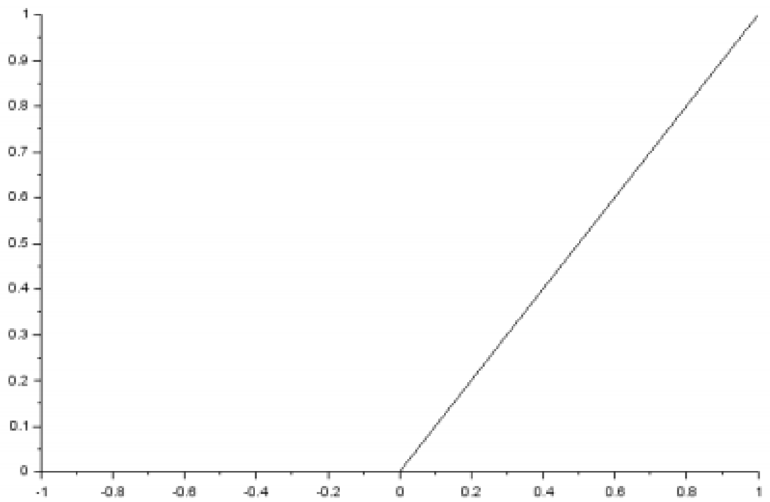
$$w'_{ji} \leftarrow w_{ji} + \eta\delta_j x_{ji}$$

8.3 ニューラルネットワークの深層化

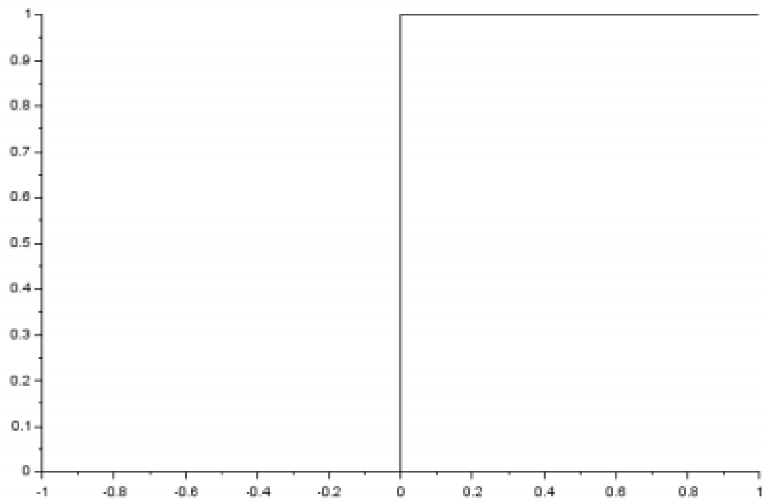
- 勾配消失問題
  - 多階層ネットワークの学習の難しさ
    - 誤差逆伝播法による多層ネットワークの学習は，重みの修正量が層を戻るにつれて小さくなってゆく

8.3 ニューラルネットワークの深層化

- さまざまな活性化関数
  - 微分によって大きく値が減らない関数
    - ReLu　　$f(x) = \max(0, x)$



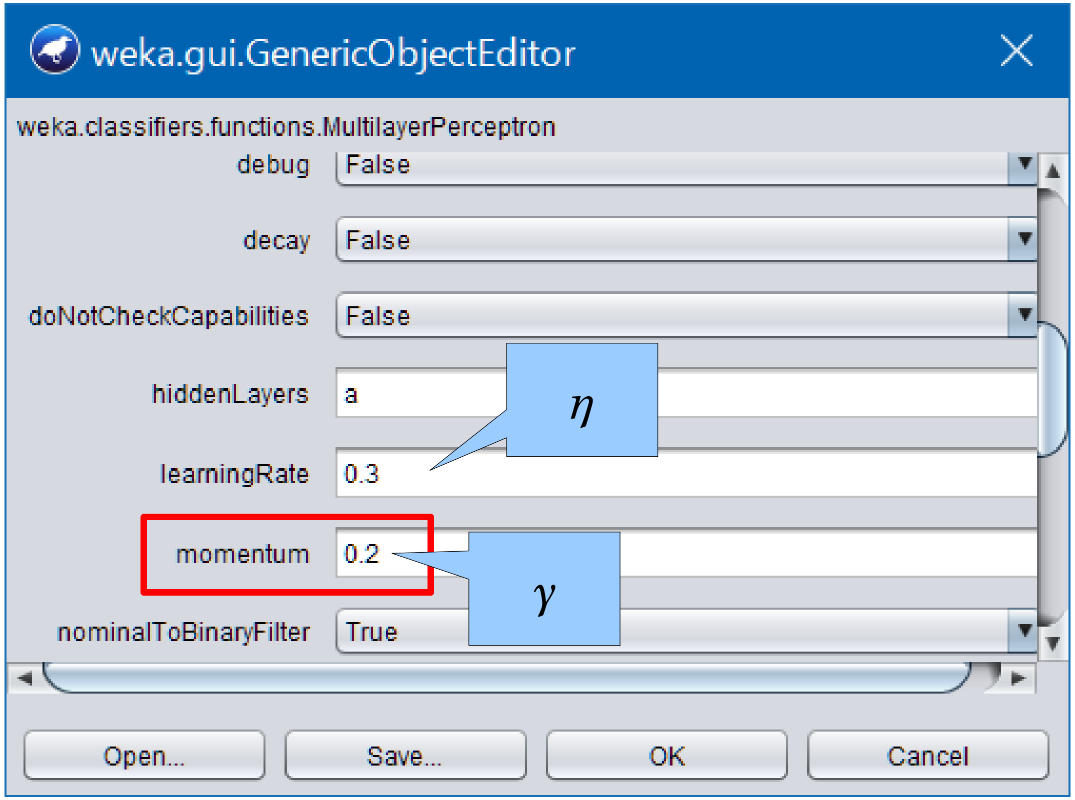
(a) rectified linear 関数



(b) (a)の導関数

学習時のパラメータ

- Weka でのパラメータ調整
  - モーメンタム（慣性）　$\boldsymbol{v}$
    - 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



$$\boldsymbol{v}_t = \gamma \boldsymbol{v}_{t-1} + \eta \frac{\partial E}{\partial \boldsymbol{w}}$$

$$\boldsymbol{w}' = \boldsymbol{w} - \boldsymbol{v}_t$$

学習時のパラメータ

- 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)

ID: 0814

学習時のパラメータ

- 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