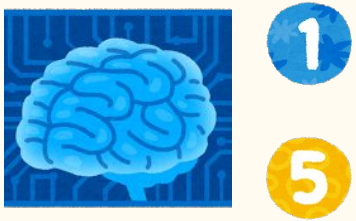
Deep Learning - その理論と実装 -

pythonで実装してみよう！

Deep Learning (DL)

1. 導入
2. 順伝播型NNと数式
3. 回帰問題への適用

Deep Learning
モデルの説明



DLによる画像分類

1. MNISTの説明
2. 分類モデルの作成

Deep Learningで
分類モデルを作成

DLの歴史

1. 3度目のNNブーム
2. 勾配消失と過学習問題
3. ReLuとドロップアウト

Deep Neural Network
の歴史と問題点

Deep Learning (DL)

Section 1

1. 導入
2. 順伝播型ニューラルネットワークと数式
3. 「住宅価格データ」への DLモデルの適用

ディープラーニングの概要

IMAGENET Deep Learningの登場

2012年に開催された大規模画像認識のコンペティション

ILSVRC (ImageNetLarge Scale Visual Recognition Challenge)

で圧勝

以降のILSVRCでは**Deep Learning**を用いた手法が主流

Deep Learningとは

深い階層構造を持った**ニューラルネットワーク**とその学習方法のこと

※ディープラーニングを知るためには、ニューラルネットワークを知る必要がある

Deep Learningとは

深い階層構造を持った**ニューラルネットワーク**とその学習方法のこと

※ディープラーニングを知るためには、ニューラルネットワークを知る必要がある

ニューラルネットワークは

脳の構造を模して作られた**多層のネットワーク**モデル

人間の脳がどのようにできているか

ご存知ですか？

人間の脳がどのようにできているか

ご存知ですか？

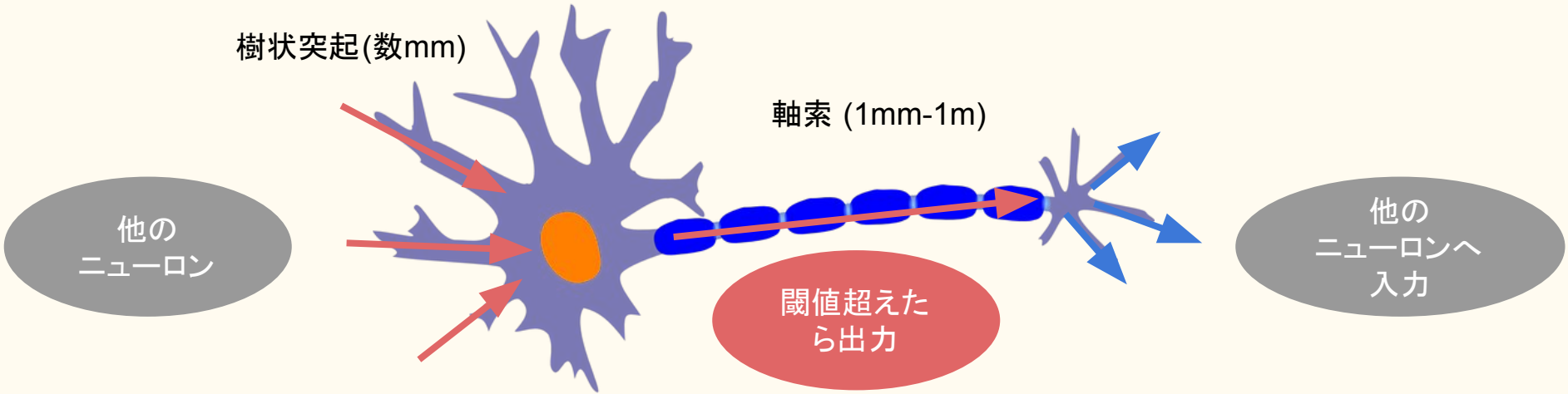
140億個の**神経細胞(ニューロン)**のネットワーク

で構成されています

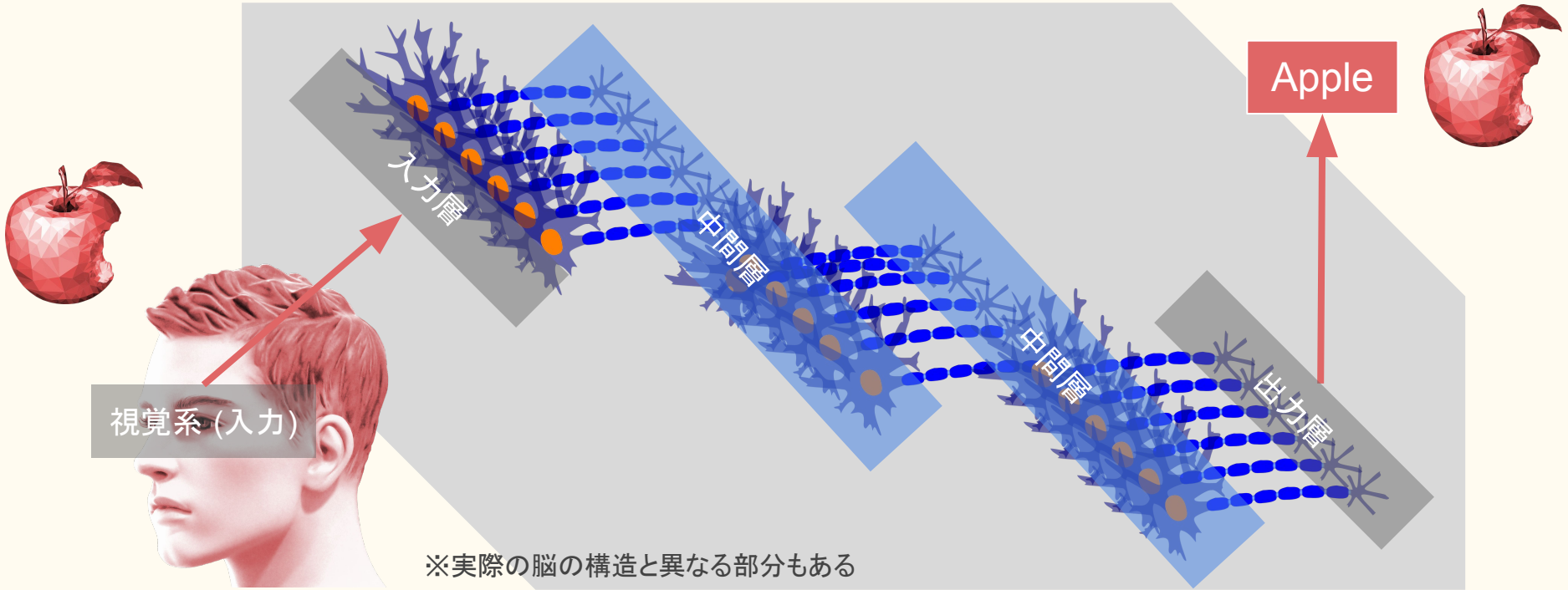
※ニューロンのネットワークだからニューラルネットワーク

神経細胞 (ニューロン)とは

- ニューロン間では電気信号による情報伝達が行われている。
- (複数の)ニューロンから入力を受け取ると自分の中で電気を計算し、**ある閾値を越えると次のニューロンにまた電気信号**を送る。



ニューロンネットワークと階層的な信号処理

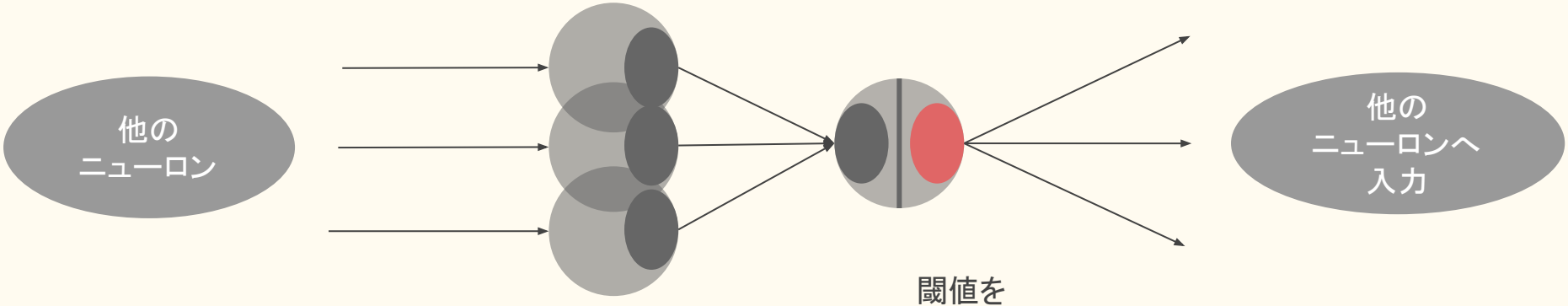


ニューロンのモデル

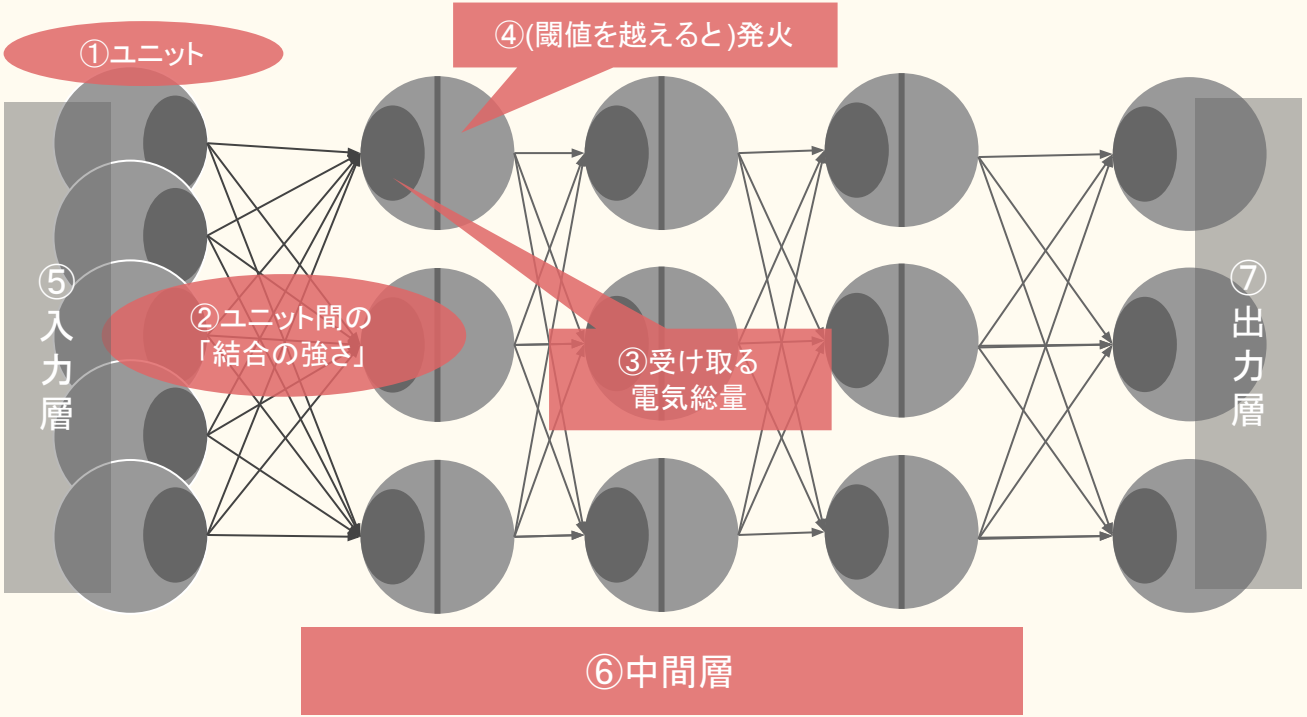
ニューロンモデルが持っている機能

①入力の仕組み　③総電気量の計算　⑤出力の仕組み

②伝達の仕組み　④発火の仕組み



ニューラルネットワーク(NN)のモデル化と登場人物の定義



・CNN系 (convolutional Neural network, 教師あり)

- 画像認識で幅広く利用。
- 畳み込みというのとプーリングを繰り返す構造

・RNN系 (Recurrent Neural Network, 教師あり)

- 時系列データ処理用。映像のデータ、言語のデータなど
- LSTM (Long-Short Term Memory)というのが主流
- MSのTay(人口チャットロボット)などの技術はLSTM

・オートエンコーダ系 (Autoencoder, 教師なし)

- 「分類」だけではなく「生成」が可能
- 変分オートエンコーダ (VAE)、生成的最適NW (GAN)というのが主流
- ピカソ風の絵を描いたり、言葉から画像を生成するのはこれ

・CNN系 (convolutional Neural network, 教師あり)

- 画像認識で幅広く利用。
- 畳み込みというのとプーリングを繰り返す構造

・RNN系 (Recurrent Neural Network, 教師あり)

- 時系
- LS
- MS

順伝播型ニューラルネットワーク

- 最も基本的かつ応用範囲が広いネットワーク -

・オートエンコーダ系 (Autoencoder, 教師なし)

- 「分類」だけではなく「生成」が可能
- 変分オートエンコーダ (VAE)、生成的最適NW (GAN)というのが主流
- ピカソ風の絵を描いたり、言葉から画像を生成するのはこれ

順伝播型NNのアウトライン

順伝播型ニューラルネットワークの特徴

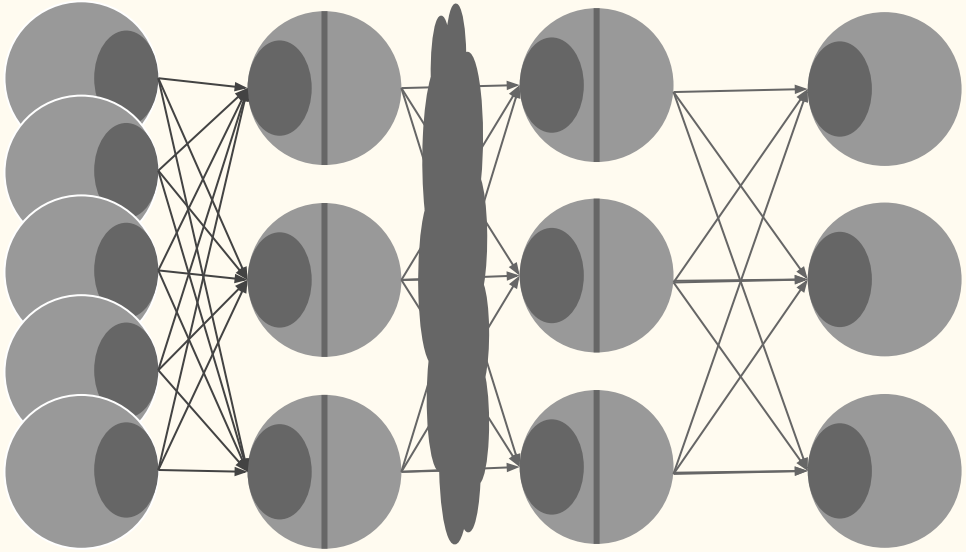
特徴

①ユニットは層状に整列
(ユニットと層の数に制限はない)

