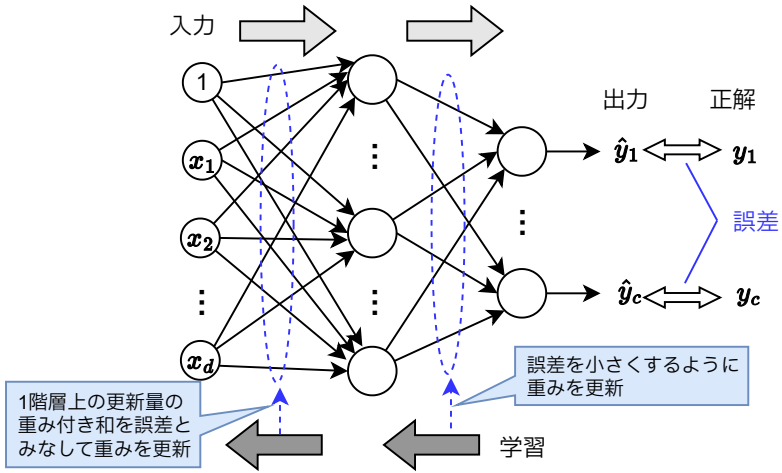
8. ニューラルネットワークの基礎

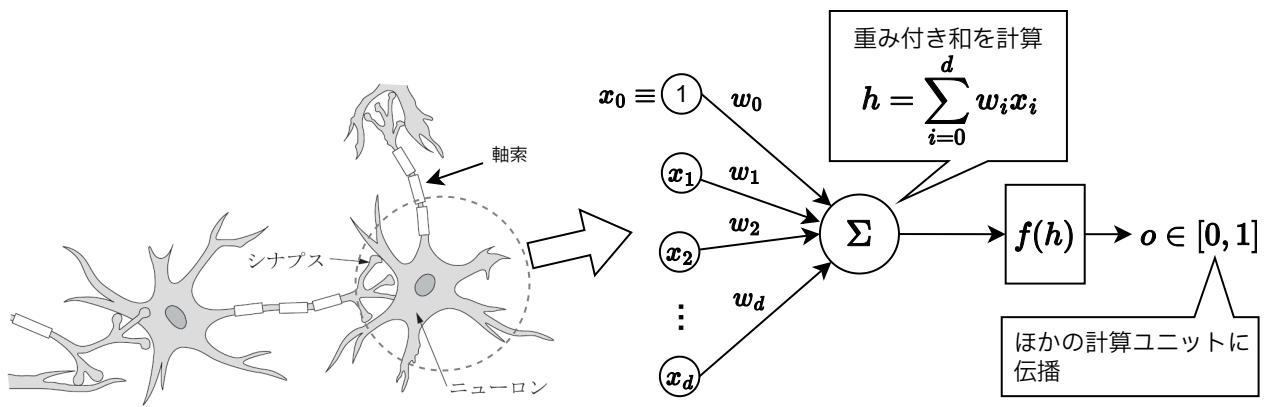


- 荒木雅弘：『Pythonではじめる機械学習』（森北出版，2025年）
- スライドとコード

- 8.1 ニューラルネットワークの計算ユニット
- 8.2 フィードフォワード型ニューラルネットワーク
- 8.3 フレームワークを用いたFNNのコーディング
- 8.4 ニューラルネットワークの深層化

8.1 ニューラルネットワークの計算ユニット (1/3)

- ニューラルネットワークとは
  - 神経細胞の情報伝達メカニズムを単純化したユニットを用いた計算機構
    - シナプスから受け取る神経伝達物質の量が一定量を超えると，その細胞も興奮して神経伝達物質を分泌するメカニズムを数理的にモデル化したもの
    - 現時点では脳の複雑な機能分化などがモデル化できているわけではない



(a) ニューロンとその結合　　(b) ニューロンの数理モデル

8.1 ニューラルネットワークの計算ユニット (2/3)

