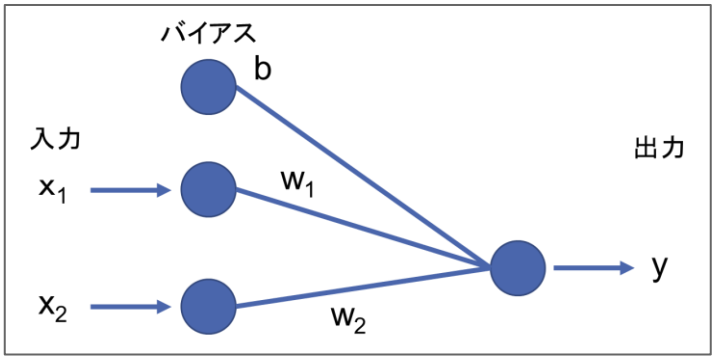
機械学習やアルゴリズムへの理解

本節の内容

e-learnin教材では、ニューラルネットワークの重みパラメータはあらかじめあたえていたが、ここではそれを自動的に求めるための基礎となる概念、手法（損失関数と勾配降下法）について説明する

（人工）ニューラルネットワーク

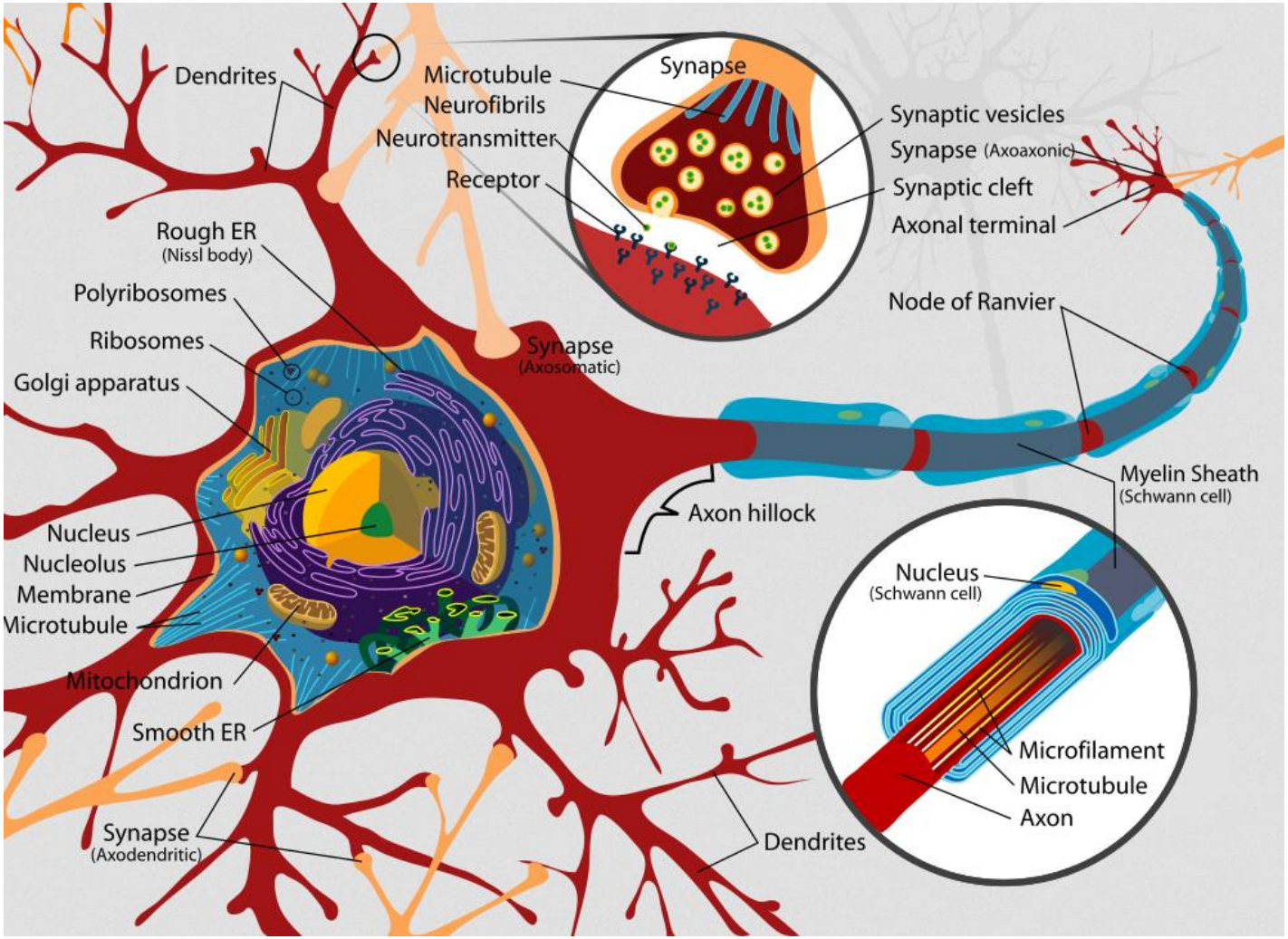
- マカロック=ピッツのモデルを源流とする人工ニューロン（という関数）を次々と組み合わせて構成された関数のこと。複数のニューロン（神経細胞）から、あるニューロンへの信号の入力を重みを持った線形和と活性化関数（後述）により数式化する。
- もっとも原始的な例（右図）
  - 入力層と出力層の2層（数え方によっては1層）のみのニューラルネットワーク
  - 右側の丸の内部では左側からの3つの重み付き和に活性化関数と呼ばれるある関数を施した結果が出力される



$$y = f(w_1 x_1 + w_2 x_2 + b)$$

- ネットワークの重みとバイアスは当初は手で（あるいは勘で）決められたが、より複雑なネットワークは人手では扱うことができず、コンピュータプログラムにより自動的に調整する必要がある。この調整過程が学習と呼ばれる