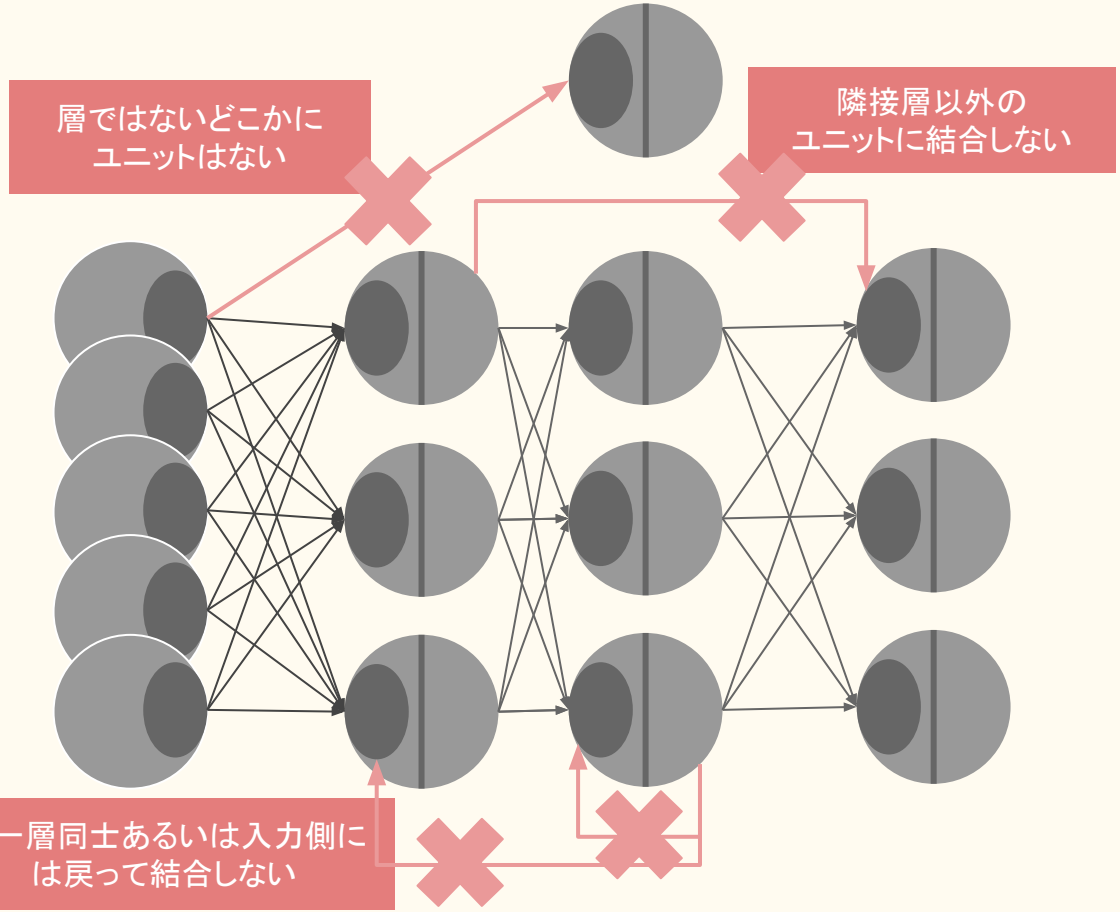
②隣接層間のみで結合

③入力側から出力側に一方向のみ
に伝播

※隠れ層が複数あるNWや逆に隠れ層を持たず入力層と出力層しかないNWもある



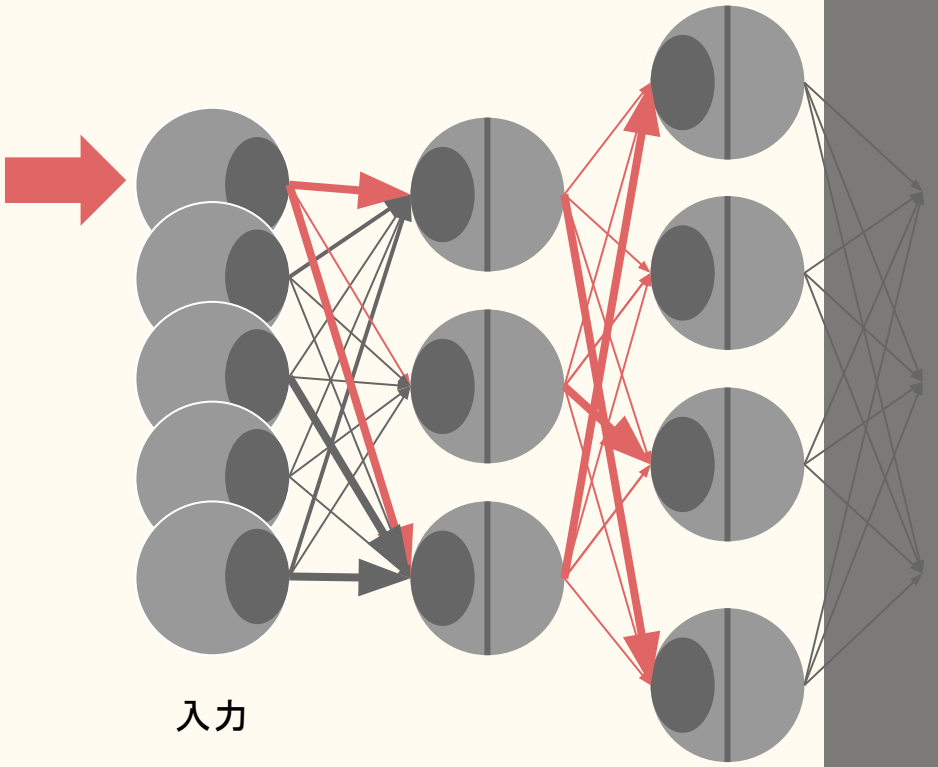
順伝播型NNで
認められていないこと



ユニット間の結合の強さ

・ユニット間の結合の強さは異なる値をもつ

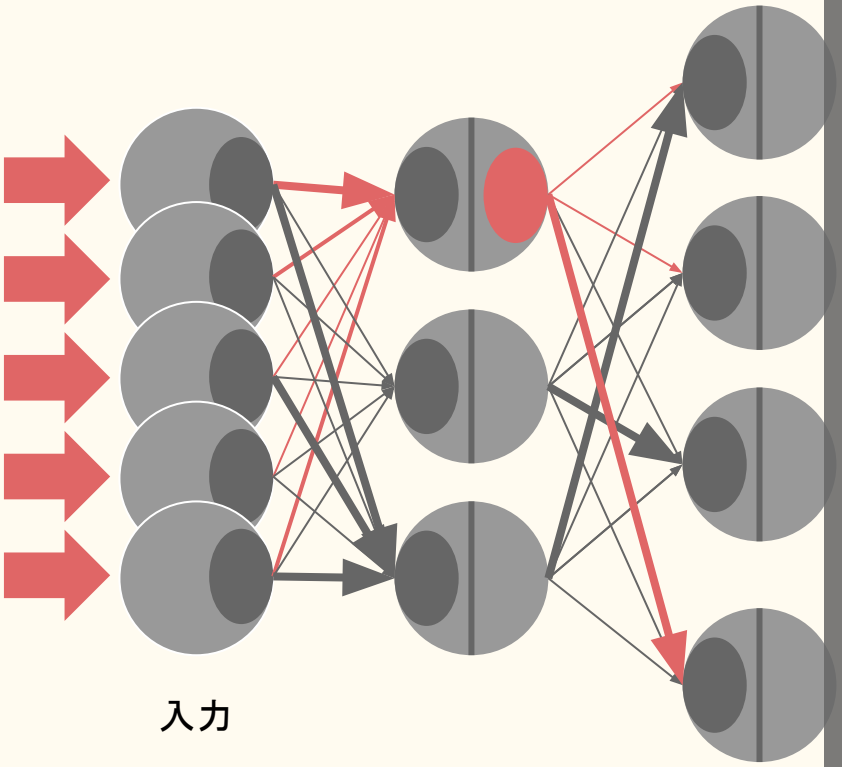
- 結合の強さはデータから学習
- あるユニットへ電気を流すと、重みに従って次層以降のユニットへ伝播する
- 結合の強さが異なるため、次層へ流れる電気量は異なる



発火の仕組み

・あるユニットでは受け取った電気の総量が閾値を超えると発火する

- 発火の仕組みには活性化関数 (後述)を使用
- それぞれのユニットで閾値は異なる
- 発火すると次の層へ電気を流す



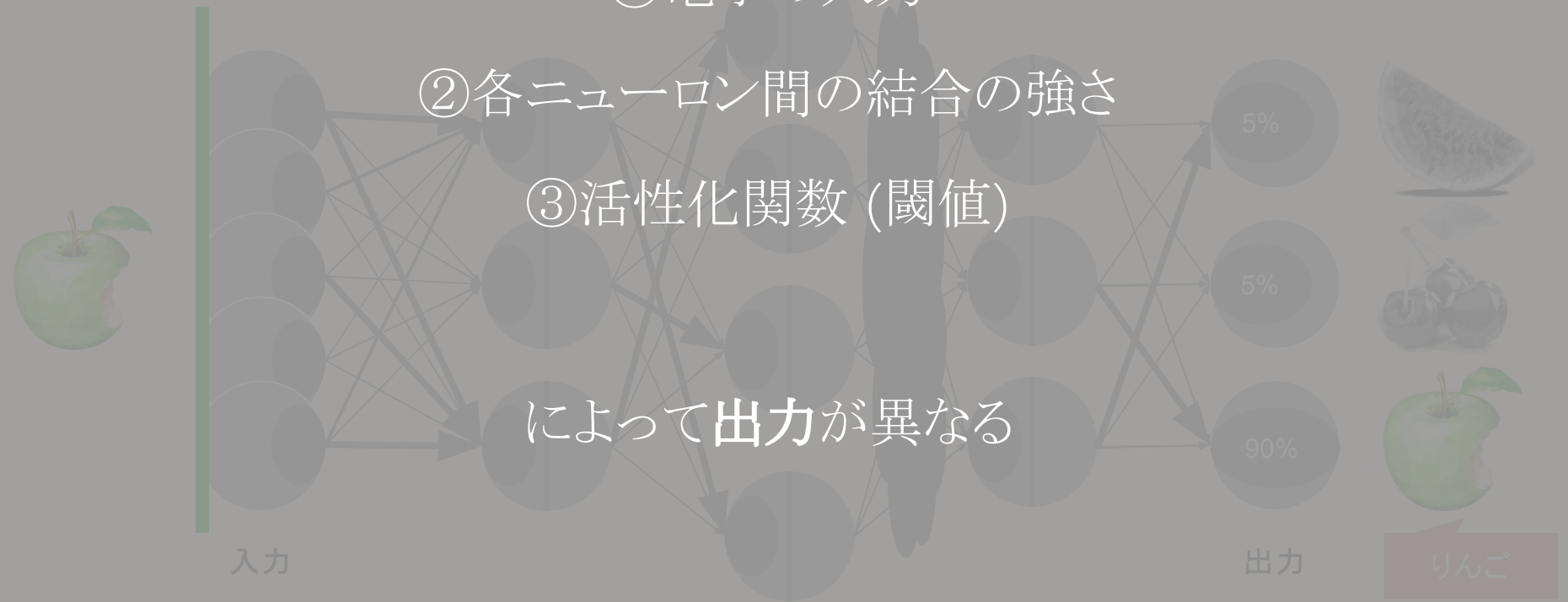
3つのフルーツを識別するモデル

①電子の入力

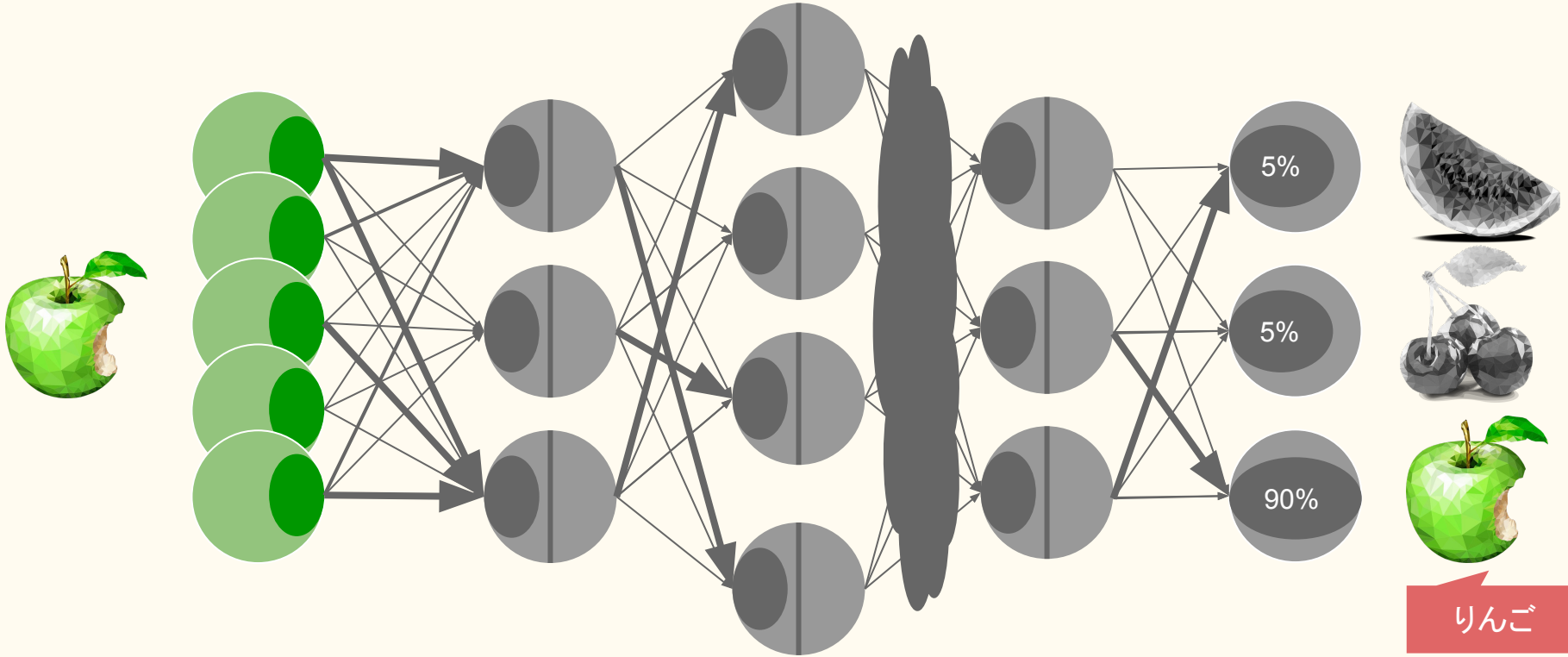
②各ニューロン間の結合の強さ

③活性化関数 (閾値)

によって**出力**が異なる



フルーツ画像の識別イメージ



順伝播型NNの数学的記述

これまで習ったモデルとの比較

線形回帰モデル

ロジスティック回帰
モデル

順伝播型NN

$$f_{reg}(\boldsymbol{x};\hat{\boldsymbol{w}}) = \hat{\boldsymbol{w}}^T\boldsymbol{x} \qquad f_{class}(\boldsymbol{x};\hat{\boldsymbol{w}}) = \frac{1}{1+\exp(\hat{\boldsymbol{w}}^T\boldsymbol{x})}$$

- n次列ベクトル
  - n個の数(スカラ)　$x_1, \cdots$を縦に並べたもの
  - スカラと区別するために太字で表現
  - i番目のスカラ値を第i成分と呼ぶ

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

第i成分

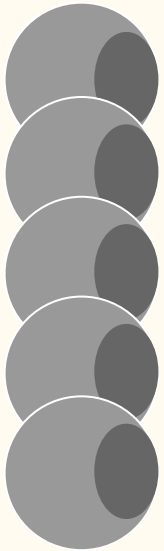
- n次行ベクトル
  - 縦ベクトルの成分を横に並べたもの
  - 内積の計算をする際によく利用される
  - 参考書の余白を節約する意味でも使われる

$$\boldsymbol{x}^T = (x_1, \cdots, x_n)$$

第i成分

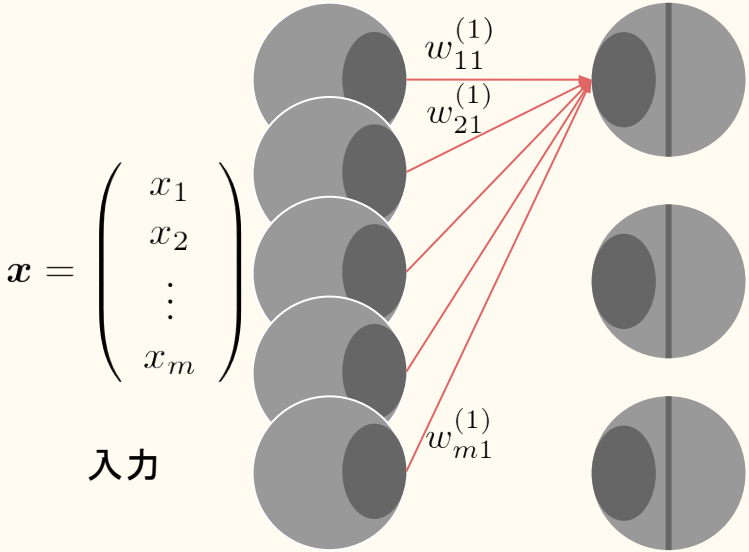
- 入力層
  - 入力(データ)をm次元列ベクトルで表記 (データ数はnで表現)

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$



入力

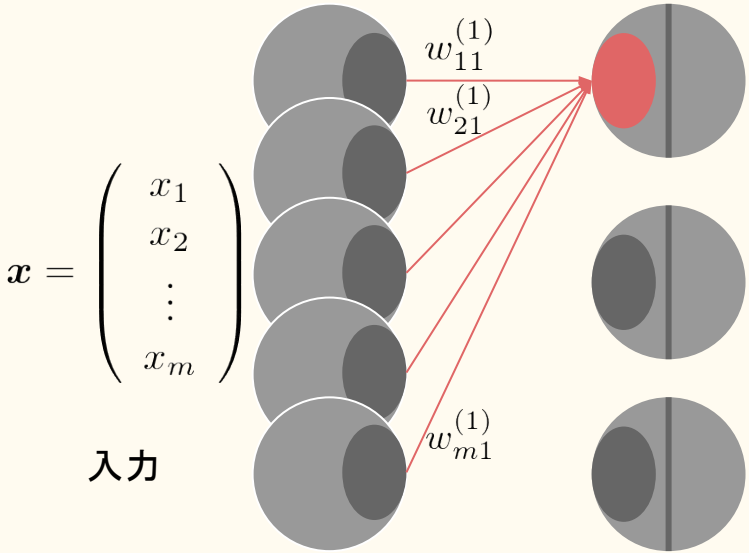
- 入力層~中間1層への結合
  - 重みの強さをm次元列ベクトルで表記
  - 入力層(1-m)成分から第(1)層の第1成分への重み



$$\boldsymbol{w}_1^{(1)} = \begin{pmatrix} w_{11}^{(1)} \\ w_{21}^{(1)} \\ \vdots \\ w_{m1}^{(1)} \end{pmatrix}$$

第1ユニットの添字

- 中間1層第1ユニットが持つ電気総量
  - 総量は結合の強さ × 入力の電気量

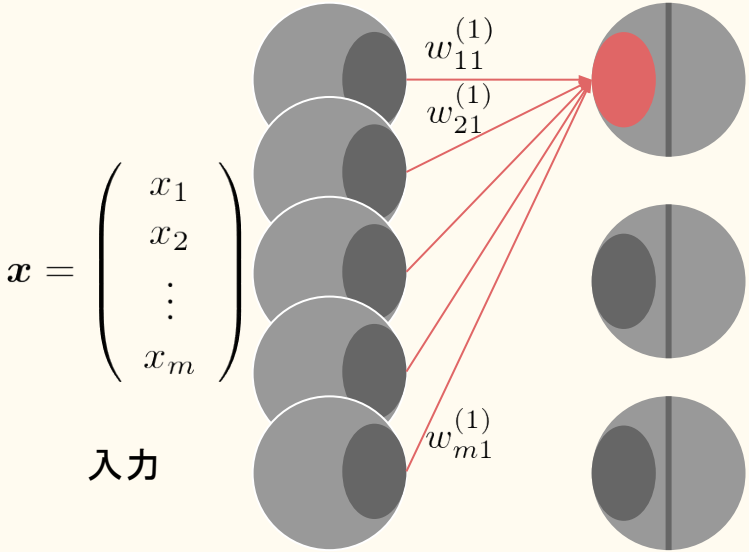


$$u_1^{(1)} = w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + \cdots + w_{m1}^{(1)} x_m$$

$$= \sum_{j=1}^{m} w_{j1}^{(1)} x_j$$

$$\boldsymbol{w}_1^{(1)} = \begin{pmatrix} w_{11}^{(1)} \\ w_{21}^{(1)} \\ \vdots \\ w_{m1}^{(1)} \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