- 初期のニューロンモデル（McCulloch&Pittsモデル）
  - 活性化関数として閾値関数を用いる

$$f(h) = \begin{cases} 0 & (h < 0) \\ 1 & (h \geq 0) \end{cases}$$

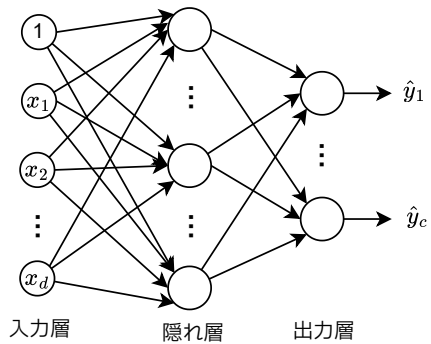
  - パーセプトロンの実装と等価
    - 線形分離可能なデータに対して，$\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} = 0$という識別面を学習可能

8.1 ニューラルネットワークの計算ユニット (3/3)

- 線形分離可能性に関係なく，任意のデータで学習可能なユニットへ
  - 活性化関数として微分可能なシグモイド関数 $\sigma(h) = \frac{1}{1+\exp(-h)}$ を用いる
  - ロジスティック回帰の実装と等価
    - 勾配降下法により，クロスエントロピー（負の対数尤度）最小となる識別面 $\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} = 0$ を学習可能
  - シグモイド関数の微分は $\sigma'(h) = \sigma(h)(1 - \sigma(h))$ と簡単な形になる

8.2 フィードフォワード型ニューラルネットワーク (1/8)

- FNN (Feedforward Neural Network) の構造
  - 非線形関数ユニットの階層的組み合わせで複雑な非線形識別面が実現できる



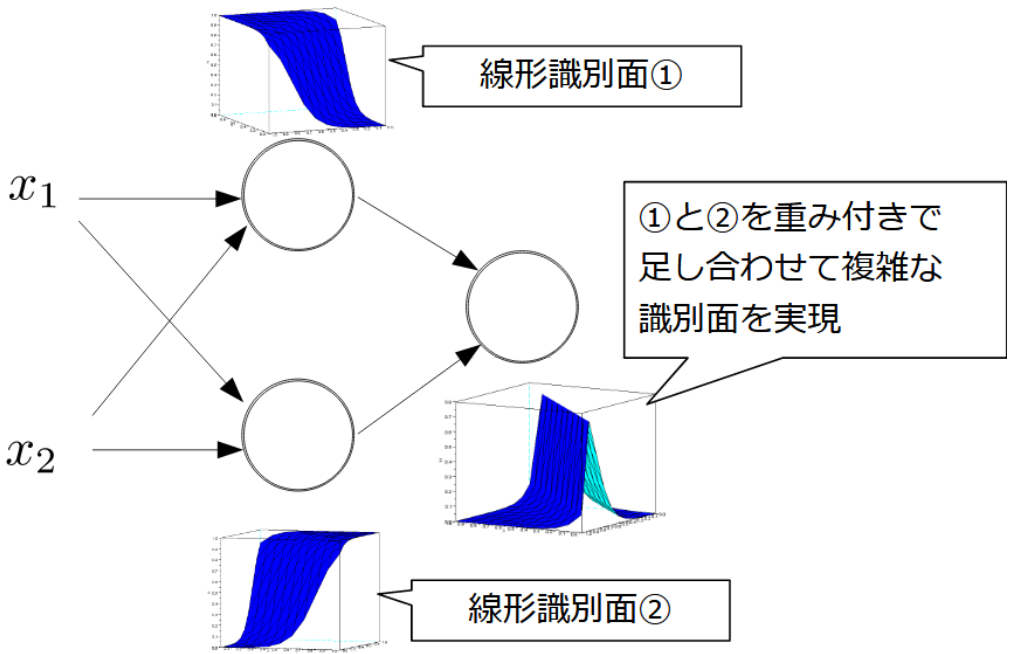
  - 多クラス識別の出力層には活性化関数として以下の softmax 関数を用いる