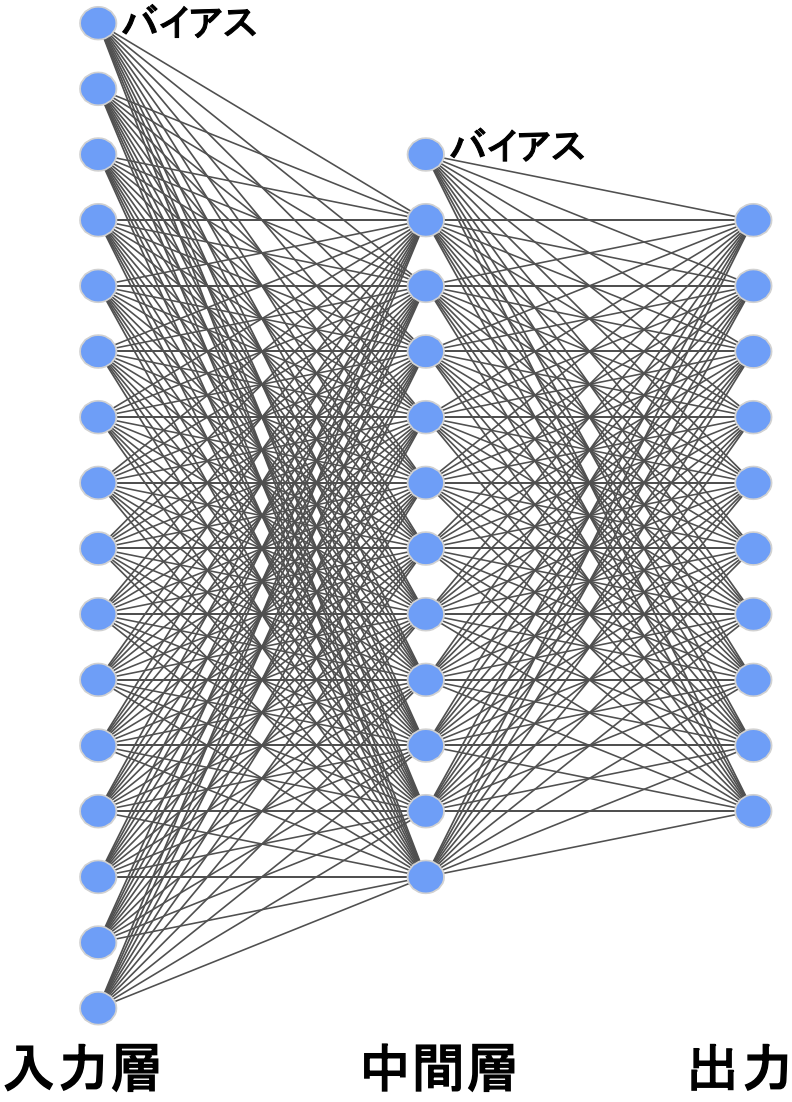
ニューロン



脳科学辞典「シナプス」より

Deep Learning（深層学習）とは

- 多層のニューラルネットワークによる機械学習手法
- 層の数に特に厳密な定義は無い
  - 入力層と出力層以外に層があればDeepと呼ぶこともある
- 層の数が増えると学習が難しく、特殊例を除きあまり実用化されていなかった
- 入力と出力層の間にある層を**中間層**、あるいは**隠れ層**と呼ぶ