- 中間1層第1ユニットが持つ電気総量
  - 総量は結合の強さ × 入力の電気量
  - 内積で表現できる (内積計算は交換可能であることに注意)

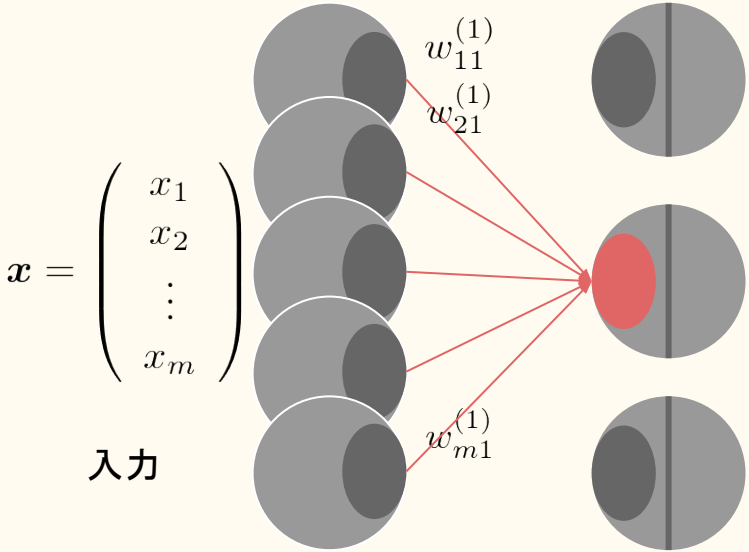


$$u_1^{(1)} = w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + \cdots + w_{m1}^{(1)} x_m$$

$$= \sum_{j=1}^{m} w_{j1}^{(1)} x_j \quad = \boldsymbol{x}^T \boldsymbol{w}_1^{(1)} = \boldsymbol{w}_1^{(1)T} \boldsymbol{x}$$

$$\boldsymbol{w}_1^{(1)} = \begin{pmatrix} w_{11}^{(1)} \\ w_{21}^{(1)} \\ \vdots \\ w_{m1}^{(1)} \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

- 中間1層第2ユニットが持つ電気総量
  - 総量は結合の強さ × 入力の電気量
  - 内積で表現できる (内積計算は交換可能であることに注意)

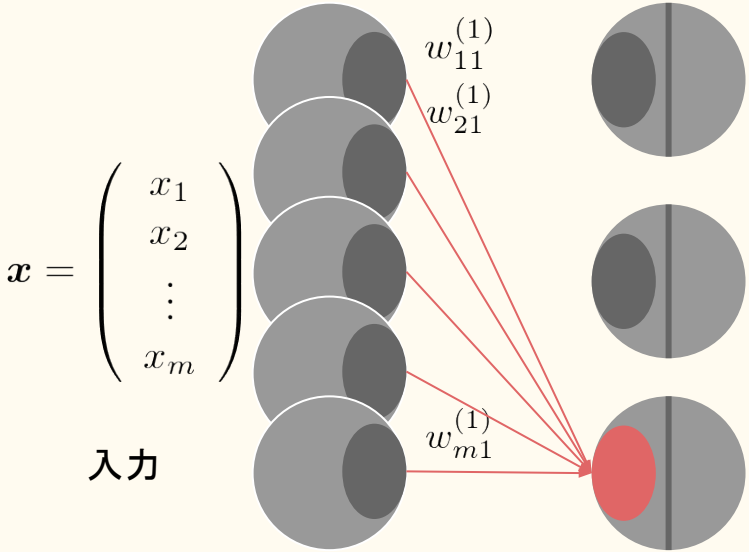


$$u_2^{(1)} = w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + \cdots + w_{m2}^{(1)} x_m$$

$$= \boldsymbol{x}^T \boldsymbol{w}_2^{(1)} = \boldsymbol{w}_2^{(1)T} \boldsymbol{x} = \sum_{j=1}^{m} w_{j2}^{(1)} x_j$$

$$\boldsymbol{w}_2^{(1)} = \begin{pmatrix} w_{12}^{(1)} \\ w_{22}^{(1)} \\ \vdots \\ w_{m2}^{(1)} \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

- 中間1層第pユニットが持つ電気総量
  - 総量は結合の強さ × 入力の電気量
  - 内積で表現できる (内積計算は交換可能であることに注意)

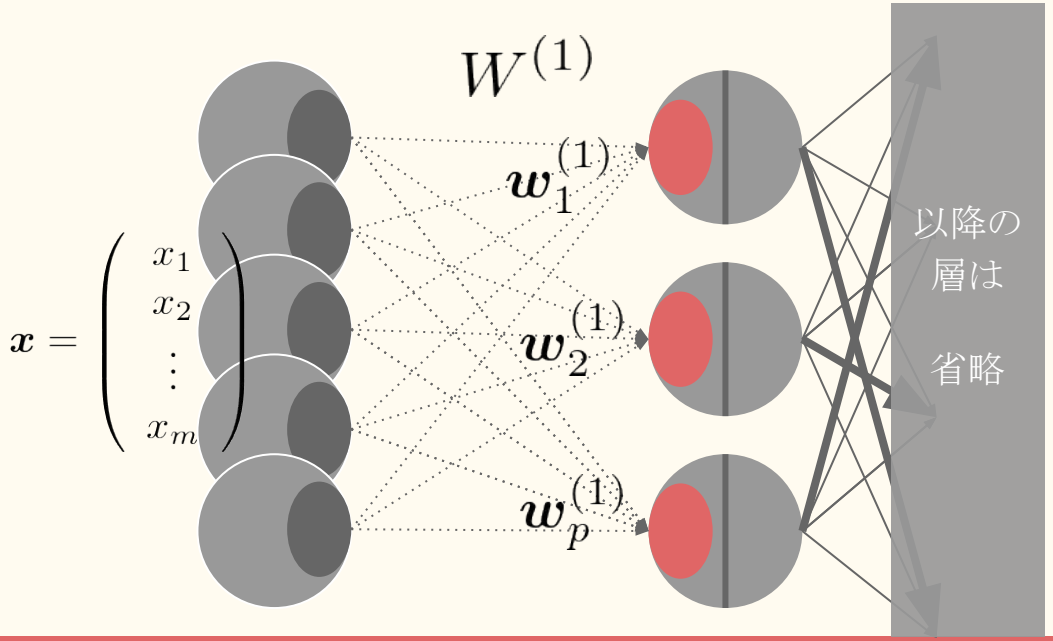


$$u_p^{(1)} = w_{1p}^{(1)} x_1 + w_{2p}^{(1)} x_2 + \cdots + w_{mp}^{(1)} x_m$$

$$= \boldsymbol{x}^T \boldsymbol{w}_p^{(1)} = \boldsymbol{w}_p^{(1)T} \boldsymbol{x} \quad = \sum_{j=1}^{m} w_{jp}^{(1)} x_j$$

$$\boldsymbol{w}_p^{(1)} = \begin{pmatrix} w_{1p}^{(1)} \\ w_{2p}^{(1)} \\ \vdots \\ w_{mp}^{(1)} \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

- 中間1層の全ユニットの持つ電気量をベクトル表記
  - 1~pの成分を持つベクトルとして表記（上付き文字は(1)）
  - 重みベクトルを並べた行列を用いて行列と入力の掛け算で表現できる



$$\boldsymbol{u}^{(1)T} = (u_1^{(1)}, u_2^{(1)}, \cdots, u_p^{(1)})$$

$$W^{(1)} = \left(\boldsymbol{w}_1^{(1)}, \boldsymbol{w}_2^{(1)}, \cdots, \boldsymbol{w}_p^{(1)}\right)^T$$

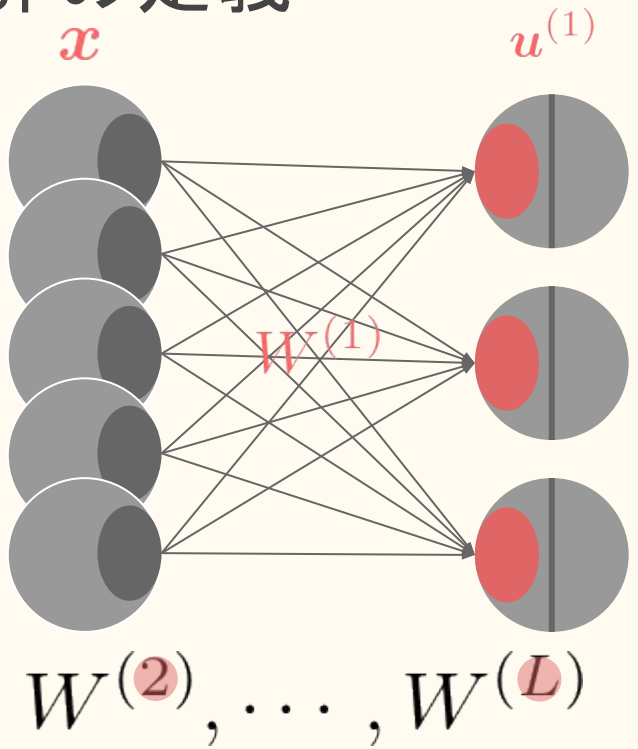
$$\boldsymbol{u}^{(1)} = W^{(1)}\boldsymbol{x}$$

ベクトルと行列の掛け算の定義

$$W\boldsymbol{x} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{p1} & w_{p2} & \cdots & w_{pm} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

$p \times m$
$m \times 1$

$$= \begin{pmatrix} \sum_{j=1}^{m} w_{1j}x_j \\ \sum_{j=1}^{m} w_{2j}x_j \\ \vdots \\ \sum_{j=1}^{m} w_{pj}x_j \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_1^T\boldsymbol{x} \\ \boldsymbol{w}_2^T\boldsymbol{x} \\ \vdots \\ \boldsymbol{w}_p^T\boldsymbol{x} \end{pmatrix}$$



第2層以降も同じ手順で記述可能

①各層の
パラメータ行列

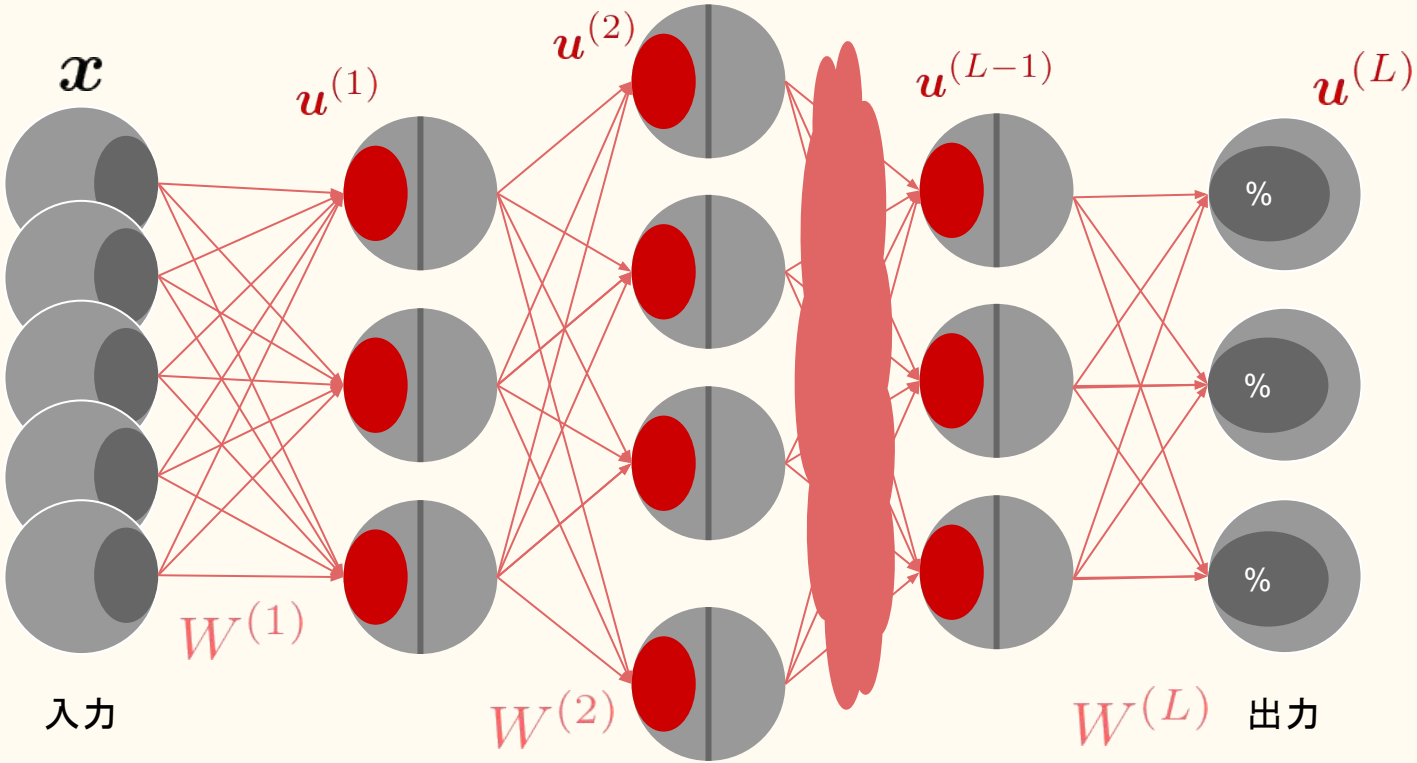
$W^{(1)}\ W^{(2)}\ \cdots\ W^{(L)}$

②入力ベクトル

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

③各層の
ユニットベクトル

$\boldsymbol{u}^{(1)}\ \boldsymbol{u}^{(2)}\ \cdots\ \boldsymbol{u}^{(L-1)}$

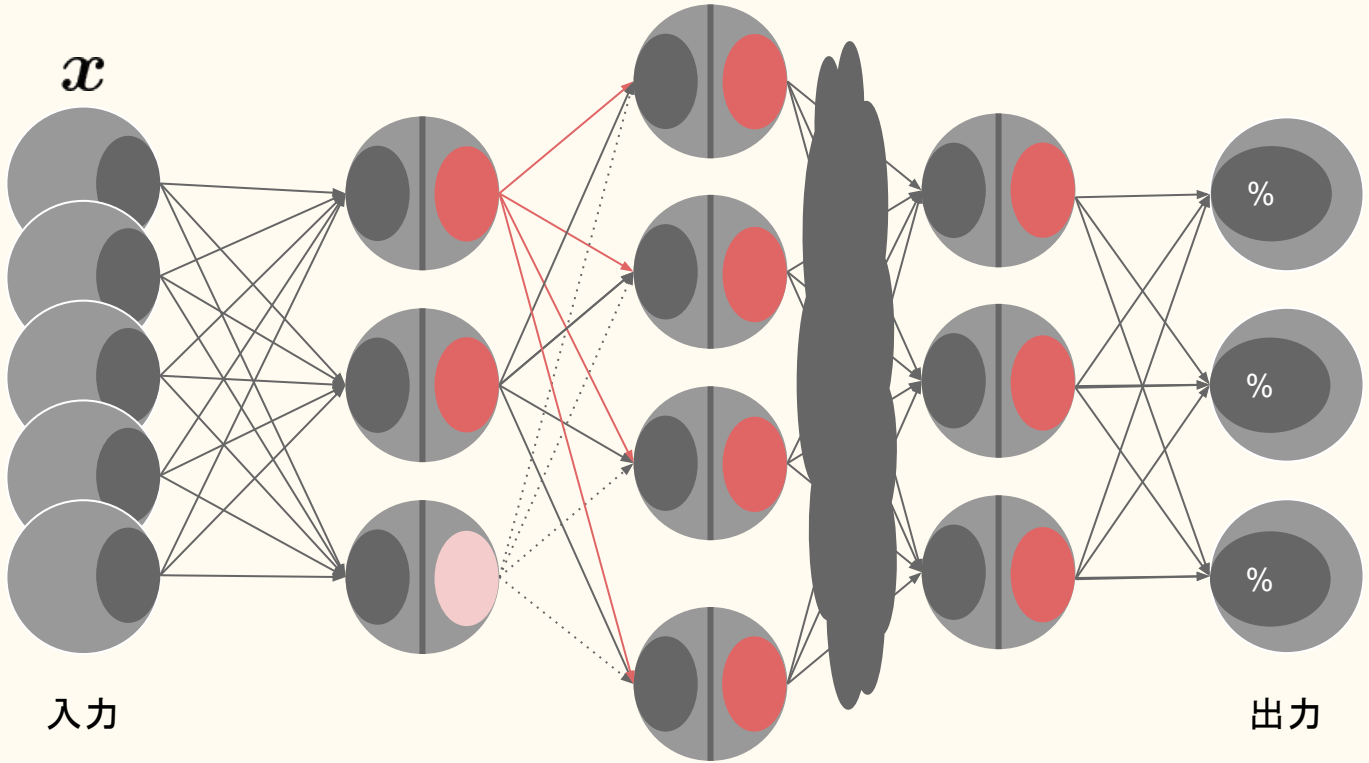


表記できていないのは隠れ層の発火

発火の仕組み

閾値を越えると
次の層へ電気を伝播
(電気量も考慮)

