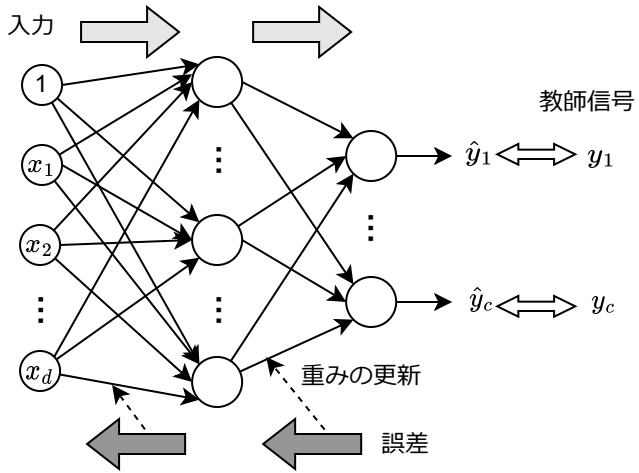
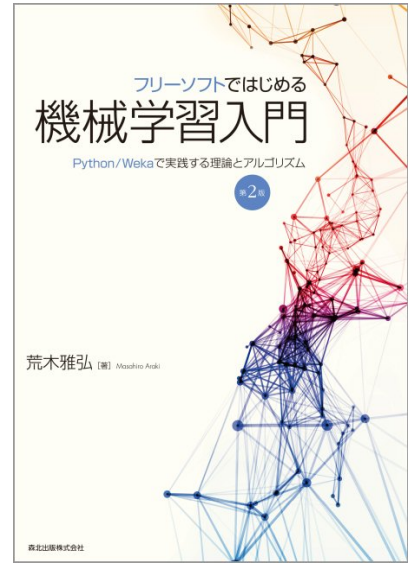
8. ニューラルネットワーク



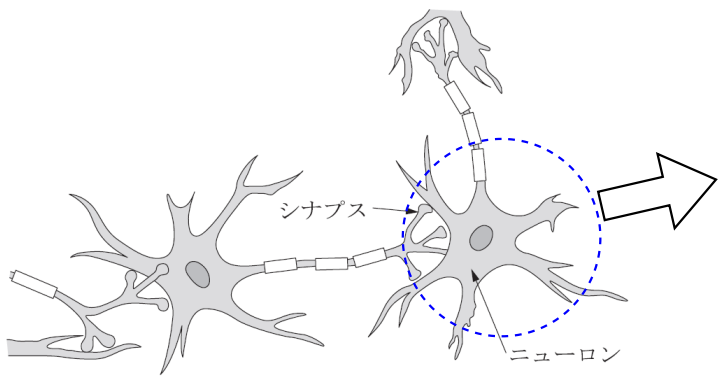
- 8.1 ニューラルネットワークの計算ユニット
- 8.2 フィードフォワード型ニューラルネットワーク
- 8.3 ニューラルネットワークの深層化



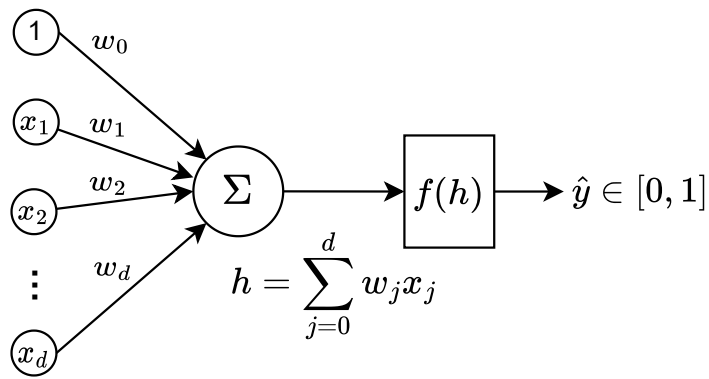
- 荒木雅弘:『フリーソフトではじめる機械学習入門(第2版)』(森北出版, 2018年)
- スライドとJupyter notebook
- サポートページ