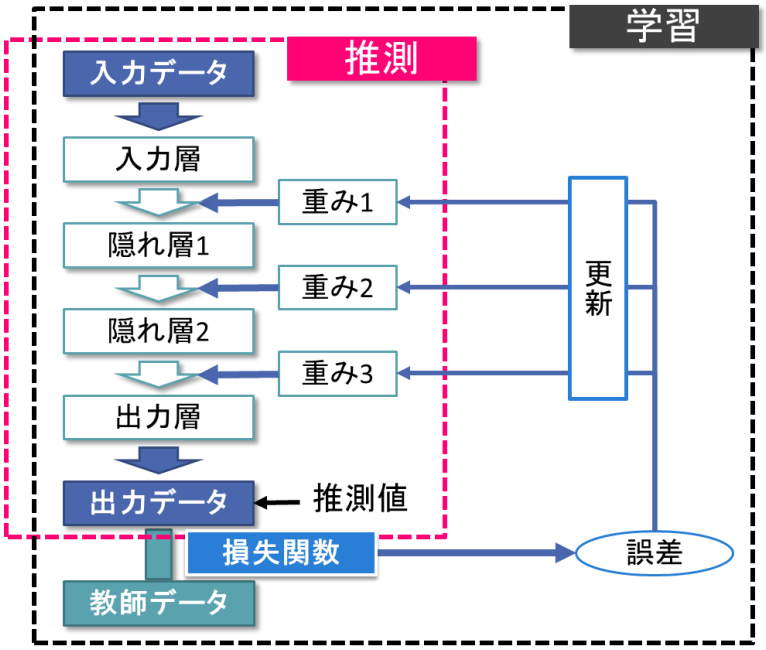


ディープラーニングの学習過程概要

入力をディープニューラルネットワークで計算し、出力を得る

出力を推測値として、教師データとの誤差を用いて各層の重みを更新する

推測と重みの更新を繰り返し、誤差が少なくなるように重みを適正な値に収束させる

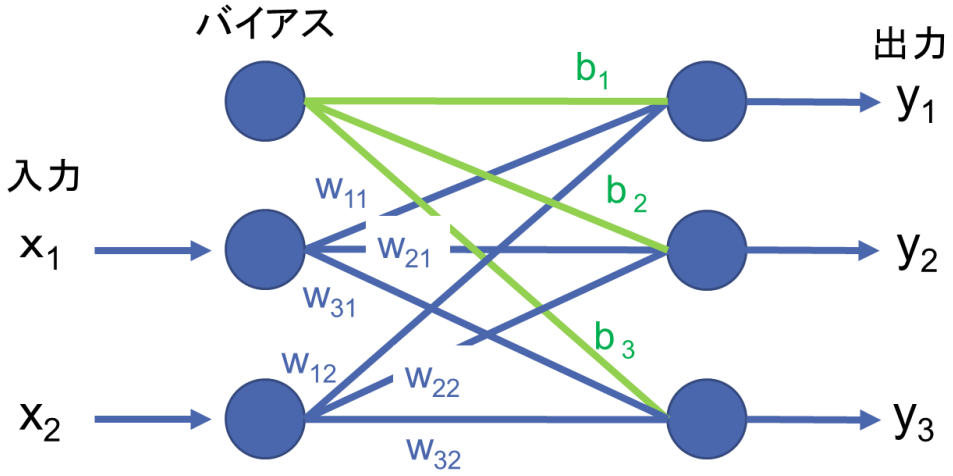


※隠れ層 = 中間層

ニューラルネットワーク内の計算

下の図は最も基本的な全結合ニューラルネットワーク
（第1層は2ユニット、第2層は3ユニット）

重み和の計算は、行列とベクトルの積として表すことができる



$$y_k = w_{k1}x_1 + w_{k2}x_2 + b_k \quad (k = 1, 2, 3)$$

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$= \begin{pmatrix} w_{11} & w_{12} & b_1 \\ w_{21} & w_{22} & b_2 \\ w_{31} & w_{32} & b_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

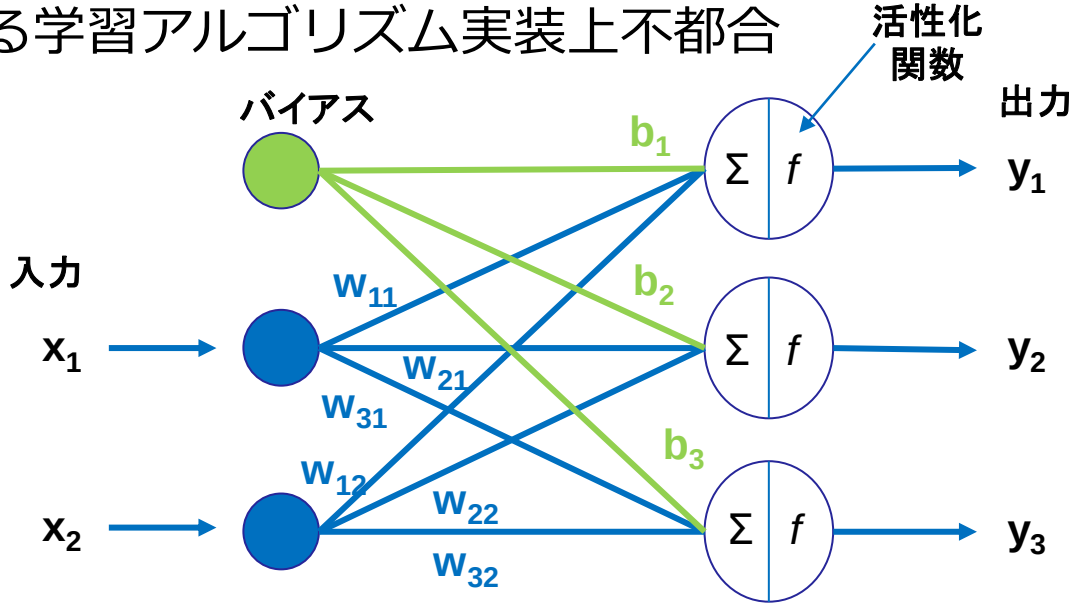
※ 実際は、次に述べる活性化関数をさらに施したものを出力とする

活性化関数（Activation function）

各ニューロン（図式内の丸印）の計算結果を活性化関数で変換して次の層へ送る

ニューロン電位がある閾値を超えると発火（シナプスから顆粒が放出）するという動きを模倣している

この発火の関数モデルは当初はステップ関数（後述）であり、現在用いられる学習アルゴリズム実装上不都合

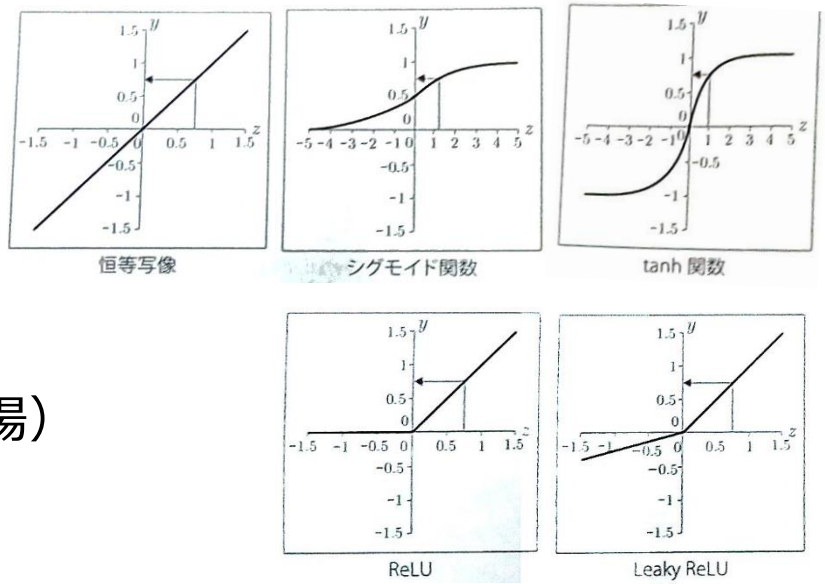


$$y_k = f(w_{k1}x_1 + w_{k2}x_2 + b_k)$$

※ここで、Σは（重み付きの）和をとるという操作を表す

活性化関数の種類（入力:x、出力:y）

- 恒等写像、線形関数
  - $y = x$
- シグモイド関数（Sigmoid）
  - $y = \frac{1}{1+e^{-x}}$
- tanh関数（e-learning教材で登場）
  - $y = \frac{e^{x}-e^{-x}}{e^{x}+e^{-1}}$
- ReLU（Rectified Linear Unit）
  - $y = \max(x,\ 0)$
- Leaky ReLU
  - $y = x \ \ if\ x \geq 0$
    $y = ax \ \ if\ x < 0$
- ソフトマックス関数（Softmax）
  - $y_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$ K：出力層のユニット数