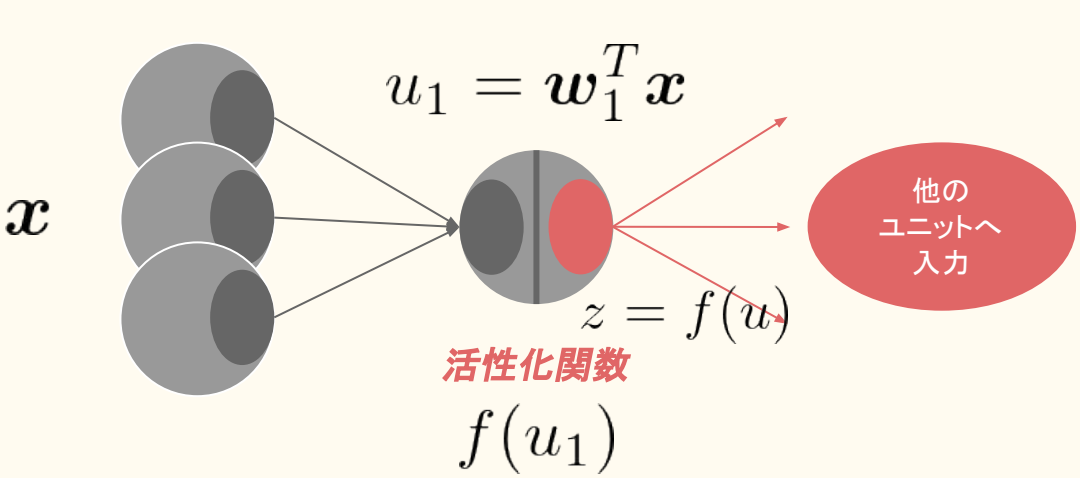
超えなければ
次の層へは伝播しない

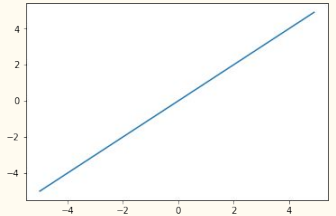


$$\underset{1\times m\ m\times n}{\boldsymbol{w}^T\boldsymbol{X}} = (w_1, w_2, \cdots, w_m)\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix}$$

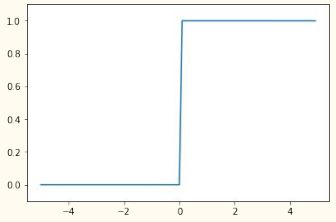
$$= \left(\sum_{j=1}^{m} x_{j1}w_j, \sum_{j=1}^{m} x_{j2}w_j, \cdots, \sum_{j=1}^{m} x_{jn}w_j\right) = \underset{1\times n}{(\boldsymbol{w}^T\boldsymbol{x}_1, \boldsymbol{w}^T\boldsymbol{x}_2, \cdots \boldsymbol{w}^T\boldsymbol{x}_n)}$$

- 活性化関数を利用
  - ユニットが持つ活性化関数には通常、単調増加する非線形関数を利用する場合が多い
  - あるユニットが受ける電気量は既に計算済み
  - 入力は電気の総量で出力は利用する活性化関数によって異なる （次スライドで説明）



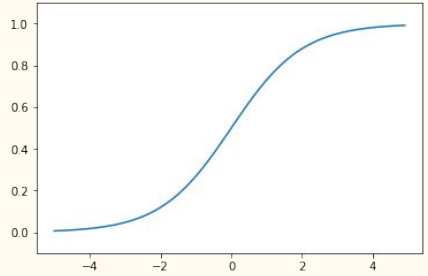


恒等関数

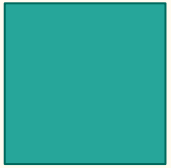


ステップ関数

- 古くからよく使われる関数
  - ロジスティックシグモイド関数（logistic sigmoid function）/ 双曲線正接関数
    - これらの関数は入力の絶対値大きな値をとると出力が飽和し一定値となる
    - その間の入力に対して出力が徐々にかつ滑らかに変化する
  - 生物の神経細胞が持つ性質をモデル化したもの

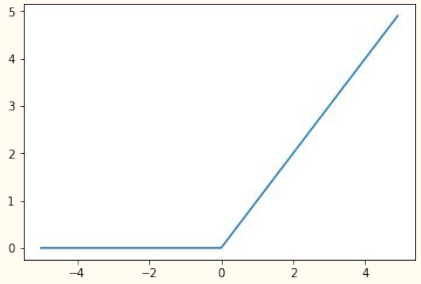


シグモイド
関数



- 近年これらの活性化関数に代わり**正規化線形関数**がよく使われる
  - **正規化線形関数** (**R**ectified **l**inear **u**nit function)
    - 入力が0より小さければ電気を伝播しない
    - 入力が0より大きければ電気をそのまま伝播する
  - 最終的により良い結果が得られることが多いため、最もよく使われる
    - 単純で計算量が小さい
    - シグモイド関数・双曲正接関数よりも学習がより早く進む
  - シグモイド関数では入力が小さすぎる (大きすぎる)と出力がほとんどの場合、0(1)になってしう

$$f(u) = \max(0, u)$$



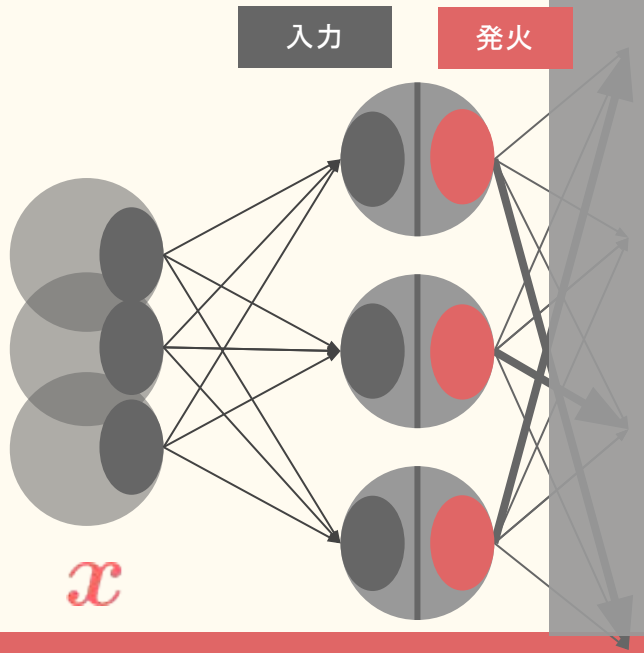
- 第1層のユニットの活性関数による出力ベクトル表現
  - 単一の活性化関数fを使うかのように表記したが各層で異なる物を選んでも構わない
  - 特に出力層のユニットの活性化関数は問題に応じて一般に中間層とは異なる物を選ぶ

入力

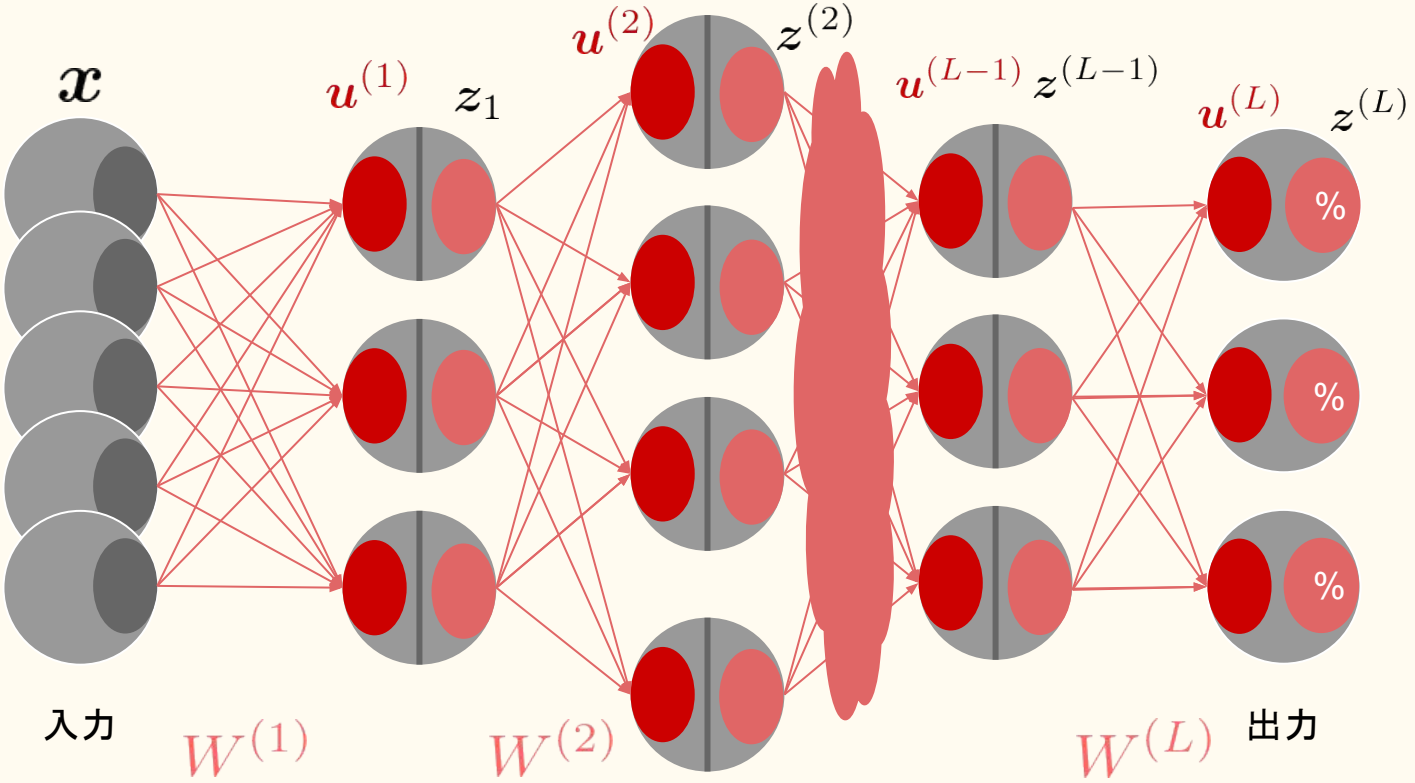
$$\boldsymbol{u}^{(1)} = \begin{pmatrix} u_1^{(1)} \\ u_2^{(1)} \\ \vdots \\ u_p^{(1)} \end{pmatrix}$$

発火

$$\boldsymbol{z}^{(1)} = \boldsymbol{f}^{(1)}(\boldsymbol{u}^{(1)}) = \begin{pmatrix} f(u_1^{(1)}) \\ f(u_2^{(1)}) \\ \vdots \\ f(u_p^{(1)}) \end{pmatrix}$$



NNを全体的に記述できた

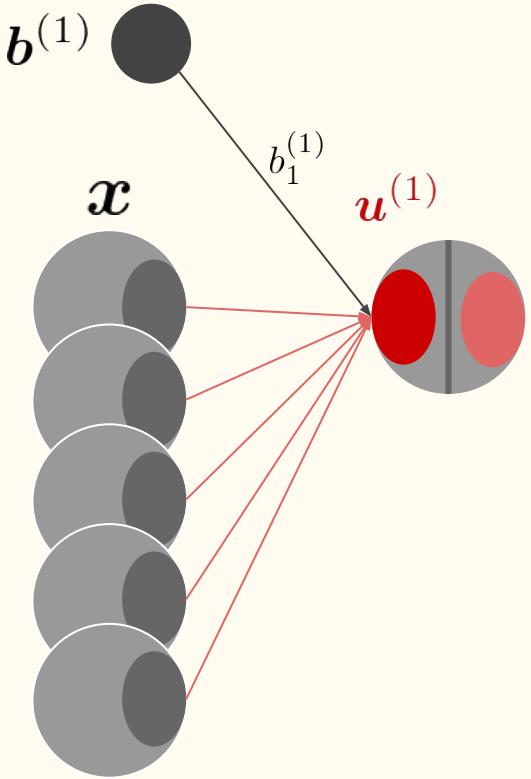


各層のユニットへバイアス項を付与

中間層にはバイアスと呼ばれる
「あるユニットが発火する傾向の高さ」を表す量を付与する

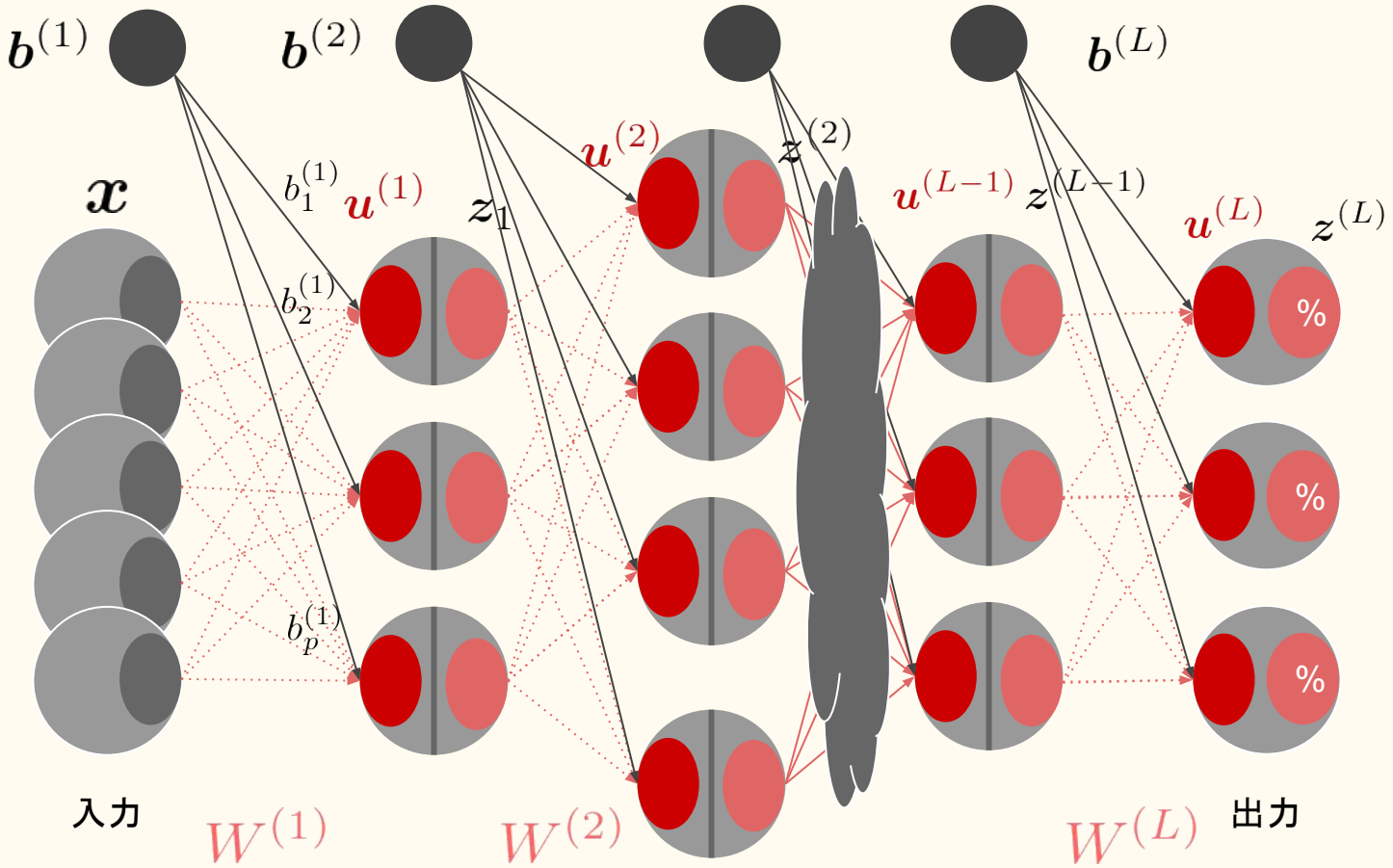
$$\boldsymbol{b}^{(1)} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_p^{(1)} \end{pmatrix}$$

※簡略化のためこれまで説明していませんでした

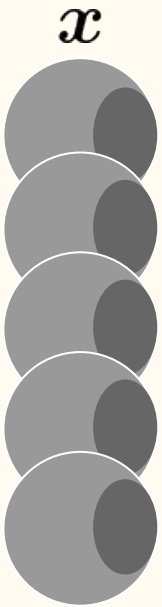


※「あるユニットが発火する傾向の高さ」を表す量

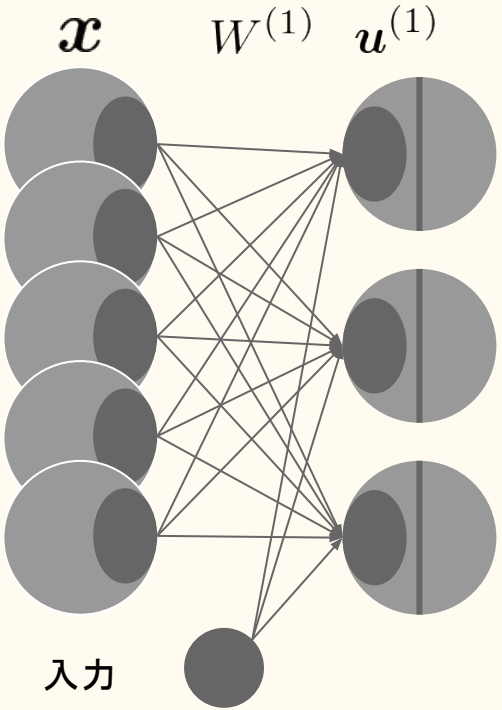
- バイアスが高ければ、受け取る電気総量が少なくても発火する
- 反対にバイアスがマイナスの値であれば、受け取るでき総量が大きくても発火しない
- バイアスの値も学習で決定する


$\boldsymbol{b}^{(1)}$
$\boldsymbol{b}^{(2)}$
$\boldsymbol{b}^{(L)}$
$\boldsymbol{x}$
$b_1^{(1)}$
$b_2^{(1)}$
$b_p^{(1)}$
$\boldsymbol{u}^{(1)}$
$\boldsymbol{z}_1$
$\boldsymbol{u}^{(2)}$
$\boldsymbol{z}^{(2)}$
$\boldsymbol{u}^{(L-1)}$
$\boldsymbol{z}^{(L-1)}$
$\boldsymbol{u}^{(L)}$
$\boldsymbol{z}^{(L)}$
%
%
%
入力
$W^{(1)}$
$W^{(2)}$
$W^{(L)}$
出力