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）

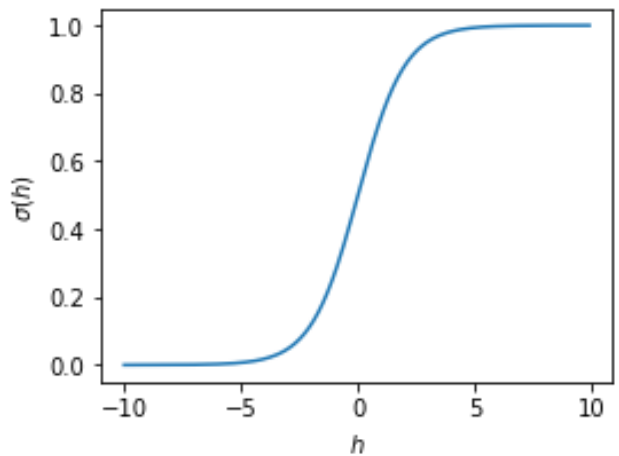
- 初期のニューロンモデル（McCulloch&Pittsモデル）
  - 活性化関数として閾値関数を用いる

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

  - 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、$\boldsymbol{w}^T \boldsymbol{x} = 0$という識別面を学習可能

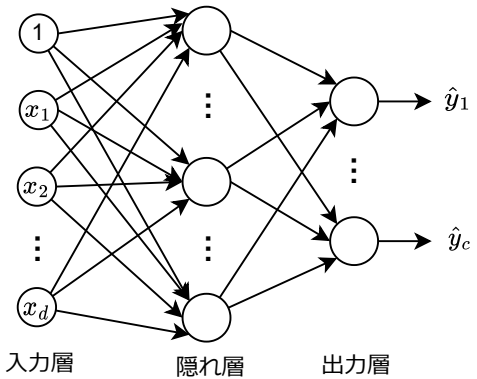
8.1 ニューラルネットワークの計算ユニット（3/3）

- 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 $\boldsymbol{w}^T\boldsymbol{x} = 0$ 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）

- ニューラルネットワークの計算ユニットを階層的に組み合わせる
  - 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