（藤田、高原、「実装ディープラーニング」, p.52）

$0 \leq y_i \leq 1$、$\sum_{i=1}^{K} y_i = 1$ となるため、確率変数とみなせる

活性化関数は他にも種類があるが割愛

損失関数（Loss Function）

推測した出力データと教師データとの間の誤差を計算する関数

n:サンプル番号、y_n：出力データ、t_n：教師データとする

- 平均二乗誤差（Mean Squared Error）　連続値の回帰の場合

$$E_n = \frac{1}{2}(y_n - t_n)^2、\ E = \frac{1}{K}\sum_{n=1}^{K} E_n$$

- クロスエントロピー　出力が離散値の場合
  - $E_n = -\sum_{k=1}^{K} t_{nk} \log y_{nk}、\ E = \sum_{n=1}^{K} E_n$

    t_{nk}：nサンプル目のクラスkの教師データ

    y_{nk}：nサンプル目のクラスkの出力データ
- 2値（バイナリ）クロスエントロピー　2値分類の場合

$$E_n = -\{t_n \log y_n + (1 - t_n)\log(1 - y_n)\}、\ E = \sum_{n=1}^{K} E_n$$

One-hotベクトル

- ラベルデータのベクトル表現
- 総ラベル数Nの i 番目のラベルを、N次元ベクトルの i 番目の要素だけを1とし、その他は0のベクトルで表す
- 例：総ラベル数5（1～5）で3番目ラベルは [0, 0, 1, 0, 0] となる

- 分類問題の教師ラベル表現として使われる
  - 分類問題では出力層のノード数を総ラベル数にし、Softmax関数を活性化関数とする
  - 教師データをOne-hotベクトルで与えると、正解ラベルに相当するノードの出力が1、それ以外のノードの出力が 0 に近づくようにノードの重みを更新する
  - Softmax関数を使うので、それぞれの出力ノードの値は、そのラベルに属する確率とみなすことができる

誤差最小化手法

- 勾配降下法

  重みパラメータ全体ならなるベクトルwは、深層学習の場合、一般に巨大次元のベクトルである。
  このwに対する誤差Eの**勾配** ∇E というベクトルを計算し、これをもとに誤差Eが小さくなる方向にwを更新する（勾配ベクトル ∇E の向きは、Eが大きくなる向きに一致する）

  - $w = w - \varepsilon \nabla E$、　ε：学習係数（0.01や0.001など小さな数）

  - 仮に誤差を二乗誤差 $\mathrm{E} = \frac{1}{2}(Y - t)^2 = \frac{1}{2}(wX - t)^2$ とすると、

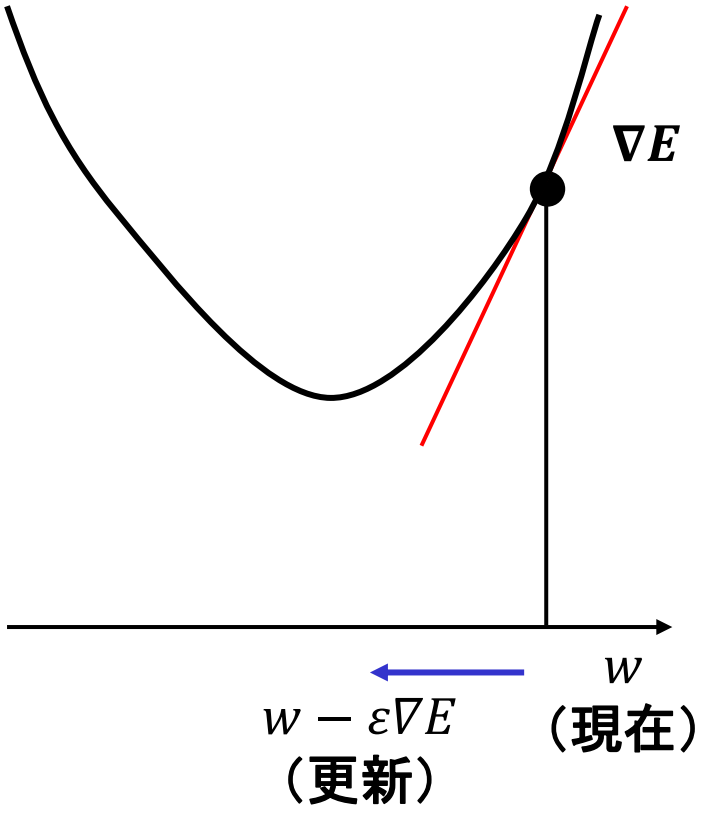
    Eのwでの微分は　$\nabla E = (wX - t)X^T$ となる