第1層から第L層までの流れを確認



第1層から第L層までの流れを確認

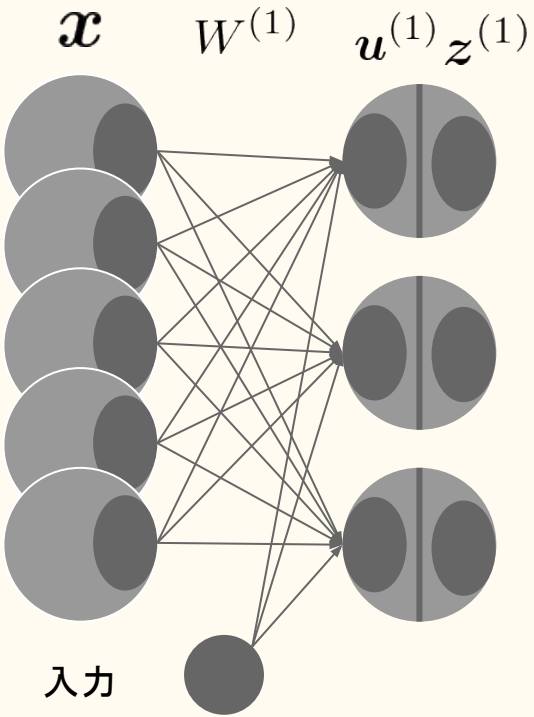


入力層から第1層への伝播

$$\underset{\boldsymbol{u}^{(1)}}{W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}}$$

各ユニットが受け取る
電気の総量

第1層から第L層までの流れを確認

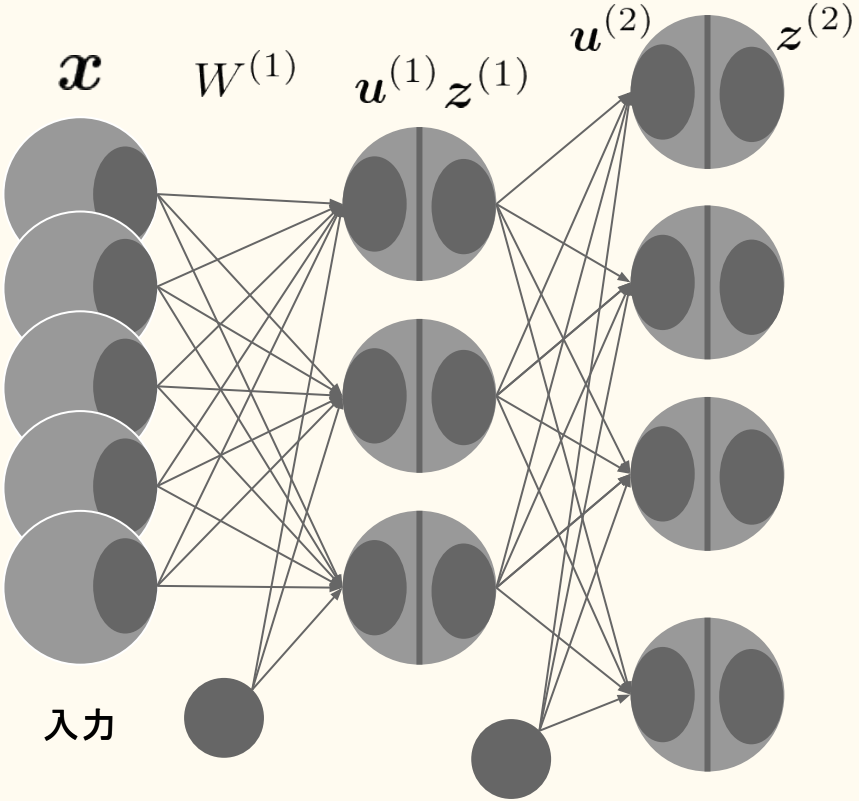


第1層の活性化

$$\boldsymbol{z}^{(1)} = \boldsymbol{f}^{(1)}(\underbrace{W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}}_{\boldsymbol{u}^{(1)}})$$

各ユニットが受け取る
電気の総量から
発火する電気

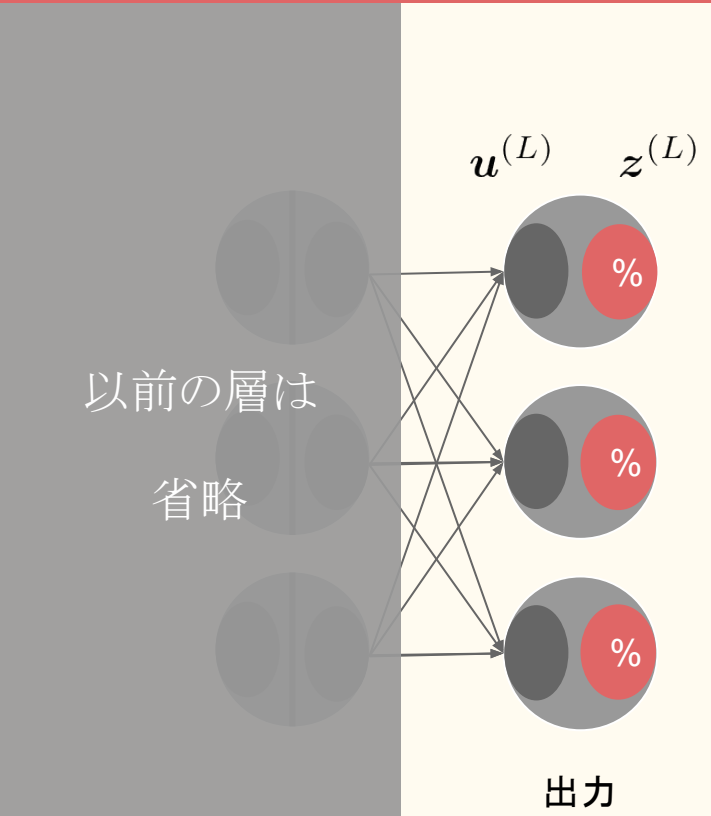
第1層から第L層までの流れを確認



第2層の活性化関数まで

$$\boldsymbol{z}^{(2)} = \boldsymbol{f}^{(2)}(W^{(2)}\underbrace{\boldsymbol{f}^{(1)}(W^{(1)}\boldsymbol{x}+\boldsymbol{b}^{(1)})}_{\boldsymbol{z}^{(1)}}+\boldsymbol{b}^{(2)})$$

各ユニットが受け取る
電気の総量から
発火する電気

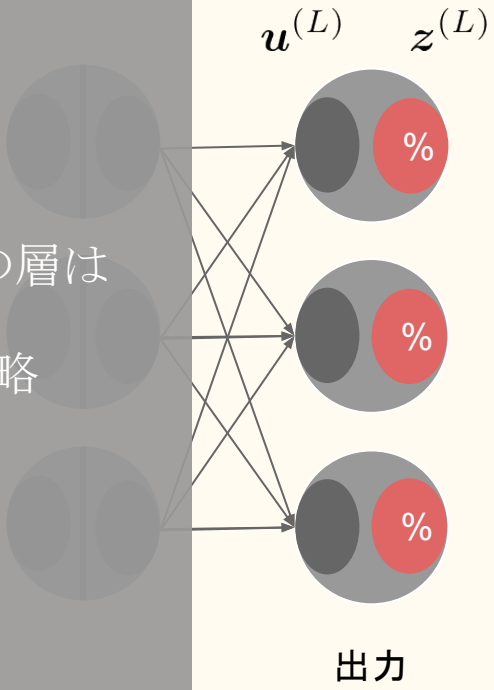


第1層から出力層の前まで

$$z^{(L)} = f^{(L)}(W^{(L)}z^{(L-1)} + b^{(L)})$$

以前の層は

省略



第1層から出力層まで

$$z^{(L)} = f^{(L)}(W^{(L)} \cdot z^{(L-1)} + b_L)$$

出力層の各ユニットには確率が
出力されている・・・

$$f^{(L)}$$

はどのような形になっている？

- 他クラス分類の場合、出力層にはsoftmax関数を利用
  - softmax関数を用いる事でベクトルの成分が正規化され、「出力確率」としてあつかえるようになるので、DNNのモデルと非常に相性が良い。
  - 線形ユニットを出力とする場合 (回帰)は「線形回帰モデル」にて説明済
  - ロジスティックシグモイド関数を出力とした場合 (2値分類)は「ロジスティック回帰モデル」似て説明済

$$softmax(\boldsymbol{u}^{(L)}) = \frac{1}{\sum_{j=1}^{p} \exp(u_1^{(L)})} \begin{pmatrix} \exp(u_1^{(L)}) \\ \exp(u_2^{(L)}) \\ \vdots \\ \exp(u_p^{(L)}) \end{pmatrix}$$

全て足すと1になる
(確率になっている)

これまで習ったモデルとの比較

線形回帰モデル

ロジスティック回帰
モデル

順伝播型NN

$$\boldsymbol{z}^{(L)} = \boldsymbol{f}^{(L)}(W^{(L)}\boldsymbol{z}^{(L-1)} + \boldsymbol{b}^{(L)})$$

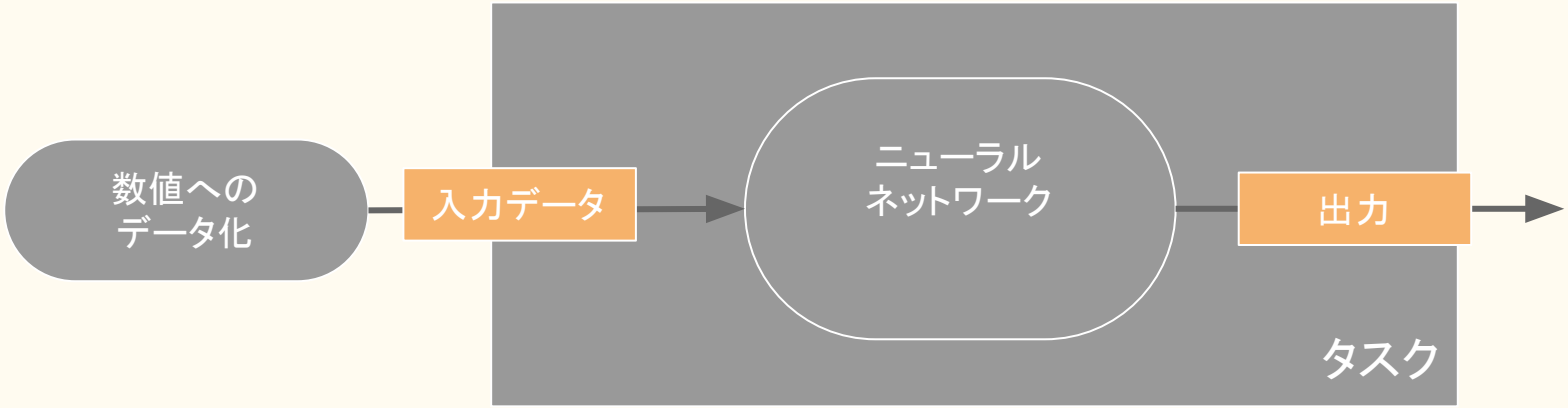
$$f_{reg}(\boldsymbol{x}; \hat{\boldsymbol{w}}) = \hat{\boldsymbol{w}}^T \boldsymbol{x} \qquad f_{class}(\boldsymbol{x}; \hat{\boldsymbol{w}}) = \frac{1}{1 + \exp(\hat{\boldsymbol{w}}^T \boldsymbol{x})}$$

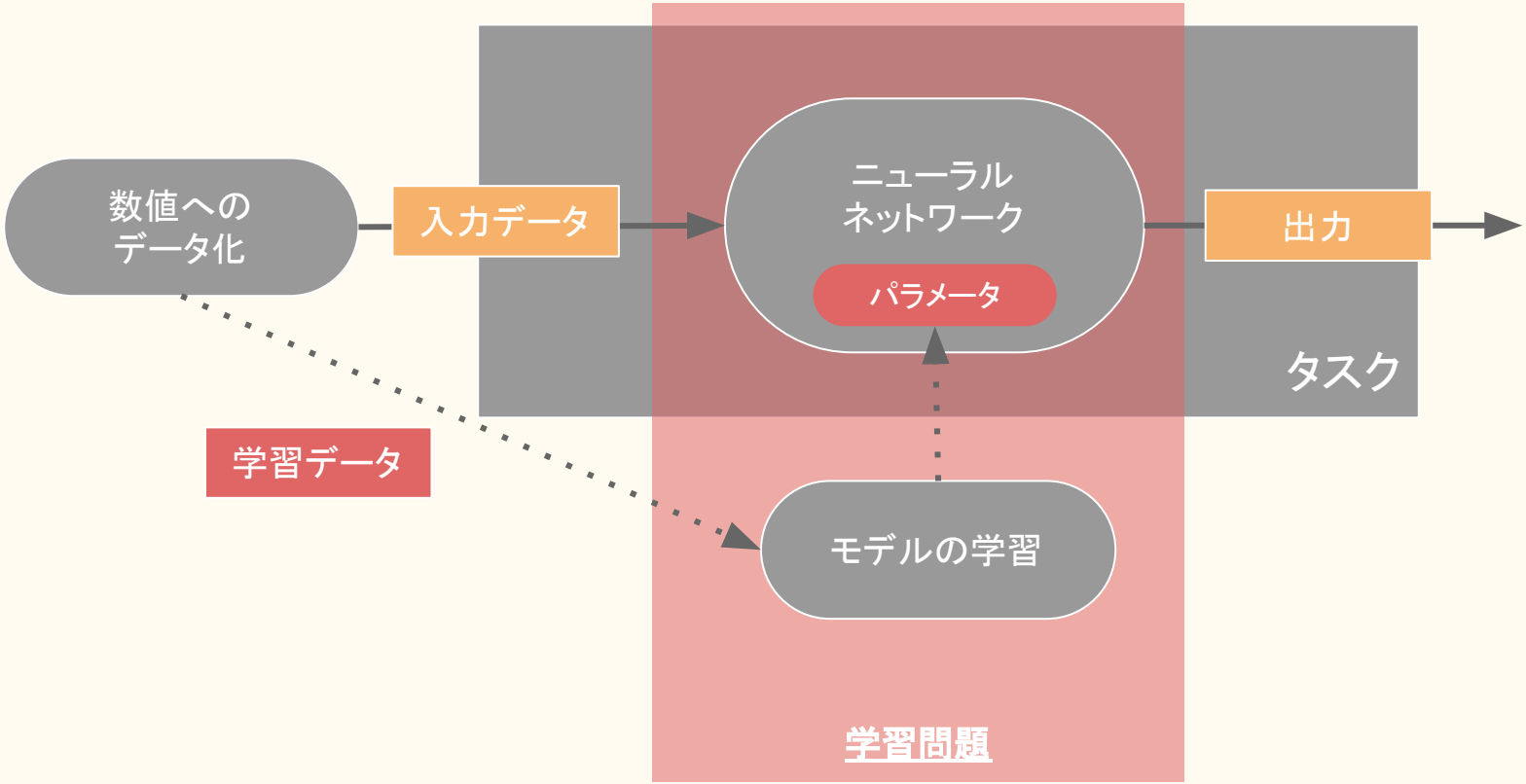
- 順伝播型NNでは与えられた入力xに対し、入力層から出力層へ上の計算を繰り返すことで情報を伝播させ、出力yを計算する
  - この関係は以下で表現可能

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x})$$

  - この関数の中身を決定するのは各層間の結合重みとユニットのバイアス　(ネットワークパラメータと呼ぶ)
  - パラメータを変化させれば、様々な関数を表現できる

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}; W^{(1)}, \cdots, W^{(L-1)}, \boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(L-1)})$$


数値への
データ化
入力データ
ニューラル
ネットワーク
出力
タスク


数値への
データ化
入力データ
ニューラル
ネットワーク
パラメータ
出力
タスク
学習データ
モデルの学習
学習問題

- 入力層、中間層、出力層、活性化関数や層の数は問題に応じて適切に設定
- 万能近似定理
  - 中間層の隠れニューロンの数を増やせば、(理論上は)任意の連続関数を近似できてしまう。
  - 参考URL http://www.yukisako.xyz/entry/neural-network

	層数	ユニット数	活性化関数
入力層	1	データ数	無し
中間層	複数	複数	Reluが主流
出力層	1	**別表**	

別表	ユニット数	活性化関数
回帰	1	恒等関数
2クラス分類	2	シグモイド関数
多クラス分類	カテゴリ数	ソフトマックス関数

学習の枠組み

- 誤差逆伝播 (アウトラインのみ)-

- 順伝播型NWが表現する関数はネットワークのパラメータを変えると変化する。よいパラメータを選ぶことでこのネットワークが望みの関数を与えるようにすることを考える

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}; W^{(1)}, \cdots, W^{(L-1)}, \boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(L-1)})$$

- 目標とする関数は、姿形を分かっていないものの、関数の入力と出力のペアが引く数与えられているとする。
  - 一つの入力xに対する望ましい出力をdとする。これらのペア1つ1つを学習データと呼ぶ。

$$\{(\boldsymbol{x}_1, \boldsymbol{d}_1), (\boldsymbol{x}_2, \boldsymbol{d}_2) \cdots (\boldsymbol{x}_n, \boldsymbol{d}_n)\}$$

- パラメータ推定
  - 順伝播NNでは学習データの入出力のペアをなるべく再現するようパラメータを推定　(学習)
  - 入力を与えた時のDNNの出力がなるべくラベル集合に近づくように調整
  - 推定の具体的な方法な次週以降から。本日はアウトラインのみの説明
- **NWが表現する関数と学習データの近さをどのように測るか、つまりそれらの近さの尺度が必要。この尺度のことを誤差関数と呼ぶ。**

$$\{(\boldsymbol{x}_1, \boldsymbol{d}_1), (\boldsymbol{x}_2, \boldsymbol{d}_2) \cdots (\boldsymbol{x}_n, \boldsymbol{d}_n)\} \longrightarrow \boxed{\text{DNN}} \longrightarrow D = \{\boldsymbol{d}_1, \boldsymbol{d}_2 \cdots \boldsymbol{d}_n\}$$

近さの尺度 = 誤差関数

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}; W^{(1)}, \cdots, W^{(L-1)}, \boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(L-1)})$$

- 順伝播型NNでは主に回帰やクラス分類の問題を扱うが問題の種別に応じて誤差関数およびネットワークの出力層の設計が変わる。
  - 表に問題の種別ごとの活性化関数と誤差関数を一覧としてまとめている

問題の種別	出力層の活性化関数	誤差関数
回帰	恒等写像	2乗誤差
2値分類	ロジスティック関数	交差エントロピー (尤度関数)
多クラス分類	ソフトマックス関数	交差エントロピー (尤度関数)