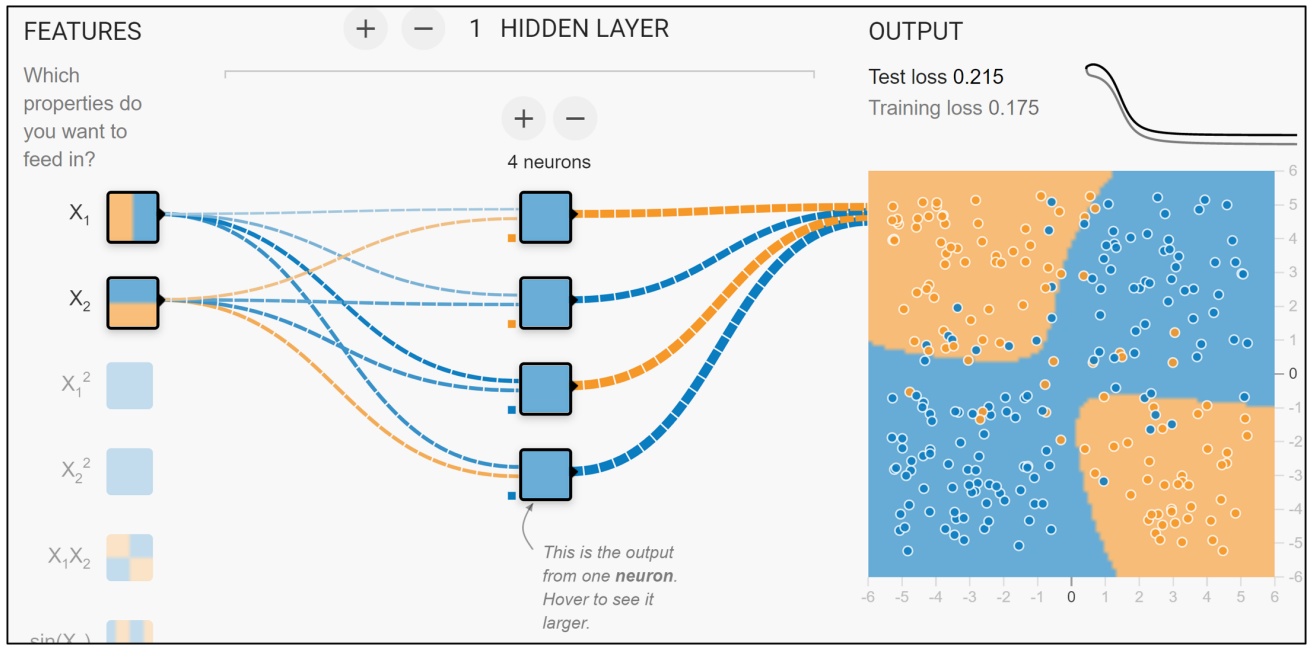


  - 多クラス識別の出力層には活性化関数として以下のsoftmax関数を用いる
    - h_k: k番目の出力層ユニットに入力される隠れ層の出力の重み付き和

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

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）

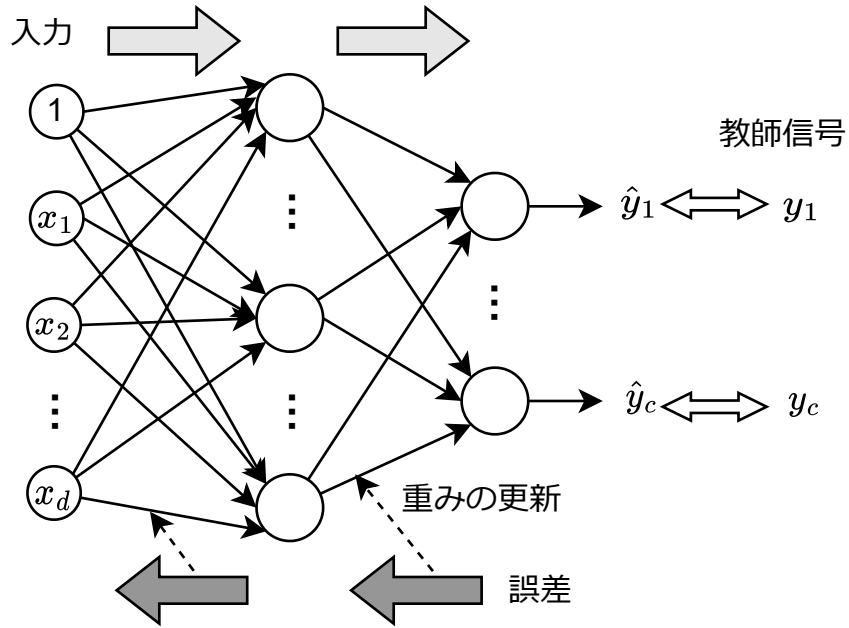
- フィードフォワード型ネットワークによる複雑な識別面の構成



  - https://playground.tensorflow.org/

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 $\boldsymbol{x}$ に対する二乗誤差

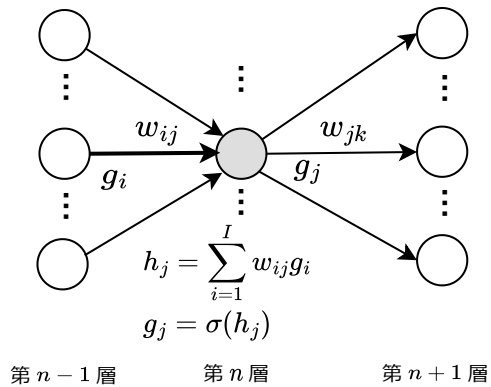
$$E(\boldsymbol{w}) \equiv \frac{1}{2} \sum_{j=1}^{c} (\hat{y}_j - y_j)^2$$

  - 確率的勾配降下法による重み w の更新式

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

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 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 $g_j(1 - g_j)$