  - 誤差をクロスエントロピーとすると、
    $\nabla E = -\sum_{n=1}^{K}(y_n - t_n)\, X$

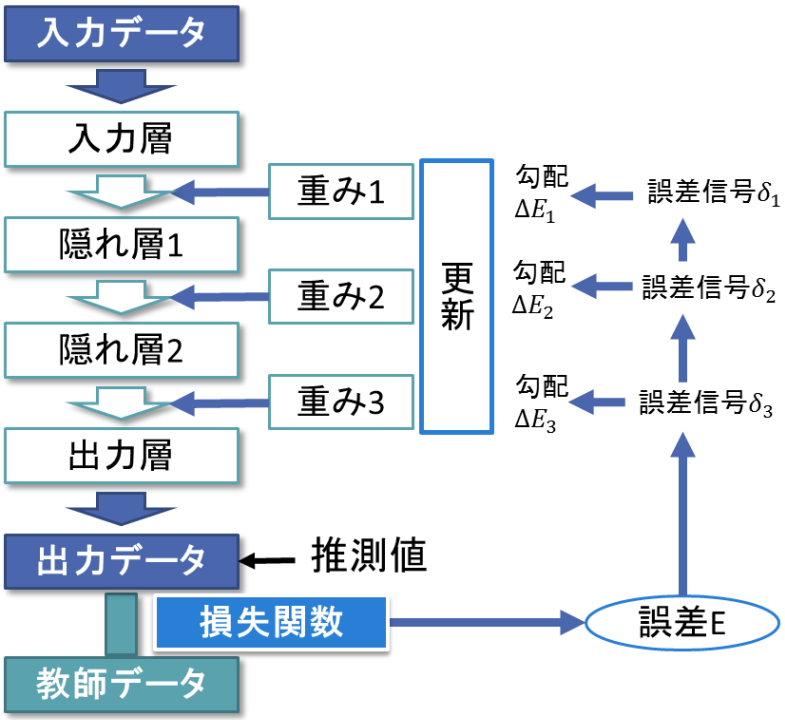
- あとで、勾配降下法をExcelで説明する

勾配降下法のイメージ

- あるウェイトwの時点での損失関数の勾配を求め、損失が減少する方向にwを更新する
- 勾配∇Eが正ならwを減少させ、負ならwを増加させる
- 学習係数 ε が大きい場合
  - 早く収束点に向かう
  - 収束点を通り過ぎることがある（収束せずに振動する）
- 学習係数 ε が小さい場合
  - 多くの学習回数が必要
  - 着実に収束点に向かう（収束しやすい）
  - 極小解で収束する場合がある（最小解ではない）
- 多次元でもあり、一概に決めることが難しい



誤差逆伝播法



∘ はアダマール積で、同じサイズの2つの行列の要素ごとの積

多層のニューラルネットワークでは誤差を伝播させて各層で重みを更新する

勾配は誤差信号δと入力から求められ、誤差信号は次のように求められる

zは隠れ層の出力

$$\delta_3 = (Y - t) \circ f_3'(Z_3)$$

f'は活性化関数の微分

$$\delta_2 = \left((W_3)^T \delta_3\right) \circ f_2'(Z_2)$$
$$\delta_1 = \left((W_2)^T \delta_2\right) \circ f_1'(Z_1)$$

$$\nabla E_3 = \delta_3 X_2^T$$
$$\nabla E_2 = \delta_2 X_1^T$$
$$\nabla E_1 = \delta_1 X_0^T$$

誤差信号が逆方向に伝播しながら勾配を求めるようになっている

勾配消失問題

- 誤差逆伝播法では、誤差が出力層側から入力層側に伝播しながら各層の重みを更新する
- 活性化関数によっては、活性化関数の微分が小さな値になり、それが層を重ねることで誤差がほぼ0になる
  - シグモイド関数の微分の最大値は0.25
  - 3層分逆伝播すると3層目は0.25^3となり、最初の誤差の約0.016倍となる
- ReLUを使うことでこの問題を軽減することができる
  - ReLUは入力が正の時の微分は常に1

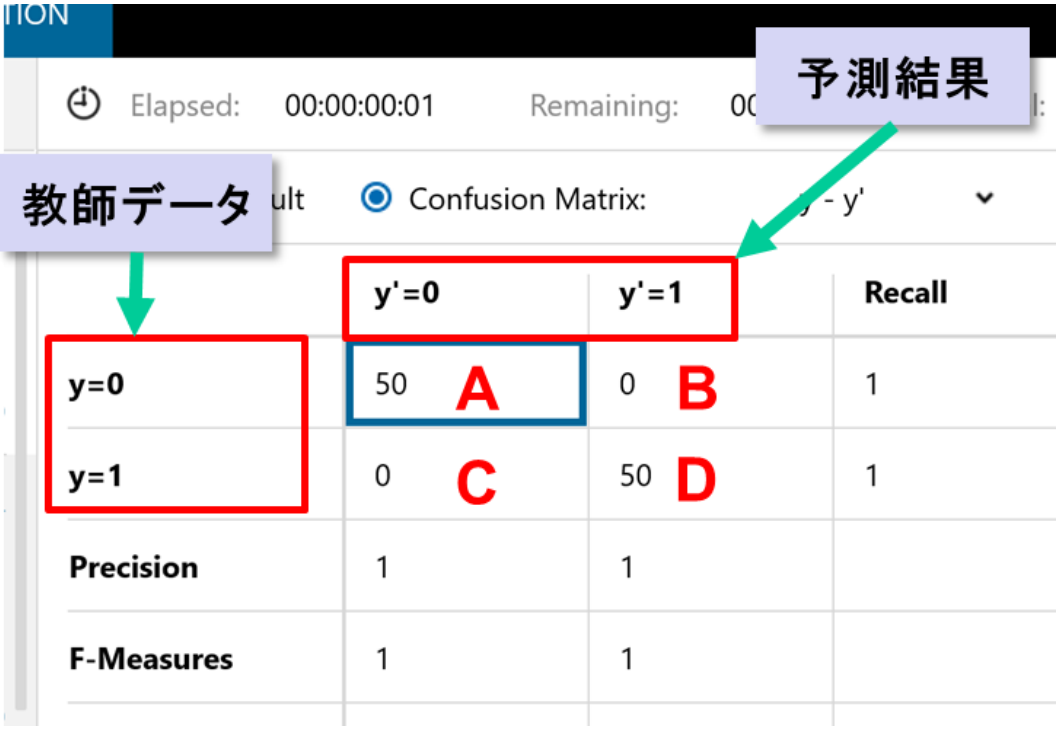
バッチ学習とミニバッチ学習

- バッチ学習
  - 用意した学習データや学習評価データを一度に使って学習を行う
- ミニバッチ学習
  - データを小分けにして学習を繰り返す
- 行列計算はデータをGPUに展開して計算を行うので、一度に計算できるデータ量はGPUのメモリに依存する
  - ミニバッチのサイズを大きくすれば学習が速くなるが、GPUのメモリ容量に応じてミニバッチのサイズを調整する必要がある
- 全サンプルの学習を1回終えることを1エポック（Epoch）と呼ぶ