- 回帰の誤差関数
  - 出力層の活性化関数を選んだ上で NWの出力が訓練データの目標出力に可能な限り近くなるように学習する
    - 目的変数の値域が [-1, 1]の場合は、出力層の活性化関数には双曲線正接関数
    - 目的変数の値域が任意の実数の場合は、恒等写像
  - 誤差関数としては2乗誤差を利用 (「線形回帰」の講義で既に説明済み)

$$E(W^{(1)}, \cdots, W^{(L-1)}, \boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(L-1)})$$

$$= \frac{1}{2} \sum_{i=1}^{n} \{\boldsymbol{d}_n - \boldsymbol{y}(\boldsymbol{x}; W^{(1)}, \cdots, W^{(L-1)}, \boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(L-1)})\}^2$$

- 2値分類の誤差関数
  - 出力層にロジスティックシグモイド関数を利用
  - 誤差関数として尤度関数を利用 (「ロジスティック回帰モデル」で既に説明済み)
    - 交差エントロピーと呼ぶ

$$E(W^{(1)}, \cdots, W^{(L-1)}, \boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(L-1)})$$

$$= -\sum_{i=1}^{n} [d_n \log y(\boldsymbol{x}_i; W^{(1)}, \cdots, W^{(L-1)}, \boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(L-1)})$$

$$+(1-d_n) \log\{1 - y(\boldsymbol{x}_i; W^{(1)}, \cdots, W^{(L-1)}, \boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(L-1)})\}]$$

- 多クラス分類
  - 出力層にはSoftmax関数を利用
  - 誤差関数には尤度関数を利用 (2値分類の時と同様)
  - データを1 of K表現して尤度関数を構成
    - クラスが真のクラスで会った時のみ 1をとり、それ以外は0とする

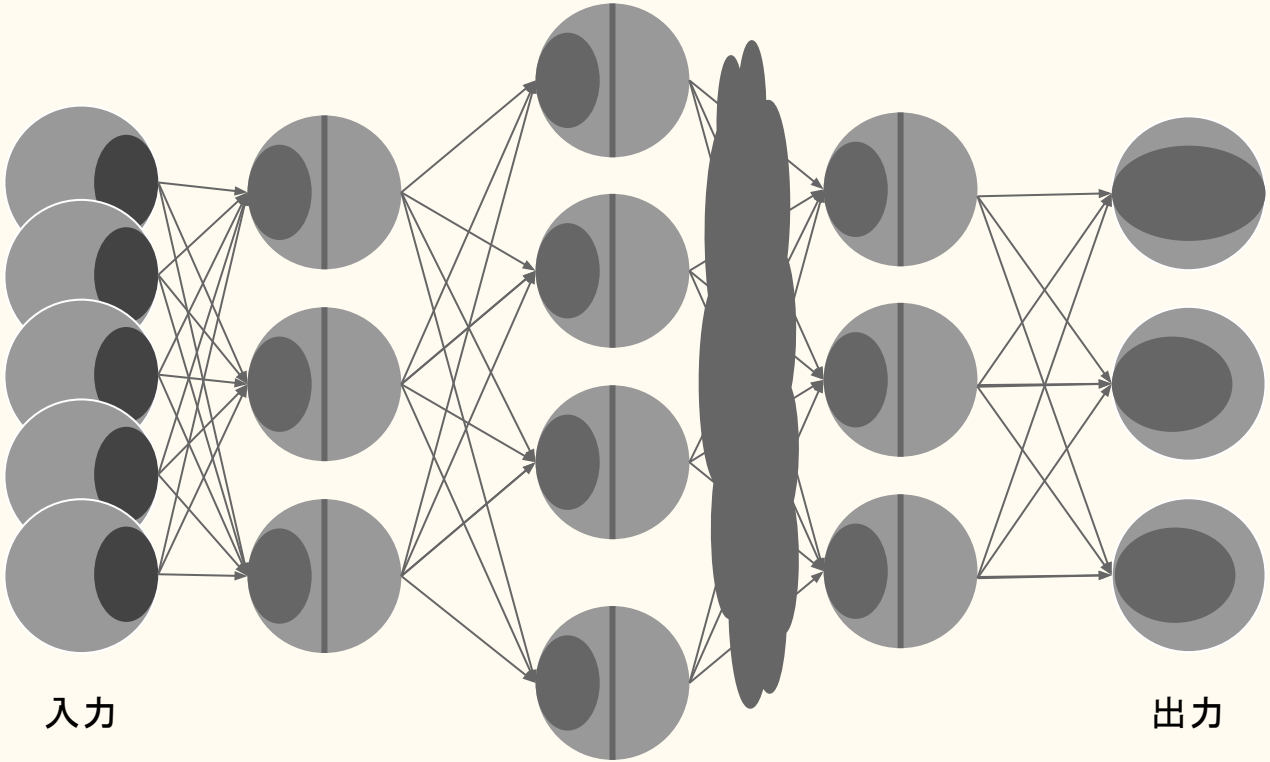
$$\boldsymbol{d}_n = (0, 0, 1, , \cdots, 0)^T$$

  - 尤度関数の構成方法は2値分類の時と同様
    - 構成の流れは割愛 (深層学習, 岡谷 貴之などを参照)

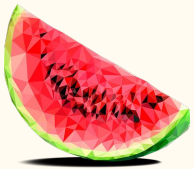
$$E(W^{(1)}, \cdots, W^{(L-1)}, \boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(L-1)})$$

$$= -\sum_{i=1}^{n}\sum_{k=1}^{K} d_{nk} \log y_k(\boldsymbol{x}_i; W^{(1)}, \cdots, W^{(L-1)}, \boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(L-1)})$$

まずパラメータ行列とバイアスに初期値を与える。

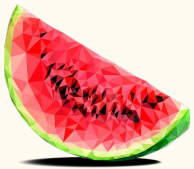


スイカを入力に与えると出力層に結果が出力される



出力
“リンゴ”
(1,0,0)

誤差関数からモデルの出力と正解との誤差を測定

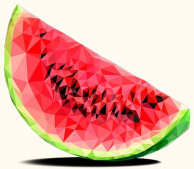


出力
“リンゴ”
(1,0,0)

正解
“スイカ”
(0,0,1)

誤差関数

誤差関数からモデルの出力と正解との誤差を測定



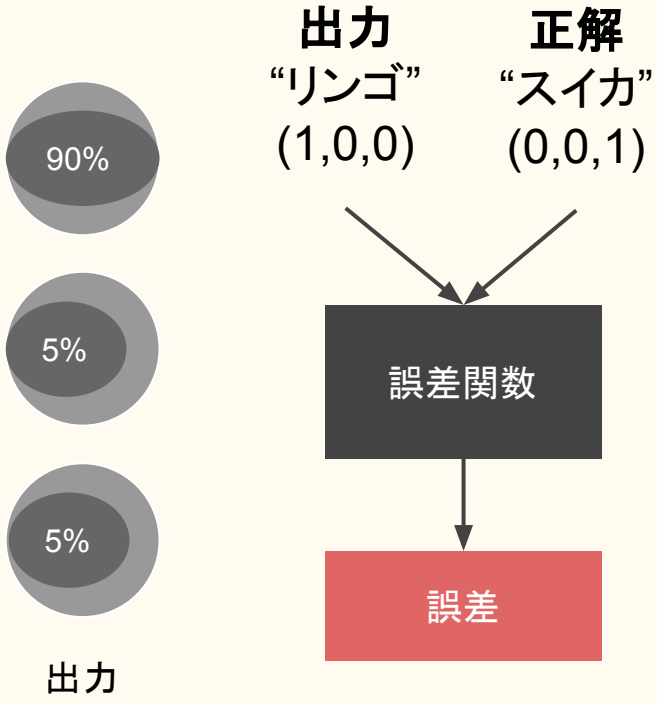
出力
“リンゴ”
(1,0,0)

正解
“スイカ”
(0,0,1)

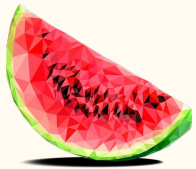
誤差関数

誤差

誤差の計算 (イメージ)



誤差を最小にするようにパラメータを更新 (逆伝播)



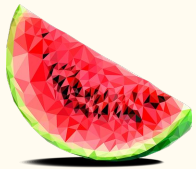
出力
“リンゴ”
(1,0,0)

正解
“スイカ”
(0,0,1)

誤差関数

誤差

誤差を最小にするようにパラメータを更新 (逆伝播)



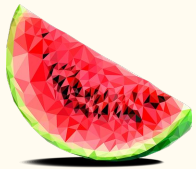
出力
“リンゴ”
(1,0,0)

正解
“スイカ”
(0,0,1)

誤差関数

誤差

誤差を最小にするようにパラメータを更新 (逆伝播)



出力
“リンゴ”
(1,0,0)

正解
“スイカ”
(0,0,1)

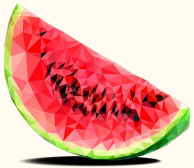
誤差関数

誤差

Newral Networkの学習の様子

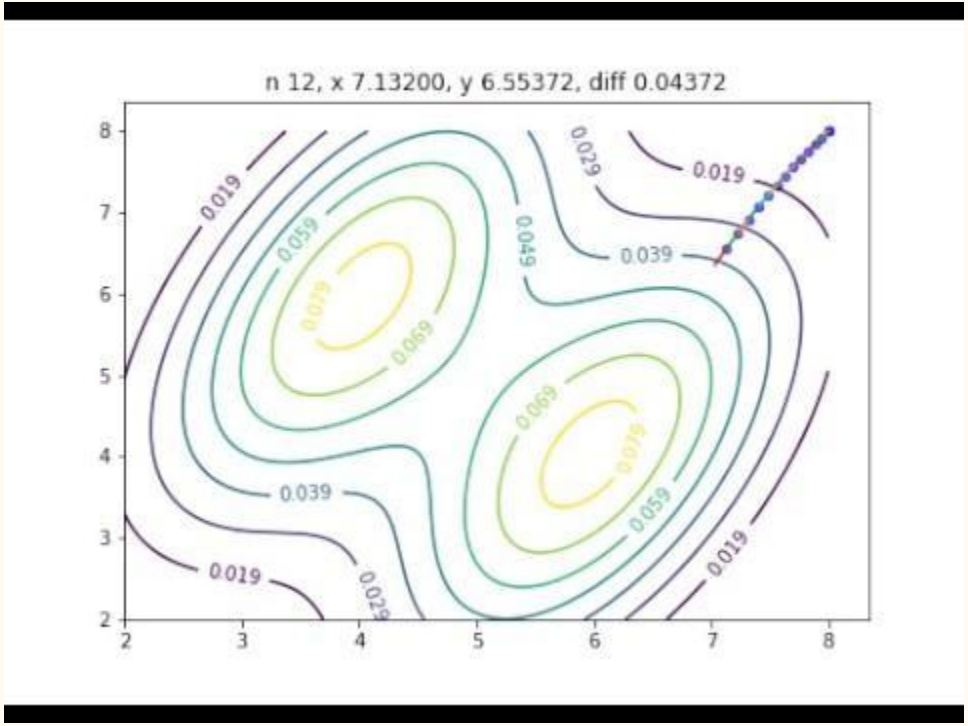
http://hhok777.hatenablog.com/entry/2016/11/08/184233

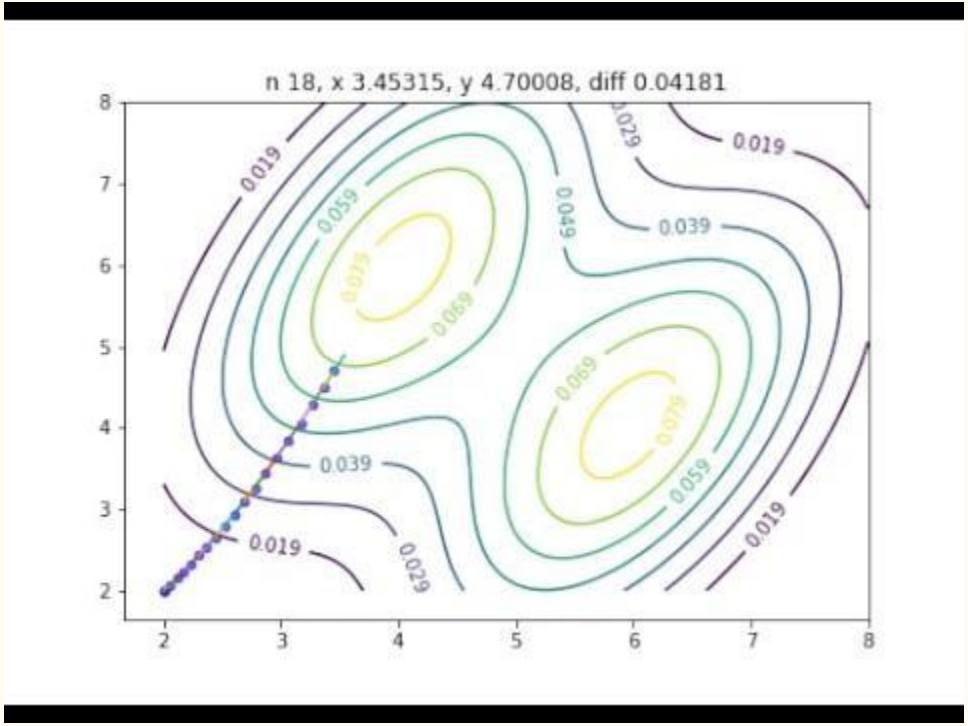
3つのフルーツを識別するモデル



正解!

出力
“スイカ”
(0,0,1)

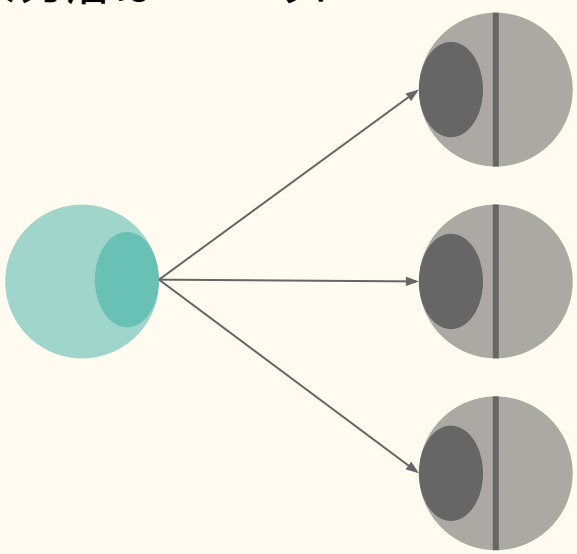

n 12, x 7.13200, y 6.55372, diff 0.04372
0.019
0.029
0.039
0.049
0.059
0.069
0.079
2
3
4
5
6
7
8


n 18, x 3.45315, y 4.70008, diff 0.04181
0.019
0.029
0.039
0.049
0.059
0.069
0.079
1
2
3
4
5
6
7
8