$$\hat{y}_k = \frac{\exp(h_k)}{\sum_{j=1}^{c} \exp(h_j)}$$

    - h_k: k番目の出力層ユニットに入力される隠れ層の出力の重み付き和

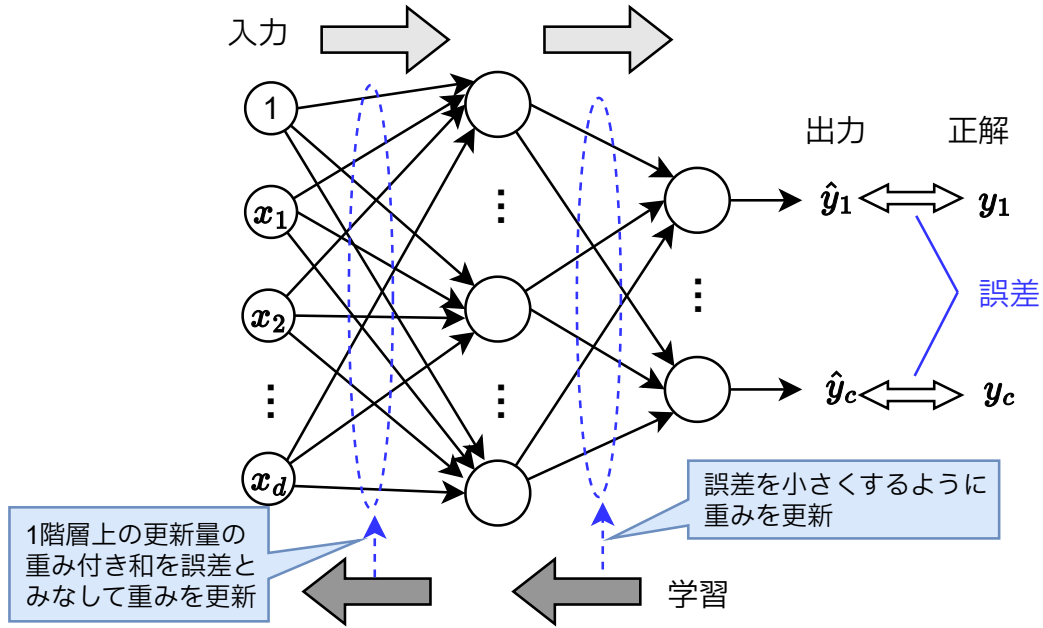
8.2 フィードフォワード型ニューラルネットワーク (2/8)

- FNN による非線形識別面の実現



8.2 フィードフォワード型ニューラルネットワーク (3/8)

- 誤差逆伝播法による学習のイメージ



8.2 フィードフォワード型ニューラルネットワーク (4/8)

- 勾配降下法による学習の準備
  - 学習データ: $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{y} \in \{0,1\}^c$ （c次元one-hotベクトル）

$$\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\} \quad (i = 1, \ldots, N)$$

  - 特定のデータ $\boldsymbol{x}_i$ に対する出力 $\hat{\boldsymbol{y}}_i$ から求める二乗誤差

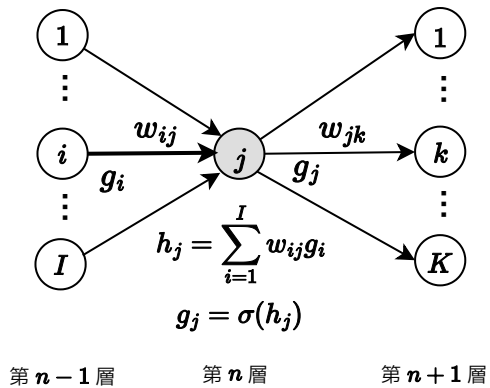
$$E(\boldsymbol{w}) \equiv \frac{1}{2} \sum_{j=1}^{c} (\hat{y}_j - y_j)^2$$