過学習、過剰適合

- ディープラーニングでは、層やユニットが増えると
  パラメータ数が増え、学習データの情報を全て保持できる
  - 学習データ自体の推測はほぼ100%にできる
- しかし、学習データ以外のデータは未知のデータなので推測精度が悪くなる場合がある
  - たとえるなら、過去問は暗記して完璧だが本番が全然ダメという受験生
- 過学習を抑える方法
  - 学習評価データを使ったエポック数の設定
  - 正則化
    - ✓ 学習時のパラメータの更新幅に制約をつける
  - ドロップアウト
    - ✓ 学習時にランダムにユニットを無効化してネットワークの構造を変える

分類の評価指標



Accuracy（正解率）: $\frac{A+D}{A+B+C+D}$

Recall（再現率）: $\frac{A}{A+B}$、$\frac{D}{C+D}$

Precision（適合率）: $\frac{A}{A+C}$、$\frac{D}{B+D}$

F-Measure（F尺度）:
RecallとPrecisionの調和平均
（逆数の算術平均の逆数）

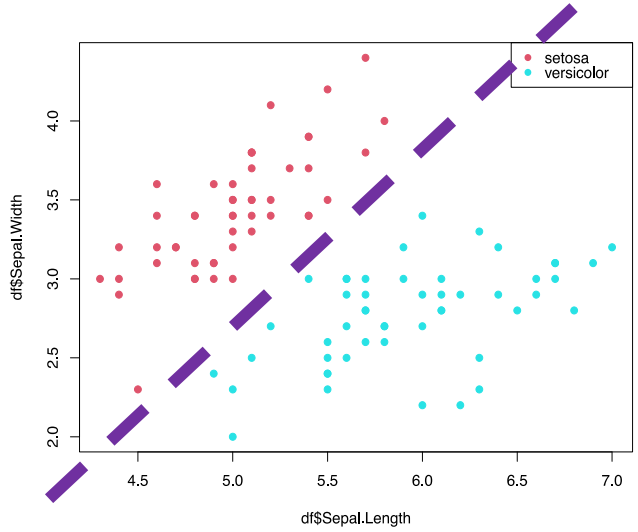
$$\frac{2 \cdot Recall \cdot Precision}{Recall + Precision}$$

損失関数と勾配降下法

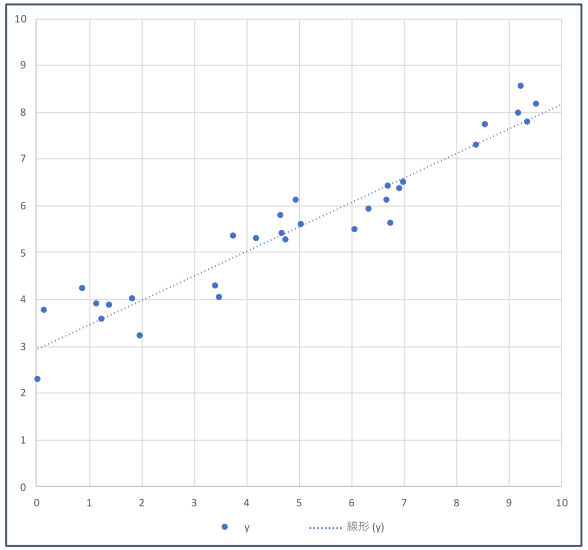
Excelによるデモ

2つのタスク：分類と回帰

- 分類
  - データをいくつかのグループに分ける
  - あるいは、個々のデータにラベルを付加する
  - 例: メールのスパム判定、画像のカテゴリ判定



- 回帰
  - 入力（説明変数）から出力（目的変数）を予測
  - 入力から出力が得られるメカニズムを関数でモデル化



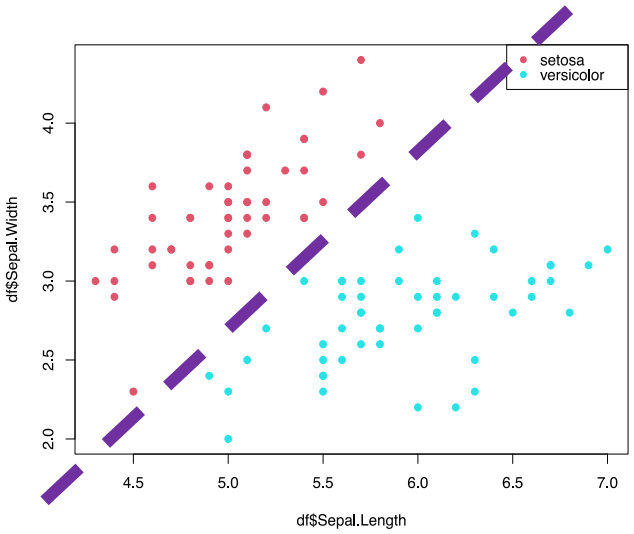
それぞれのタスクに関する手法の利用場面

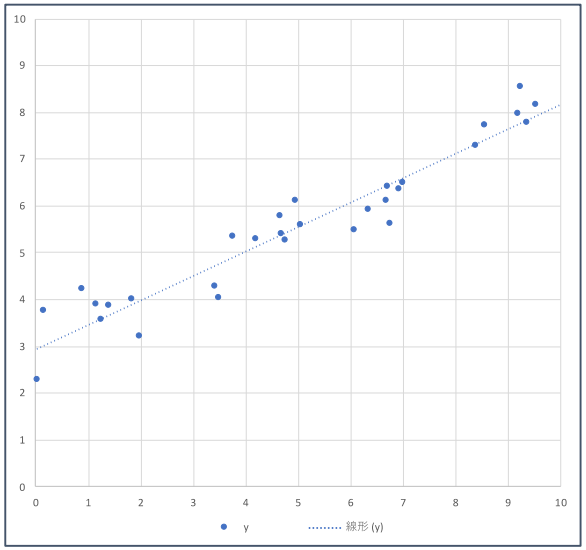
- 未知データについての推測
  - 分類タスクの場合、未知データがどのグループに属するかを推測
  - 回帰タスクの場合、未知の入力値から出力値を推測