「住宅価格データ」への
DLモデルの適用

ボストン価格データとDeep Learning

入力層は1ユニット



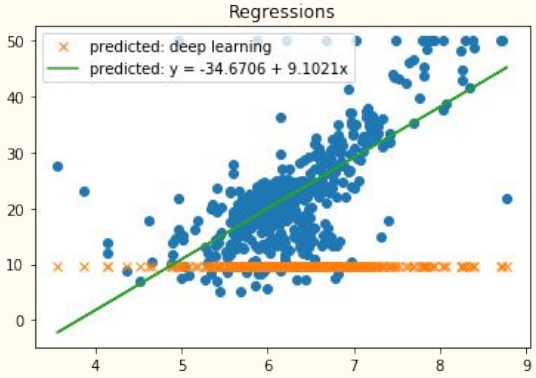
中間層は5層

出力層は1ユニット

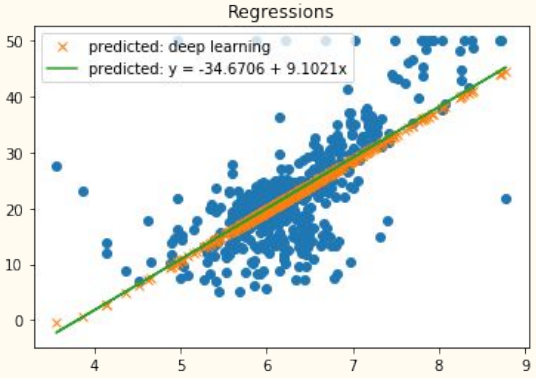
住宅価格

ボストン価格データ：線形回帰とDeep Learning

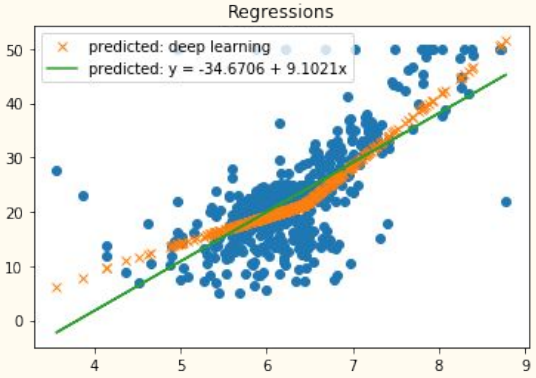
リンクの数を増やせば自由度が高いモデルを構築可能



・隠れ層：5
・ユニット(リンク)数：3
・活性化関数：relu



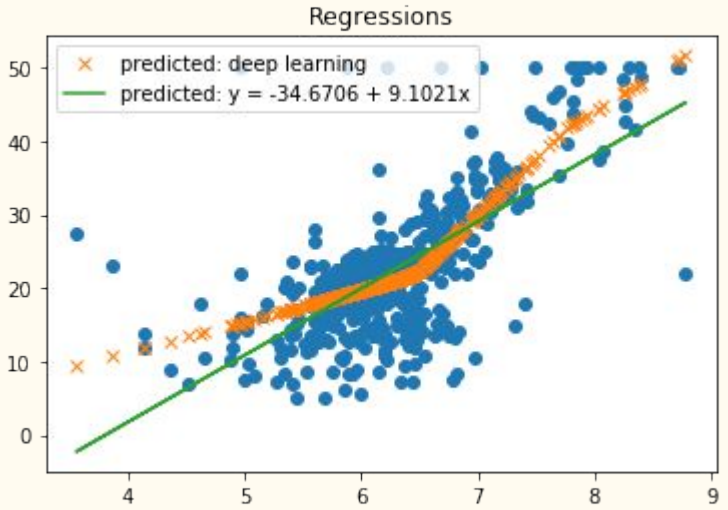
・隠れ層：5
・リンク数：5
・活性化関数：relu



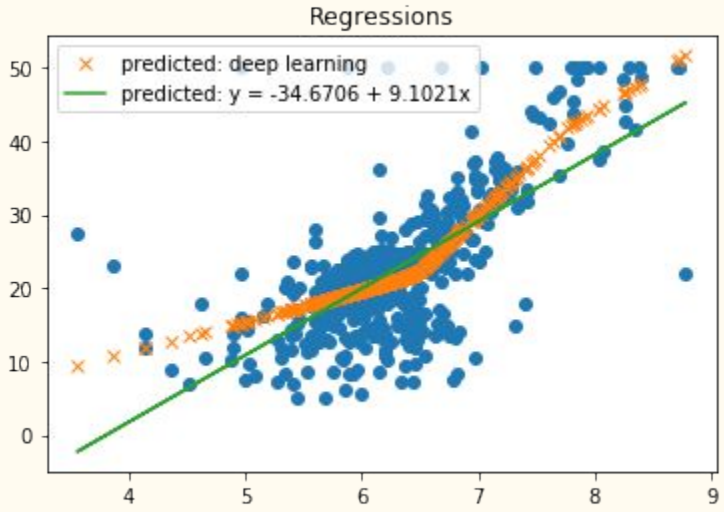
・隠れ層：5
・リンク数：10
・活性化関数：relu

ボストン価格データ：線形回帰とDeep Learning

・隠れ層：5
・リンク数：100
・活性化関数：relu



可能であれば後ほど一緒にやってみましょう。



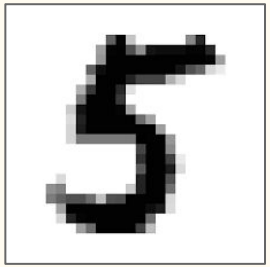
DLによる画像分類

Section 2

1. MNISTの説明
2. 分類モデルの作成

これまでの講義を参考に

Deep Learningで「画像認識モデル」を作ってみましょう



Deep Learning

これは 5 です

データの説明

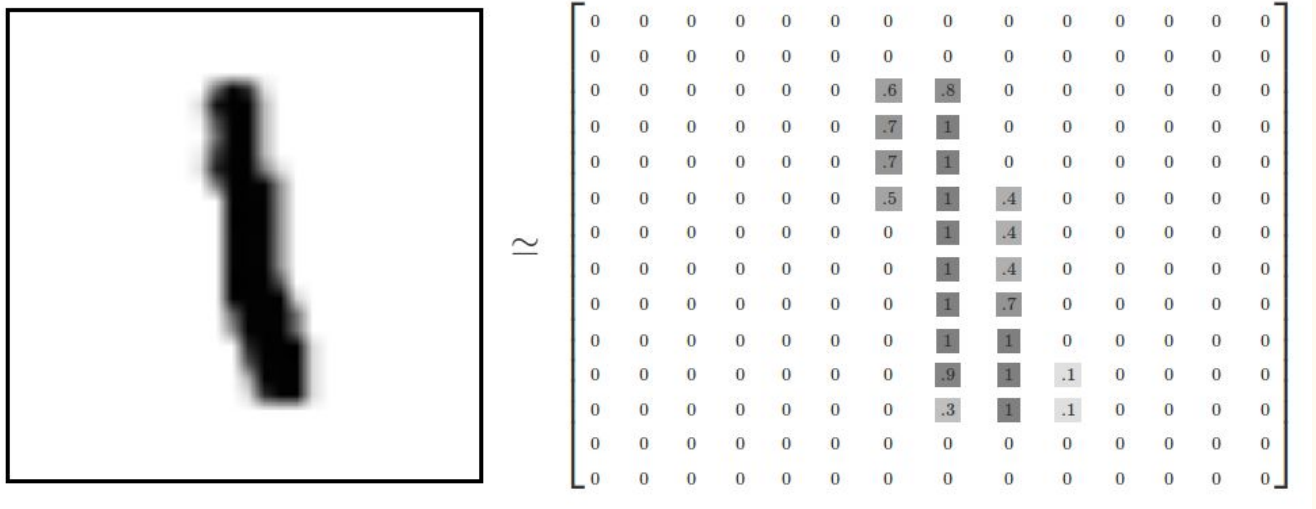
-MNIST-

MNIST (手書き数字データ)

- レコード数:約6万画像
- カラム数:784ピクセル (28 × 28)
- データセットの詳細:

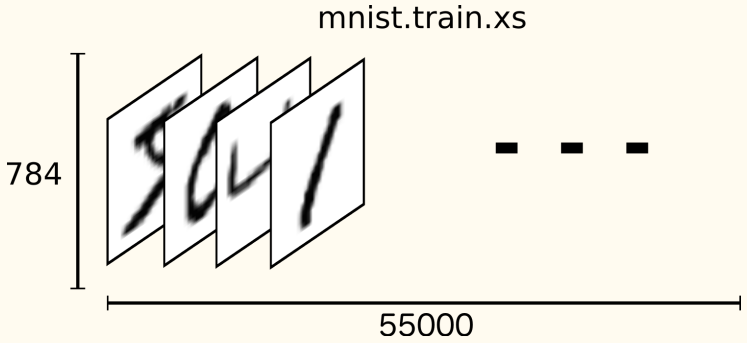
http://tensorflow.classcat.com/2016/03/09/tensorflow-cc-mnist-for-ml-beginners/

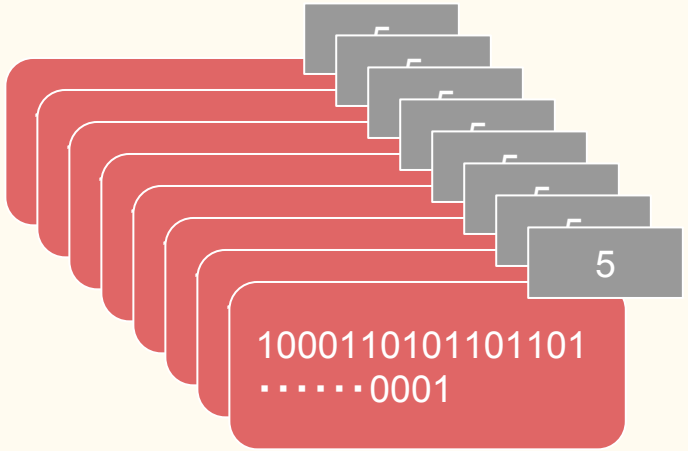
データ詳細



http://tensorflow.classcat.com/2016/03/09/tensorflow-cc-mnist-for-ml-beginners/

データ詳細

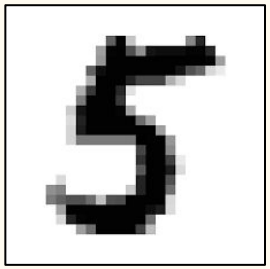




http://tensorflow.classcat.com/2016/03/09/tensorflow-cc-mnist-for-ml-beginners/

画像から数値を読み取る仕組み

- 人が見ると、何が書いてあるかがわかる



人

これは 5 です

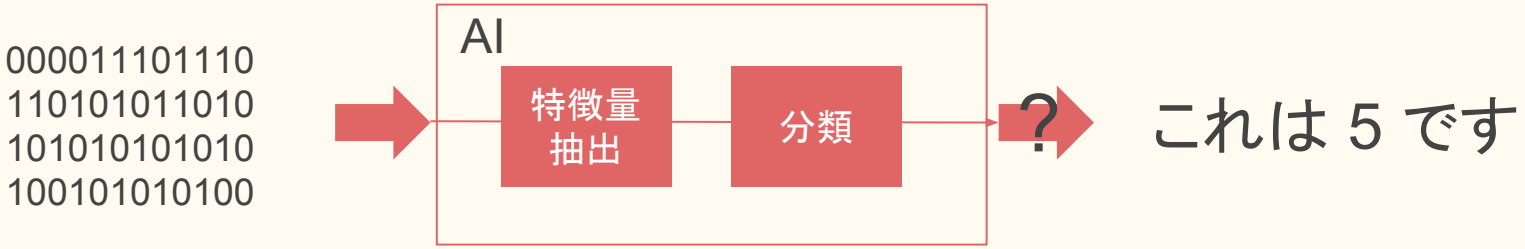
- コンピュータからみると、画像はただの数値の列である。どうやって判断しているのか？

```
000011101110
110101011010
101010101010
100101010100
```

AI

? これは 5 です

画像から数値を読み取る仕組み (詳細はAppendix)

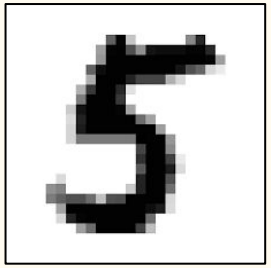


大まかな流れ

1. 入力データを特徴量と呼ばれる、画像の特徴を表す数値に変換
    - 入力データ(白黒画像の場合)は、1ピクセル当たり0 ~ 255の数値で表現される
    - 特徴量は、平均値などの統計量が使用される
2. 特徴量の値を使って、画像を分類
    - 機械学習の分類モデルを使って分類する

入力データ (イメージ)

例えば、28ピクセル x 28ピクセルの画像を、仮に 5ピクセル x 5ピクセルで表現すると



255	0	0	0	255
255	0	255	255	255
255	0	0	0	255
255	255	255	0	255
255	0	0	0	255

数値は、値が大きいほど白色に近く、0に近いほど黒色に近い
(今回は、わかりやすいように背景色を暗くしていますが、実際のデータでは数値のみです)

keras (ケラス)

・Pythonで書かれたオープンソースのDeep Learning用のラッパーライブラリ

・TensorFlowをバックエンドに使用していて、直感的な記述でニューラルネットを記述できるのがkerasを利用するメリット

・直感的に書ける分処理がブラックボックス化するので、一歩踏み込んでDeep Learningを研究したい方はTensor FlowやchainerをTensor Flowを利用するのがオススメ。

・sudo pip install tensorflow keras

※本講義は初心者向けなのでKerasを利用してプログラミングをします。

keras (ケラス)の使い方

・Sequentialでモデルを作成

- model = Sequential

・.add()で入力層、中間層と出力層を設定

- model.add(Dense(units=64, activation='relu', input_dim=100))
- model.add(Dense(units=10, activation='softmax'))

・.compile()で学習プロセスを設定

- model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

keras (ケラス)の使い方

・.fit()で学習を実行

- model.fit(x_train, y_train, epochs=5, batch_size=32)

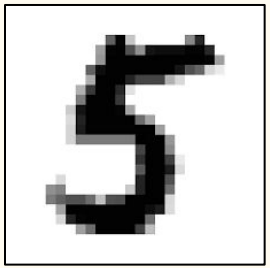
・.predict()で新しいデータに対して予測

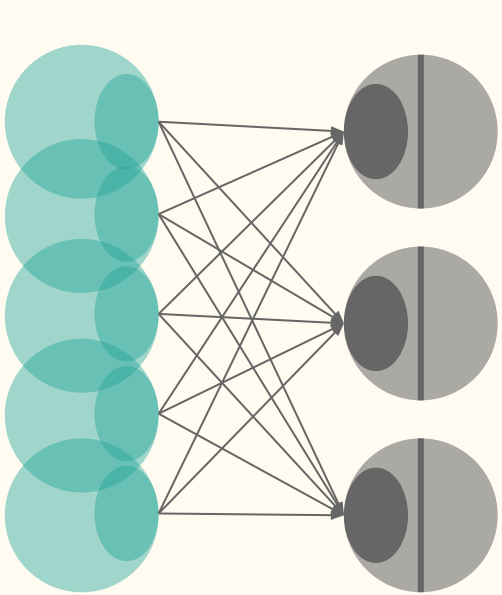
- classes = model.predict(x_test, batch_size=128)

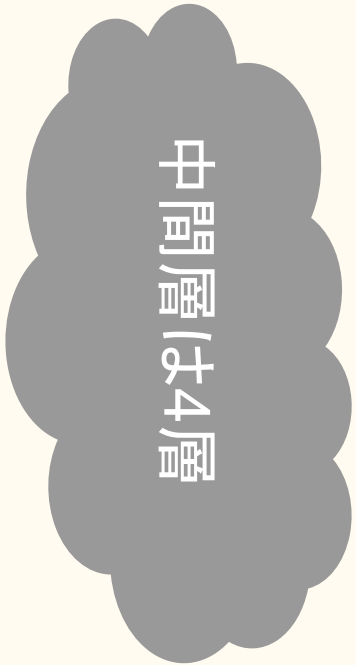
利用するDeep Learningの設定

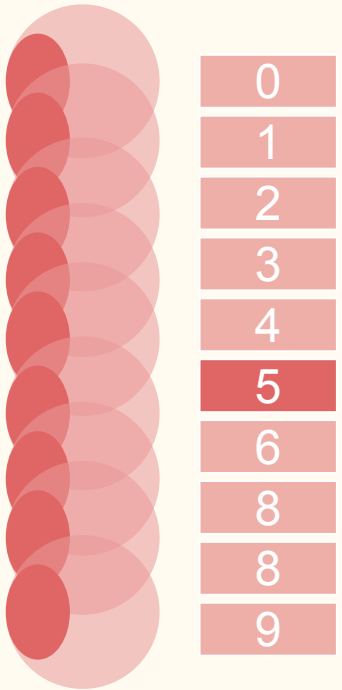
入力層は784ユニット

出力層は10ユニット



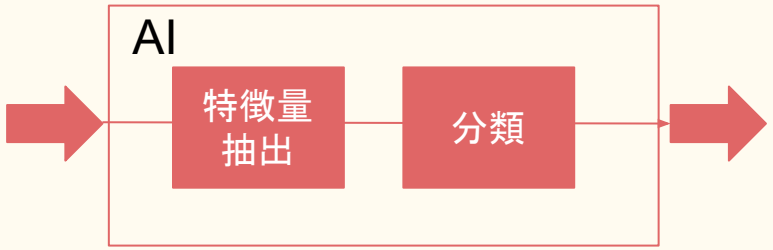






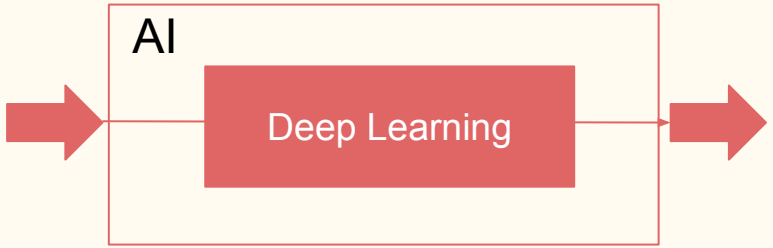
Deep Learningとそれ以外の手法の違い

000011101110
110101011010
101010101010
100101010100



これは 5 です

000011101110
110101011010
101010101010
100101010100



これは 5 です

データの確認と分析

-MNISTデータ-

https://github.com/ded-studyai/studyai0324

ミニバッチ学習

・規模が大きいニューラルネットの学習は大きな計算コストが必要で数値検査を効率化する必要がある。計算機が持つ並列計算資源の利用が不可欠。

・重みの更新をサンプル1つ単位で行うのではなく、少数のサンプルの集合を一まとめにしてその単位で重みの更新を行う。ひとまとめにしたサンプル集合をミニバッチと呼ぶ。

$$E_t(\boldsymbol{w}) = \frac{1}{N_t} \sum_{n \in D_t} E_n(\boldsymbol{w})$$

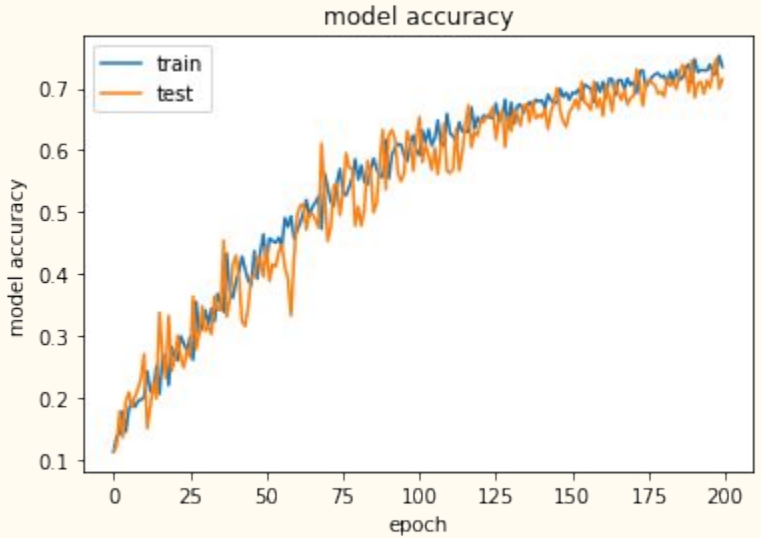
エポック (epoch)

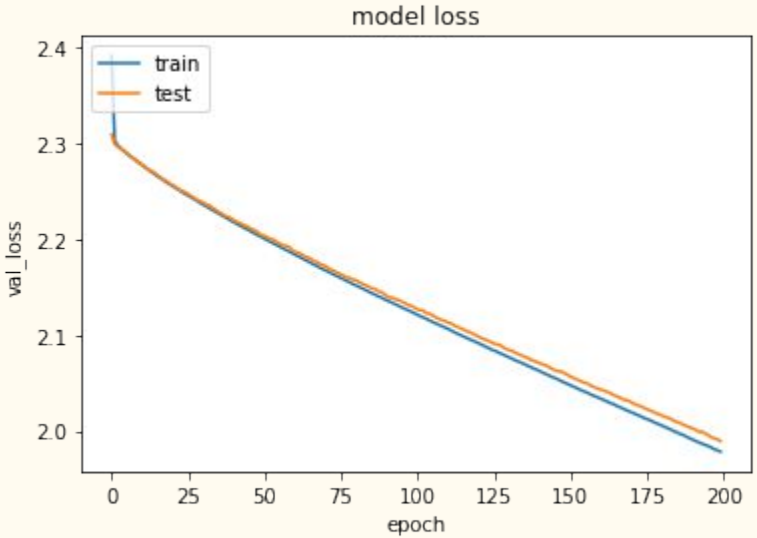
・単位を表す (0から学ぶDeep Learning)