  - 確率的勾配降下法による重み $\boldsymbol{w}$ の更新式（η は学習率）

$$\boldsymbol{w}' \leftarrow \boldsymbol{w} - \eta \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}$$

8.2 フィードフォワード型ニューラルネットワーク (5/8)

- 修正量の計算
  - 第n-1層のi番目のユニットから第n層のj番目のユニットへの重みw_{ij}の更新を考える



  - 修正量の計算に合成関数の微分公式を適用

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{ij}} = \frac{\partial E(\boldsymbol{w})}{\partial h_j} \frac{\partial h_j}{\partial w_{ij}} \qquad (1)$$

8.2 フィードフォワード型ニューラルネットワーク (6/8)

- 修正量の計算の分解
  - (1)の右辺第1項を誤差信号 ϵ_j と置き，合成関数の微分公式を適用

$$\epsilon_j = \frac{\partial E(\boldsymbol{w})}{\partial h_j} = \frac{\partial E(\boldsymbol{w})}{\partial g_j}\frac{\partial g_j}{\partial h_j} \qquad (2)$$

  - (1)の右辺第2項
    - $h_j = \sum_{i=1}^{I} w_{ij} g_i$ から $\frac{\partial h_j}{\partial w_{ij}} = g_i$

8.2 フィードフォワード型ニューラルネットワーク (7/8)

- 誤差項の分解
  - (2)の右辺第1項
    - 第n層が出力層の場合

$$\frac{\partial E(\boldsymbol{w})}{\partial g_j} = g_j - y_j$$

    - 第n層が隠れ層の場合

$$\frac{\partial E(\boldsymbol{w})}{\partial g_j} = \sum_{k=1}^{K} \frac{\partial E(\boldsymbol{w})}{\partial h_k} \frac{\partial h_k}{\partial g_j} = \sum_{k=1}^{K} \epsilon_k w_{jk}$$

  - (2)の右辺第2項 : 活性化関数（シグモイド関数）の微分 $g_j(1 - g_j)$

8.2 フィードフォワード型ニューラルネットワーク (8/8)

- 誤差逆伝播法による学習の手順
  - 入力: 学習データ $\boldsymbol{X}, \boldsymbol{y}$
  - 出力: 学習後の FNN

1. FNN のリンクの重み $\boldsymbol{w}$ を小さな初期値に設定
2. 事前に設定したエポック数（繰り返し回数）だけ，以下を繰り返す

for $\boldsymbol{x} \in \boldsymbol{X}$

ネットワークの出力 $\hat{\boldsymbol{y}}$ を計算

for 出力層から入力層に向かって順に

if 出力層:

各ユニットのエラー量 $\epsilon = (\hat{y}_k - y_k)\hat{y}_k(1 - \hat{y}_k)$ を計算

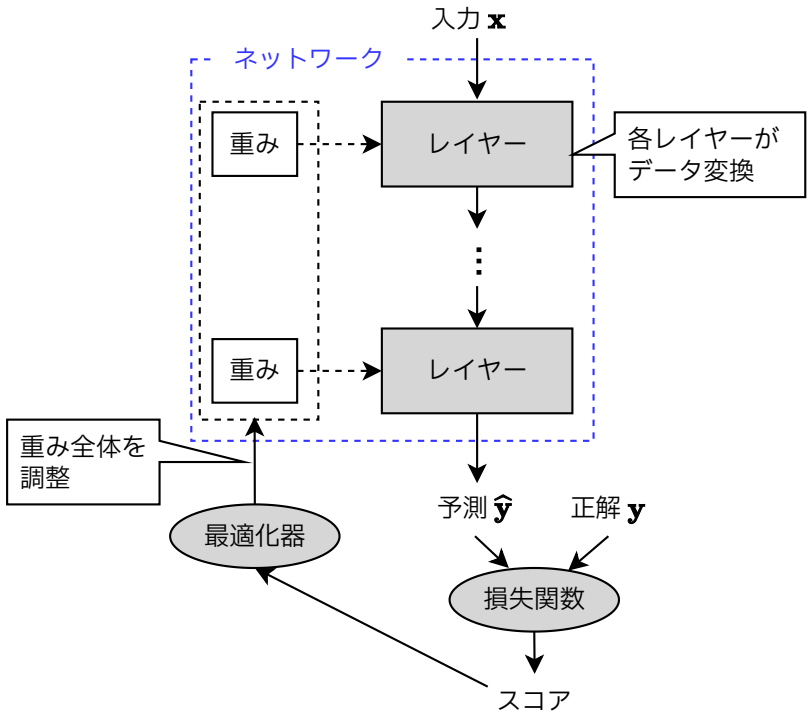
else

各ユニットのエラー量 $\epsilon = \sum_{k=1}^{K} \epsilon_k w_{jk} g_j(1 - g_j)$ を計算

各ユニットに至る重みの更新 $w' = w - \eta\, \epsilon\, g$

8.3. フレームワークを用いた FNN のコーディング (1/7)

- 深層学習ライブラリ keras の枠組み



8.3. フレームワークを用いた FNN のコーディング (2/7)

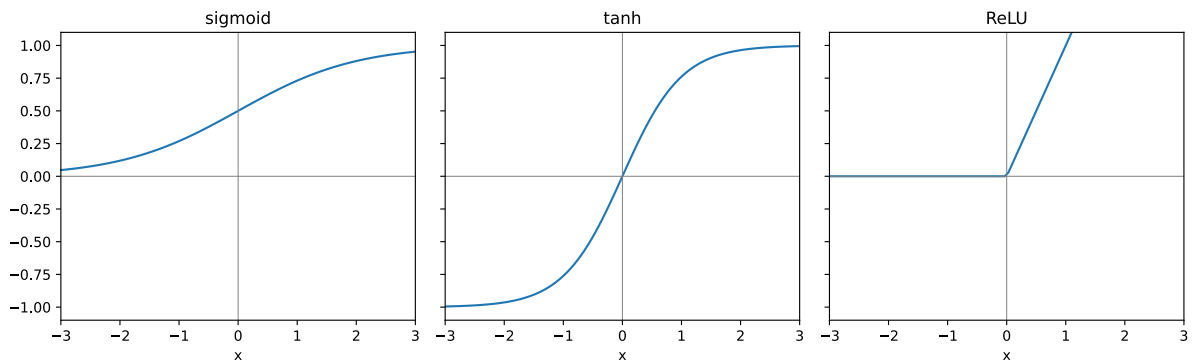
- ネットワークはレイヤーを積み重ねて定義
  - 2層 FNN の例

```
model = keras.Sequential([
    keras.Input(shape=(d,)),
    layers.Dense(n_hidden, activation="relu"),
    layers.Dense(n_class, activation='softmax'),
    ])
```

  - レイヤーの種類
    - Dense: 密結合層
      - 隣接する層間のすべてのユニット間で結合をもつ
    - Flatten: 入力情報の変換
      - 2次元データを1次元のベクトルに変換

8.3. フレームワークを用いた FNN のコーディング (3/7)

- 活性化関数：レイヤーの activation 属性で指定
  - 'softmax' : ソフトマックス関数
  - 'sigmoid' : シグモイド関数　$f(x) = 1/(1 + exp(-x))$
  - 'tanh' : 双曲線正接　$f(x) = \tanh(x)$
  - 'relu' : rectified linear関数　$f(x) = \max(0, x)$



8.3. フレームワークを用いた FNN のコーディング (4/7)