- 既存データの説明
  - 分類の様子から知見を得る（そもそもグループがいくつあるかとか、各グループ間の関係など）
  - 変数間の関係から知見を得る

数学的視点から見た機械学習のタスク

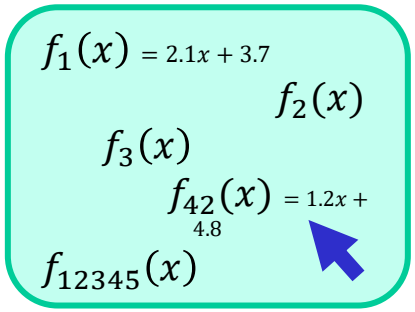
- 分類および回帰
  - 関数のあてはめ（フィッティング）
  - パラメータを持った関数族を設定し、損失関数を適切に定め（2乗誤差など）、その値が最小になるようにパラメータを決定する





損失関数と最適化

- 回帰問題の場合、当てはめに利用する関数族を設定する
  - 例： $y = ax + b$（直線の族。 a、b の組一つ一つについて、直線 $y = ax + b$ が族に含まれる）
- 関数族のパラメータを変数とする関数を適切に設定し、それを最小化することを目標とする。この関数を**損失関数**という。
  - 上の例の場合、 損失関数は $f(a, b)$ という形の関数
  - 損失関数というのはパラメータの悪さ（良さ）を測るための指標となるもの
  - 具体例は次の最小2乗法を見てください



候補となる関数族

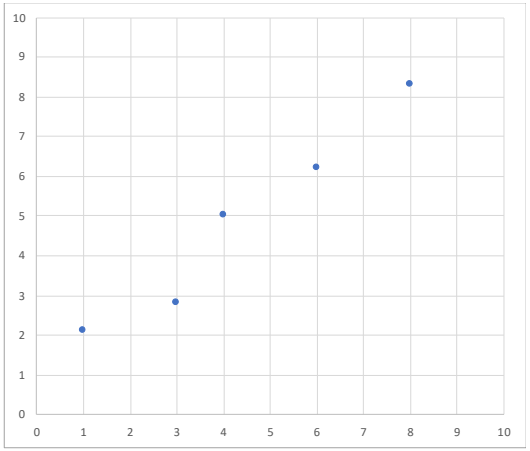
数値化（損失関数の値）

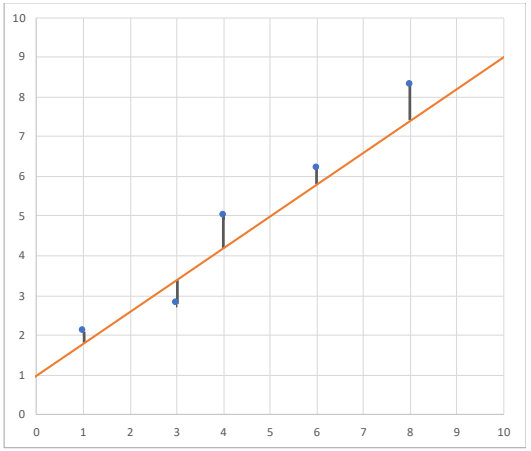
候補となる各関数を数値化し、その値の一番小さいものを選ぶ。

各関数を数値化する際に利用するのが損失関数という関数（関数の関数）

最小2乗法（あとでExcelでも説明します）

- 回帰問題の場合、損失関数として以下の2乗誤差が用いられることが多い
- 下図左のように入力と出力の組 $\{(x_i, y_i)\}_{i=1,\ldots,N}$ が与えられているとき、例えば、入力 x に対して出力 $\hat{y}$ を $\hat{y} = ax + b$ の形の関数で予測（**単回帰**）
- **2乗誤差** $E = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{2}\left(\begin{matrix}(y_i - \hat{y}_i)\text{たち}\\ \text{の2乗の和}\end{matrix}\right) = \frac{1}{2}$（下図右の縦棒の長さの2乗の和） を最小にするように a、 b を決定する