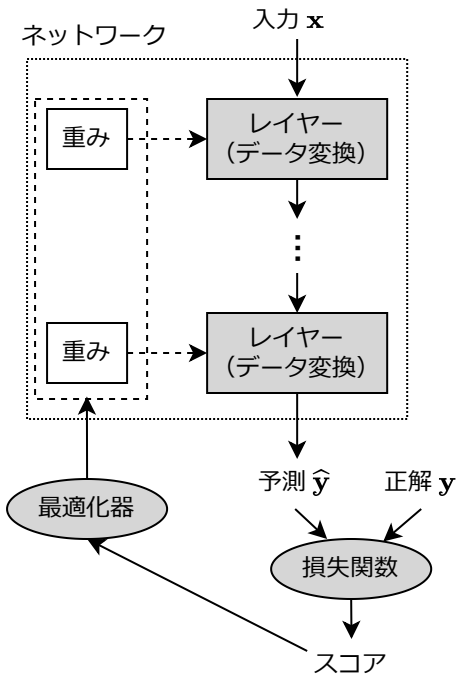
8.2 フィードフォワード型ニューラルネットワーク（8/8）

- 誤差逆伝播法による学習の手順
  1. リンクの重み $\boldsymbol{w}$ を小さな初期値に設定
  2. 個々の学習データ $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ に対して以下繰り返し
     1. 入力 $\boldsymbol{x}_i$ に対するネットワークの出力 $\hat{\boldsymbol{y}}_i$ を計算
     2. 出力層の k 番目のユニットに対してエラー量 $\epsilon_k = (\hat{y}_k - y_k)\hat{y}_k(1 - \hat{y}_k)$ を計算
     3. 出力層に近い隠れ層から順に、j 番目のユニットに対してエラー量 $\epsilon_j = \sum_{k=1}^{K} \epsilon_k w_{jk} g_j(1 - g_j)$ を計算
     4. 以下、入力層に至るまでエラー量の計算を繰り返し
     5. 重みの更新

$$w' = w - \eta \epsilon g$$

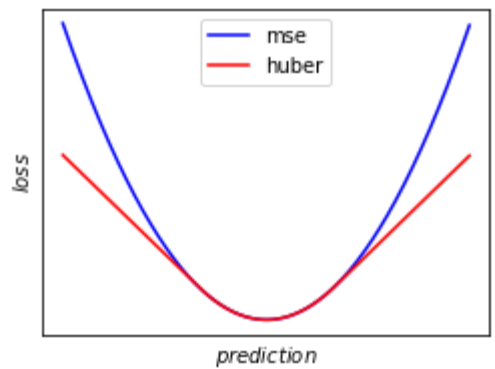
ニューラルネットワークによる学習の枠組み（1/5）

- 深層学習ライブラリ keras の枠組み



ニューラルネットワークによる学習の枠組み（2/5）

- 回帰問題の損失関数
  - 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



ニューラルネットワークによる学習の枠組み（3/5）

- 識別問題の損失関数
  - 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

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

ニューラルネットワークによる学習の枠組み（4/5）

- 勾配降下法による最適化
  - モーメンタム（慣性）の導入（最適化器SGDのmomentum属性で γ を指定）
    - 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

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

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

ニューラルネットワークによる学習の枠組み（5/5）

- 実用的な最適化器
  - 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
    - 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 L-BFGS が勧められている