- 損失関数と最適化器を指定してモデルをコンパイル

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

- metrics は，各エポックで学習が進んでいることを確認するための評価指標
  - 'acc' : 正解率
  - 'mse' : 平均二乗誤差

8.3. フレームワークを用いた FNN のコーディング (5/7)

- 損失関数
  - 回帰
    - 二乗誤差: 'mean_squared_error'
    - 外れ値の影響を小さくしたい場合はHuber損失: 'Huber'
      - 一定の範囲内は二乗誤差，範囲外は線形損失
  - 識別
    - 2値識別: 2値クロスエントロピー 'binary_crossentropy'

      $E(\boldsymbol{w}) = -\{y \log(\hat{y}) + (1-y) \log(1-\hat{y})\}$
    - 多クラス識別: クロスエントロピー: 'categorical_crossentropy'
      - 2つの確率分布 y と $\hat{y}$ の近さを表す

        $E(\boldsymbol{w}) = -\sum_{j=1}^{c} y_j \log(\hat{y}_j)$

8.3. フレームワークを用いた FNN のコーディング (6/7)

- 最適化器
  - 確率的勾配降下法(SGD)
    - モーメンタム（慣性）の導入
  - 準ニュートン法(L-BFGS)
    - 2次微分（近似）を更新式に加える
  - AdaGrad
    - 学習回数と勾配の2乗を用いた学習係数の自動調整
  - RMSProp
    - 学習係数調整の改良：勾配の2乗の指数平滑移動平均を用いることで直近の変化量を反映
  - Adam: Adaptive Moment Estimation
    - 分散に関するモーメントも用いて，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 L-BFGS が勧められている

8.3. フレームワークを用いた FNN のコーディング (7/7)

- 学習
  - ミニバッチのサイズとエポック数（繰り返し回数）を指定
    - 繰り返し毎に損失関数の値とmetricsで指定した値が表示される

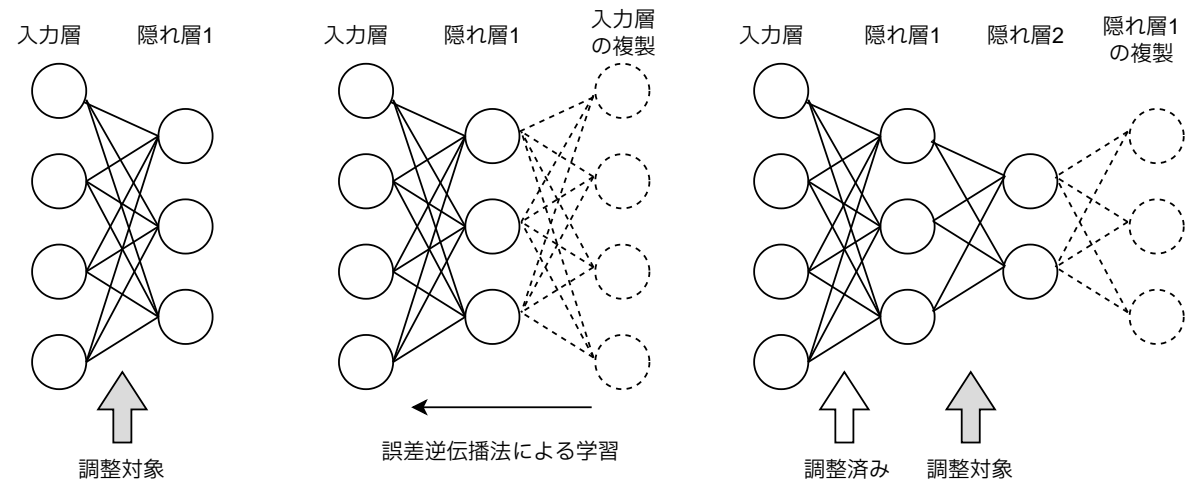
```
model.fit(X_train, y_train, batch_size=200, epochs=3)
```

- 評価
  - score0は損失関数の値
  - score1以降は metrics で指定したもの

```
score = model.evaluate(X_test, y_test)
```