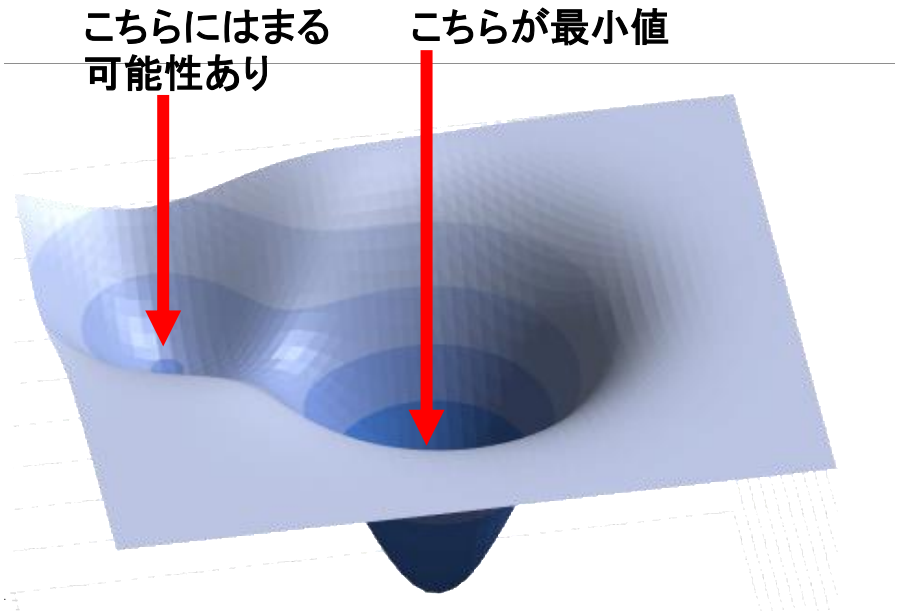




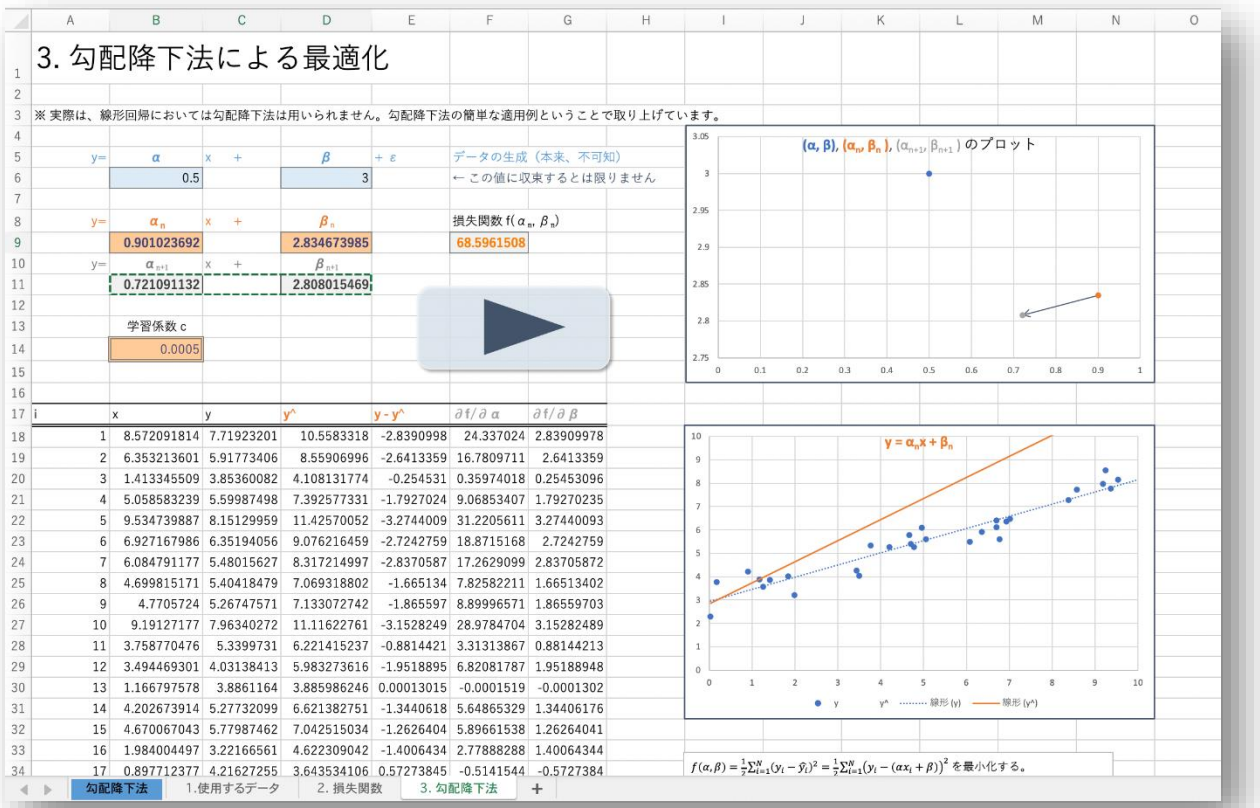
$\hat{y} = ax + b$

勾配降下法による最適化（Excelでも説明）

- 損失関数などの関数が最小値をとる点を探索する方法
- 候補点（ときにはランダムで選択）から出発し、損失関数が減少する向きへ歩を進める
- 損失関数 f が増加する向きというのは、関数の**勾配** $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$ の向きとして計算できる（あとで説明します）
- 勾配 ∇f の向きとは逆向きに少しずつ移動することを繰り返せば、最小値に行き着くだろうという（ある意味楽観的な）手法
- 深層学習でも用いられる



Excelによる勾配降下法のデモ



i	x	y	y^	y - y^	∂f/∂α	∂f/∂β
1	8.572091814	7.71923201	10.5583318	-2.8390998	24.337024	2.83909978
2	6.353213601	5.91773406	8.55906996	-2.6413359	16.7809711	2.6413359
3	1.413345509	3.85360082	4.108131774	-0.254531	0.35974018	0.25453096
4	5.058583239	5.59987498	7.392577331	-1.7927024	9.06853407	1.79270235
5	9.534739887	8.15129959	11.42570052	-3.2744009	31.2205611	3.27440093
6	6.927167986	6.35194056	9.076216459	-2.7242759	18.8715168	2.7242759
7	6.084791177	5.48015627	8.317214997	-2.8370587	17.2629099	2.83705872
8	4.699815171	5.40418479	7.069318802	-1.665134	7.82582211	1.66513402
9	4.7705724	5.26747571	7.133072742	-1.865597	8.89996571	1.86559703
10	9.19127177	7.96340272	11.11622761	-3.1528249	28.9784704	3.15282489
11	3.758770476	5.3399731	6.221415237	-0.8814421	3.31313867	0.88144213
12	3.494469301	4.03138413	5.983273616	-1.9518895	6.82081787	1.95188948
13	1.166797578	3.8861164	3.885986246	0.00013015	-0.0001519	-0.0001302
14	4.202673914	5.27732099	6.621382751	-1.3440618	5.64865329	1.34406176
15	4.670067043	5.77987462	7.042515034	-1.2626404	5.89661538	1.26264041
16	1.984004497	3.22166561	4.622309042	-1.4006434	2.77888288	1.40064344
17	0.897712377	4.21627255	3.643534106	0.57273845	-0.5141544	-0.5727384

さまざまな工夫

デモのような簡単な場合でも、単純な実装では実用にならない可能性がある。手法の改良、他の手法の採用が必要になる。

- 勾配降下法の改良
- 局所的な最小値にはまりにくくする工夫
  - 確率的勾配降下法など
- 多層の場合にシステマティックに対応
  - 単純に合成関数の勾配を求めると面倒（→誤差逆伝播法）