kerasのコーディング（1/4）

- ニューラルネットワークの構造と活性化関数の指定

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='sigmoid'),
    keras.layers.Dense(10,  activation='softmax')
])
```

- レイヤーの種類
  - 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
  - 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

kerasのコーディング (2/4)

- Activation: 活性化関数
  - '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　$f(x) = \max(0, x)$

kerasのコーディング（3/4）

- modelのコンパイル

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

- 損失関数、最適化器、評価指標（複数可）を指定
  - optimizer: 最適化手法
    - 'sgd' : 確率的最急降下法
    - 'adam' : Adaptive Moment Estimation
  - metrics: 評価指標
    - 'acc' : 正解率
    - 'mse' : 平均二乗誤差

kerasのコーディング（4/4）

- 学習
  - ミニバッチのサイズと繰り返し数を指定
    - 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
  - score[0]は損失関数の値
  - score[1]以降はmetricsで指定したもの

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

8.3 ニューラルネットワークの深層化

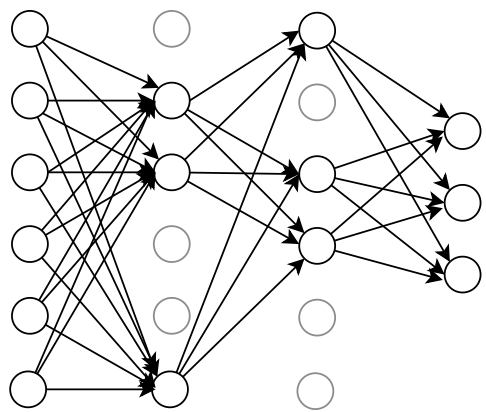
- ニューラルネットワークの構造の決定
  - 隠れ層のユニット数：その層で実現される非線形変換の複雑さを表す
  - 層数：低次の特徴表現から高次の特徴表現への変換の段階数を表す
- 多階層における誤差逆伝播法の問題点
  - 修正量が消失／発散する

8.3 ニューラルネットワークの深層化

- 勾配消失問題への対処
  - 初期はオートエンコーダなどによる重みの事前学習が用いられた
  - 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)：$f(x) = \max(0, x)$
      - 誤差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

8.3 ニューラルネットワークの深層化

- 多階層学習における工夫
  - 過学習の回避: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
  - ドロップアウトの効果
    - 正規化のような役割



8.4 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を用いたニューラルネットのコーディング
- 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.

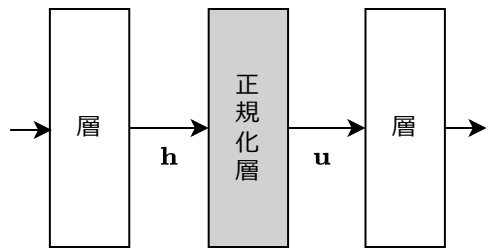
補足　ニューラルネットワークの深層化

- 多階層学習における工夫
  - 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

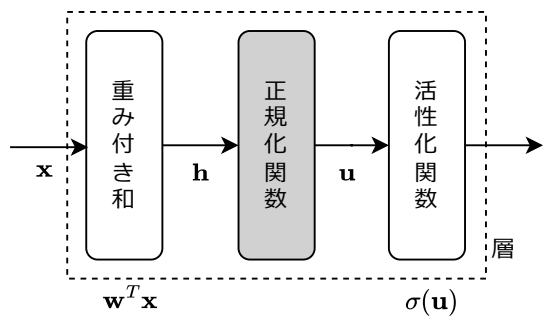
$$\boldsymbol{m}=\frac{1}{|B|}\sum_{i=1}^{|B|}\boldsymbol{h}^{(i)},\quad \boldsymbol{v}=\frac{1}{|B|}\sum_{i=1}^{|B|}(\boldsymbol{h}^{(i)}-\boldsymbol{m})^2,\quad \boldsymbol{u}^{(i)}=\frac{\boldsymbol{h}^{(i)}-\boldsymbol{m}}{\sqrt{\boldsymbol{v}}}$$

補足 ニューラルネットワークの深層化

- 多階層学習における工夫
  - 正規化
    - 正規化層を設ける場合：層の間に入れる



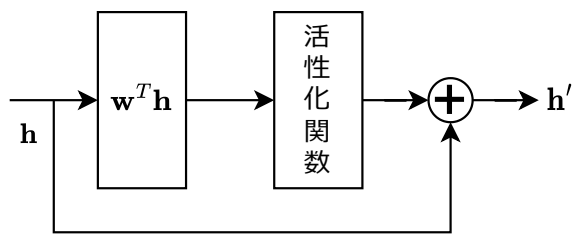
    - 正規化関数を設定する場合：重み付き和と活性化関数の間に入れる



補足 ニューラルネットワークの深層化

- 多階層学習における工夫
  - スキップ接続
    - 層による変換をスキップして出力に加える
      - 学習時に誤差がそのまま伝わる
      - 現在の入力を逐次更新することで重要な情報を失わない
      - 残差をモデル化しているとみなせる（ResNet）