8.4 ニューラルネットワークの深層化 (1/7)

- 勾配消失問題への初期の対処法
  - 入力側から順にオートエンコーダなどによって重みを事前学習
  - その後，全体を誤差逆伝播法で微調整



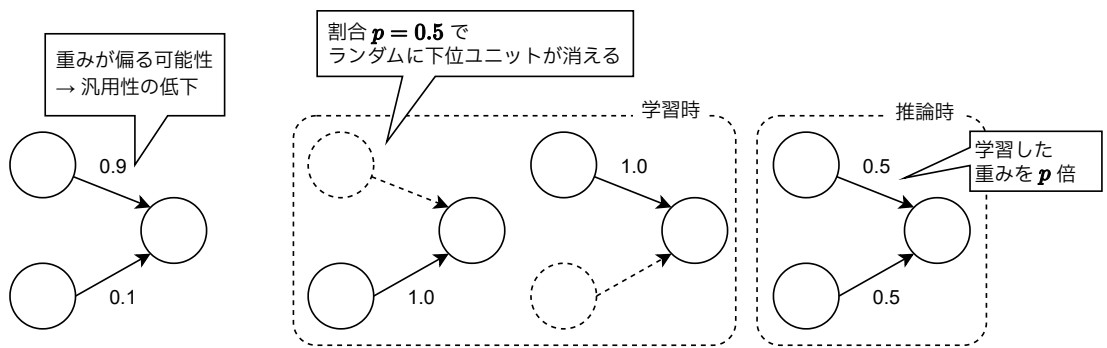
(a) 事前学習対象の重み　(b) オートエンコーダによる自己教師あり学習　(c) 1階層上の事前学習

8.4 ニューラルネットワークの深層化 (2/7)

- 勾配消失問題への対処法の発展
  - 活性化関数の工夫により，事前学習の必要は薄れてきた
    - ReLU(rectified linear) : $f(x) = \max(0, x)$
      - 勾配消失が起こりにくい
      - 0 を出力するユニットが多くなる
    - 双曲線正接 tanh : $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
      - 微分の値が大きい
      - 負の値でも勾配がある

8.4 ニューラルネットワークの深層化 (3/7)

- 過学習への対処
  - ドロップアウト
    - 学習時に一定割合のユニットをランダムに消す
    - 認識時には学習後の重みに消去割合を掛ける
    - 正規化のような役割



(a) 通常の学習　　(b) ドロップアウト

8.4 ニューラルネットワークの深層化 (4/7)

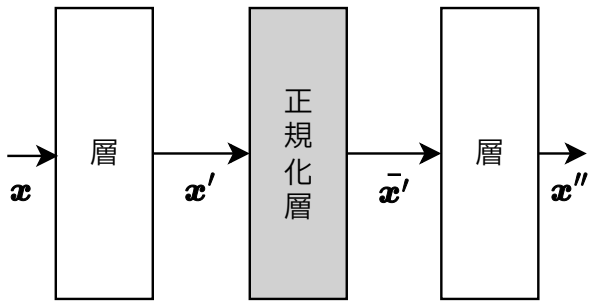
- 学習の安定化，高速化
  - バッチ正規化の必要性
    - データが空間内の特定の領域に偏ってしまうと，層による非線形変換が生じにくい
      - たとえば活性化関数をReLUとしたとき，データが0をまたぐことで非線形性が生じる
      - データの平均が0近辺で，分散のスケールが一定のときは学習させやすい
    - バッチ入力 $\{\boldsymbol{x}^{(i)}\}\quad i=1,\ldots,|B|$ に対して，変換結果 $\boldsymbol{h}^{(i)}$ を平均0，分散1に変換

$$\boldsymbol{m}=\frac{1}{|B|}\sum_{i=1}^{|B|}\boldsymbol{x}^{(i)},\ \boldsymbol{v}=\frac{1}{|B|}\sum_{i=1}^{|B|}(\boldsymbol{x}^{(i)}-\boldsymbol{m})^2$$

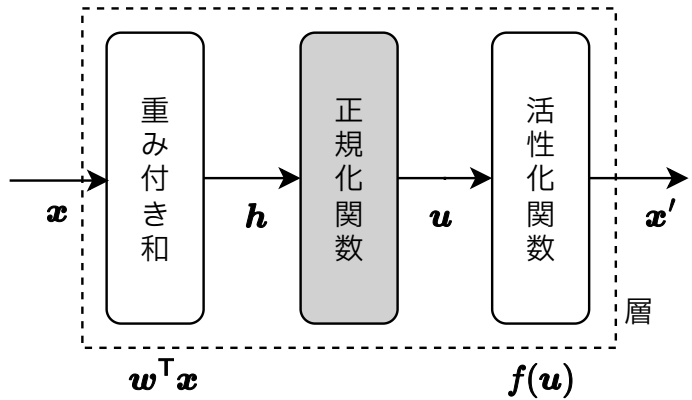
$$\boldsymbol{x}'^{(i)}=\frac{\boldsymbol{x}^{(i)}-\boldsymbol{m}}{\sqrt{\boldsymbol{v}+\epsilon}},\ \boldsymbol{h}^{(i)}=\gamma\boldsymbol{x}'^{(i)}+\beta$$

8.4 ニューラルネットワークの深層化 (5/7)

- 学習の安定化，高速化
  - バッチ正規化の方法



(a) 正規化層を設定する場合

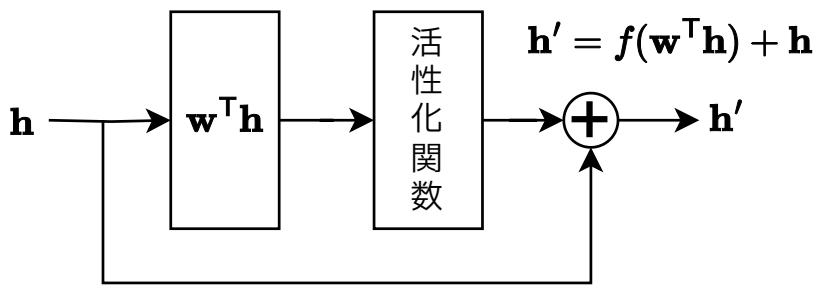


(b) 正規化関数を設定する場合

8.4 ニューラルネットワークの深層化 (6/7)

- 学習の安定化，高速化
  - スキップ接続
    - 層による変換を行わないデータの流れを追加して，そのデータを出力に加える
    - より学習が容易な残差をモデル化しているとみなせる

$$f(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{h}) = \boldsymbol{h}' - \boldsymbol{h}$$



8.4 ニューラルネットワークの深層化 (7/7)

- 深層学習における現実と理論のギャップ
  - 多層にすると性能が向上する理由
    - 関数がジャンプを持つ場合：階段関数で近似
    - 関数が非均一的な滑らかさを持つ場合：異なる幅を持つ短冊状の関数で近似
  - 過学習しにくい理由
    - 暗黙的正則化：巨大なDNNは小さなNNの集合体で，その中の1つが当たりを引き当てている
    - 二重降下：パラメータ数がデータ数より多くなると汎化誤差は下がり続ける
  - 最適解がみつかりやすい理由
    - 過剰パラメータを持つ層が損失関数全体を押し下げ，損失関数の値が0となるところが多く現れる

まとめ

- ニューラルネットは，ロジステック回帰を多段階にしたもので，非線形識別面を実現している
- ニューラルネットは誤差逆伝播法で学習する
- kerasを用いたFNNのコーディング
- 多階層ニューラルネットの学習
  - 参考）今泉 允聡: 深層学習の原理に迫る　数学の挑戦 (岩波科学ライブラリー), 岩波書店, 2021.