$$\boldsymbol{h}' = \boldsymbol{h} + f(\boldsymbol{h}) \Leftrightarrow f(\boldsymbol{h}) = \boldsymbol{h}' - \boldsymbol{h}$$



補足 ニューラルネットワークの深層化

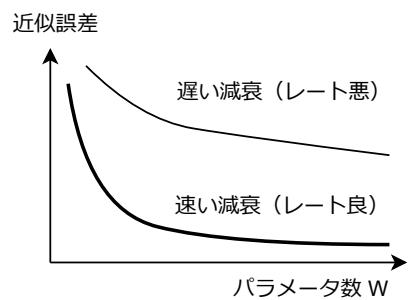
- 深層学習における現実と理論のギャップ
  - なぜ多層にすると性能が向上する？
    - 2層NN隠れ層のユニット数が十分に多ければ、任意の非線形関数を十分な精度で近似できるはず
  - なぜ過学習しない？
    - 「パラメータが多い＝バイアスが小さい」モデルは過学習しやすいはず
  - なぜ最適に近いパラメータが見つかる？
    - 損失関数が複雑な形をしていると、局所最適解で学習が止まることが多いはず

補足 ニューラルネットワークの深層化

- なぜ多層にすると性能が向上する？
  - 関数の表現可能性ではなく、近似の速度に着目する
    - 近似誤差レート α（パラメータ数と関数近似誤差の関係）による評価
      - 表現したい関数 $f(x)$, NNによる関数 $g_\theta(x)$, NNのパラメータ数 W, 定数 $C > 0$

$$\min_{\theta} \max_{x} |f(x) - g_\theta(x)| \leq CW^{-\alpha}$$

      - パラメータ数 W を増加させたときの近似誤差の減少率が、近似誤差レート α で表現できる
      - よいレートを持つ手法は、少ないパラメータで目的の関数を正確に達成できる



補足 ニューラルネットワークの深層化

- なぜ多層にすると性能が向上する？
  - DNNによる近似誤差レート改善の可能性
    - 関数が滑らかな場合
      - 2層NNの近似誤差レートをDNNで改善することはできない
    - 関数がジャンプを持つ場合
      - DNNでは滑らかな関数を表現する層の間に階段関数の層を挟むことで近似可能
    - 関数が非均一的な滑らかさを持つ場合
      - DNNでは前半の層でデータの分割を行い、後半の層で層毎に異なる幅を持つ短冊状の関数を表現することで近似可能
  - 特徴抽出処理との関係
    - 抽出される特徴量が低次元かつ十分滑らかな構造を持つ場合、DNNの近似誤差レートは改善する

補足 ニューラルネットワークの深層化

- なぜ過学習しない？
  - 従来の機械学習の知見:「パラメータが多い＝バイアスが小さい」モデルは過学習しやすい
    - 過学習のしやすさの程度

$$\text{過学習しやすさ} = \sqrt{\frac{NNの自由度}{学習データの数}}$$

    - NNの自由度の定義

$$\text{自由度} = \text{係数} \times \text{層数} \times \text{パラメータ数}$$

補足 ニューラルネットワークの深層化

- 過学習しにくい理由
  - 仮説1：暗黙的正則化
    - 観察される現象：DNNの重みの値は概して小さく、初期値の周辺に限定されている
    - 宝くじ仮説：巨大なDNNは小さなNNの集合体で、学習によって当たりのNNを引き当てている
    - 根拠：学習後に枝刈り（半分以上のエッジの消去）を行っても、ほぼ同じ性能が維持できる
  - 仮説2：損失平坦性
    - 学習後の最適パラメータの近くで損失関数が平坦なら過学習は起こりにくい
    - 入力の摂動によるモデルの変化が小さいことが汎化性能が高いことにつながる
  - 仮説3：二重降下
    - パラメータ数の増加によって、一旦下降した汎化誤差は上昇に転じるが、パラメータ数がデータ数より多くなると再度下降することが実験的に示された
      - データ数・パラメータ数を無限として解析すると、無駄なパラメータが減少し、過学習しやすさはパラメータ数に反比例する

補足 ニューラルネットワークの深層化

- なぜ最適に近いパラメータが見つかる？
  - 局所最適解で学習が止まらない理由
    - 過剰パラメータ化
      - NNのある層のパラメータを一定以上過剰にすると、勾配降下法は損失関数の値をほぼ0にするパラメータに到達する
      - 過剰パラメータを持つ層が損失関数全体を押し下げる
        - 損失関数は負にならないため、0に近い多くの値が最適値となる
    - 確率的勾配降下法の効果の解明
      - 更新されたパラメータの散らばりは、損失関数の値が小さくなる点に集中する分布に従う

補足 ニューラルネットワークの深層化

- 深層学習における現実と理論のギャップ
  - なぜ多層にすると性能が向上する？
    - 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
  - なぜ過学習しない？
    - 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
  - なぜ最適に近いパラメータが見つかる？
    - 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