・1エポックとは学習において訓練データを全て使い切った時の回数

・例えば、10000個の訓練データに対して100個のミニバッチで学習する場合、確率的勾配降下法を100回繰り返したら全ての訓練データを「見た」ことになります。この場合、100回＝1エポックとなります。

エポック (epoch)とモデル精度





https://github.com/ded-studyai/studyai00324

DLの歴史

Section 3

1. 3度目のNNブーム
2. 勾配消失と過学習問題
3. ReLUとドロップアウト

ニューラルネットワークの歴史

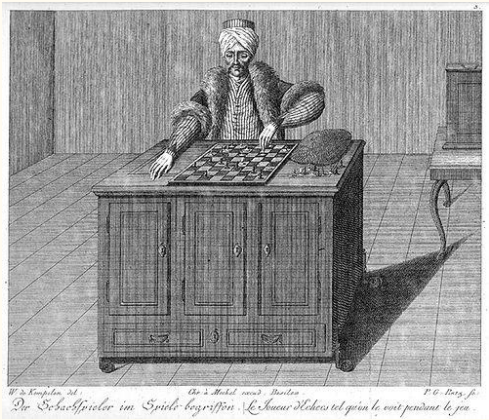
18世紀の機械人形「ターク」(チェス)

・1770年に制作 (**ヨーロッパとアメリカで展示**)

・84年の間に行われたほとんどのチェスの試合に勝利

・実は**チェスの名人が内部に隠れて操** 作する、一種の手品
(ようはイカサマ)

・1820年代にロンドンのロバート・ウィリスが見破るまで誰にもばれなかった



出展：wikipedia (https://ja.wikipedia.org)

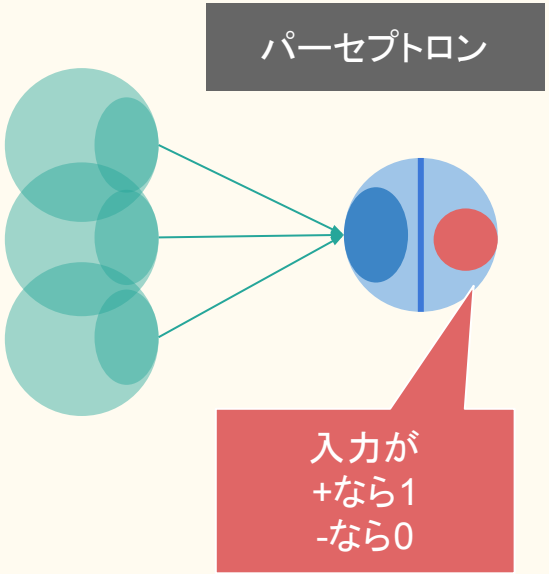
1950~1960年代:NNの第一次ブーム

・1940年代に研究が開始

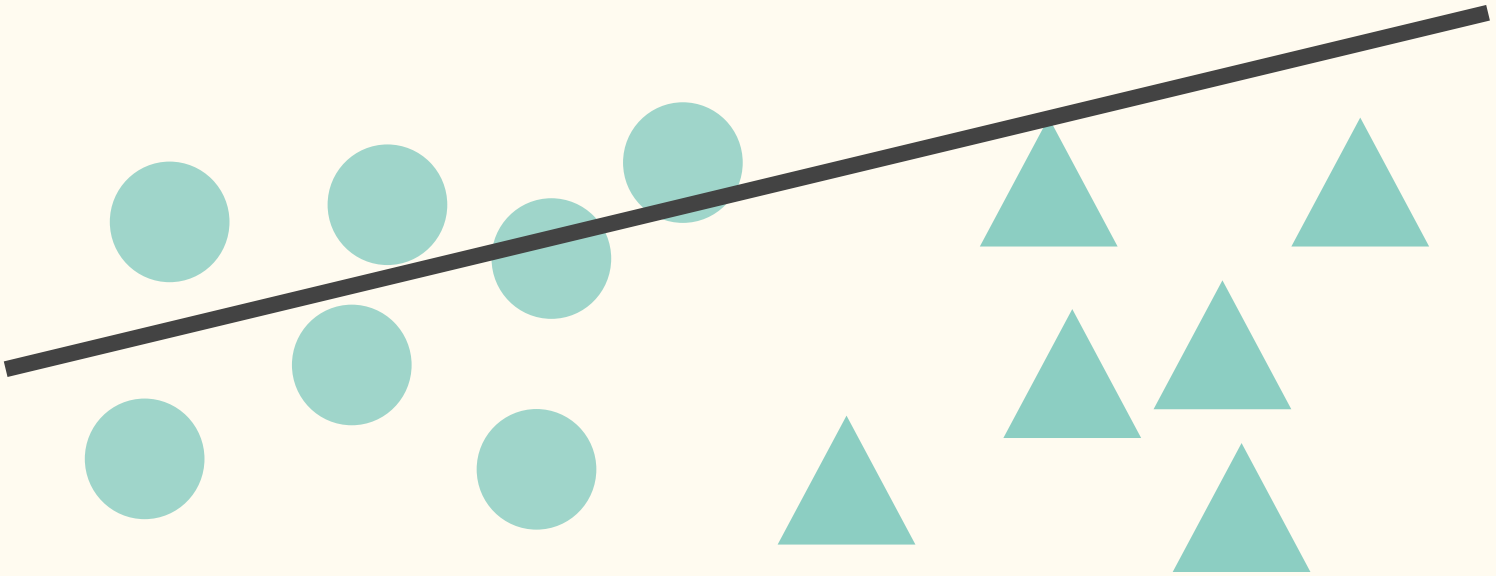
- 人口ニューロンの数理モデルやその学習方法が提案

・1950-60年代にはブームとなる

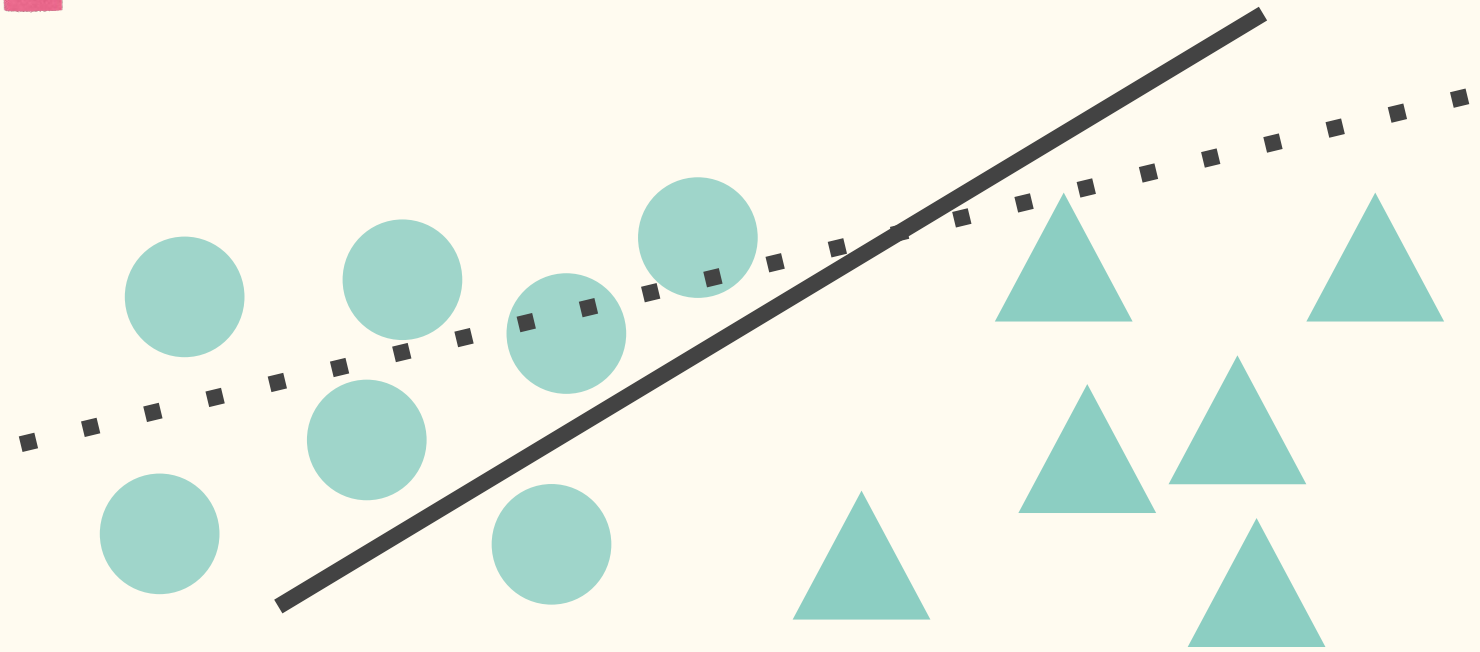
- パーセプトロン提案 (1957年)
- パーセプトロンの収束定理 (Rosenblatt 1962)
  - 与えられた学習データが線形分離可能ならば必ず
    パーセプトロンの学習が収束して学習データ内の点を
    正しく分類する重みを求める事ができる



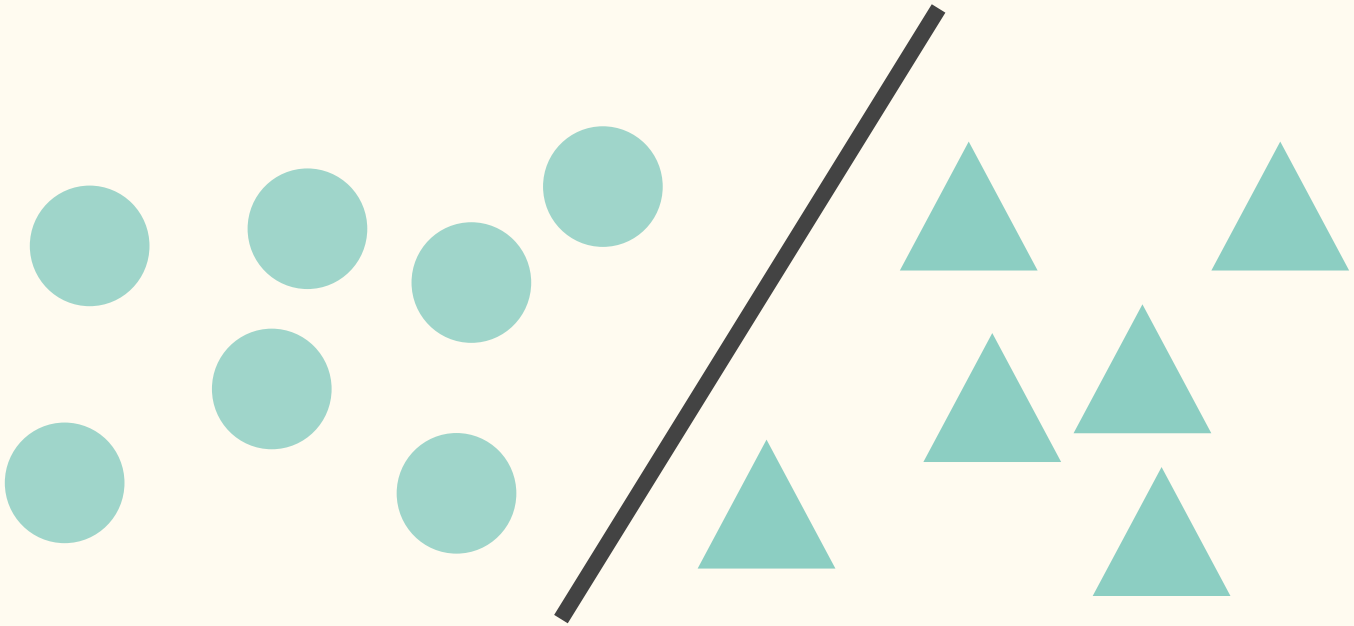
パーセプトロン



パーセプトロン



パーセプトロン



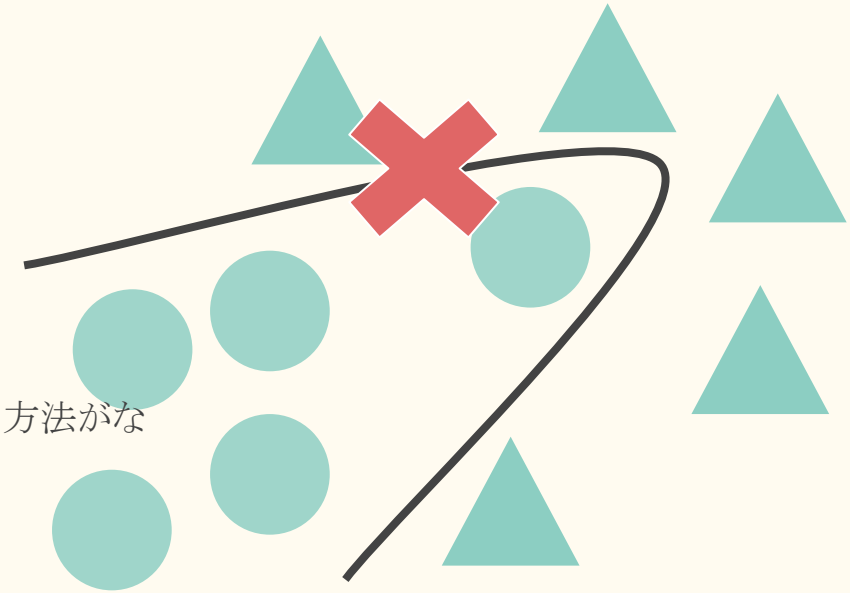
1970年代:第一次ブームの終焉

・1970年代にはブームが終焉し冬の時代へ突入

- パーセプトロンにはいくつかの弱点があったため

・パーセプトロンの弱点

- 収束はするが遅い
- 線形分離不可能な時に重みを計算できない
- 一層のモデルではうまくいくが多層モデルの学習方法がなかった



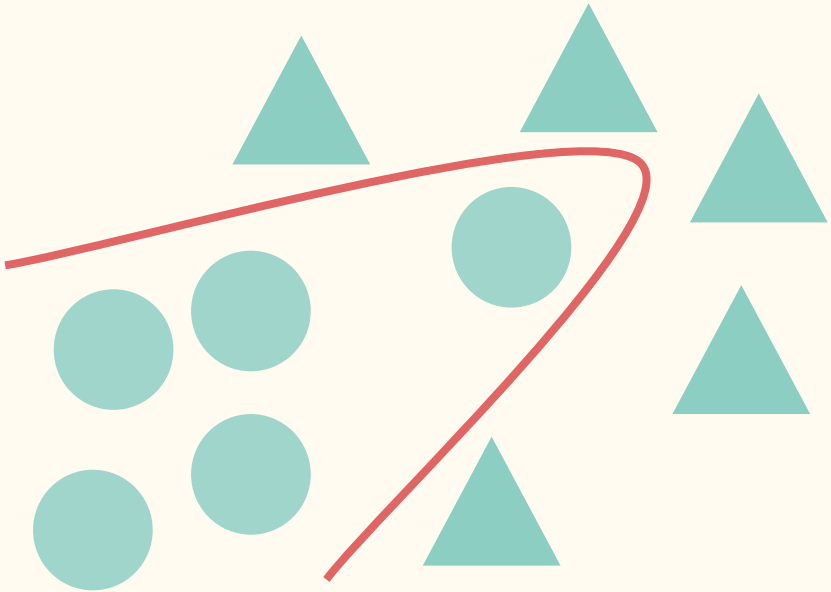
1980年代:第二次ブームの到来

・1980年代より研究が盛んになりブームとなる

- 多層のNNが開発されたため

・多層NN

- 人間の脳を模倣して(複数のニューロンを接続して)作成したネットワーク
- 線形分離でない問題も解ける
- 学習には勾配法と誤差逆伝播法を利用



1980年代：第二次ブームの到来

-第一次ブームとの比較 -

	出力値	出力の仕組み	微分との相性
1950-60年代	真偽の値 (あるクラスに属するか否かのみ)	重みとユニとの線形和の符号のみを利用	悪い
1980年代	確率値 (あるクラスに属する確率)	重みとユニットとの線形和をシグモイド関数で変換	良い (計算コストが小さい)

1980年代：第二次ブームの到来

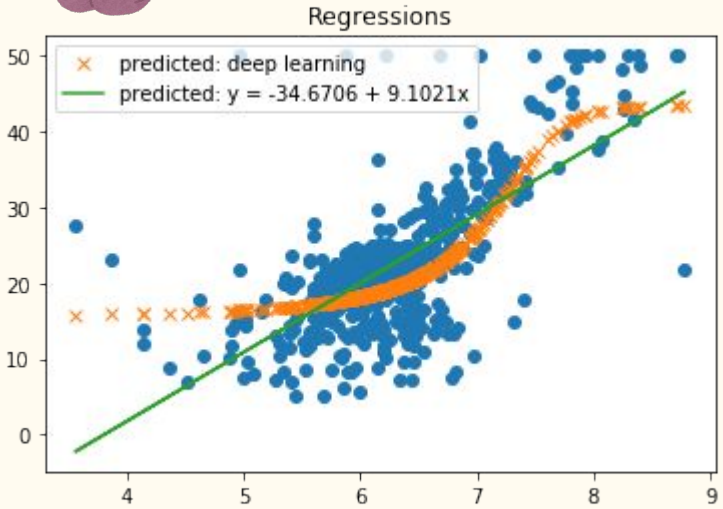
-第一次ブームとの比較 -

	出力値	出力の仕組み	微分との相性
1950-60年代	真偽の値 (あるクラスに属するか否かのみ)	重みとユニとの線形和の符号のみを利用	悪い
1980年代	確率値 (あるクラスに属する確率)	重みとユニットとの線形和をシグモイド関数で変換	良い (計算コストが小さい)

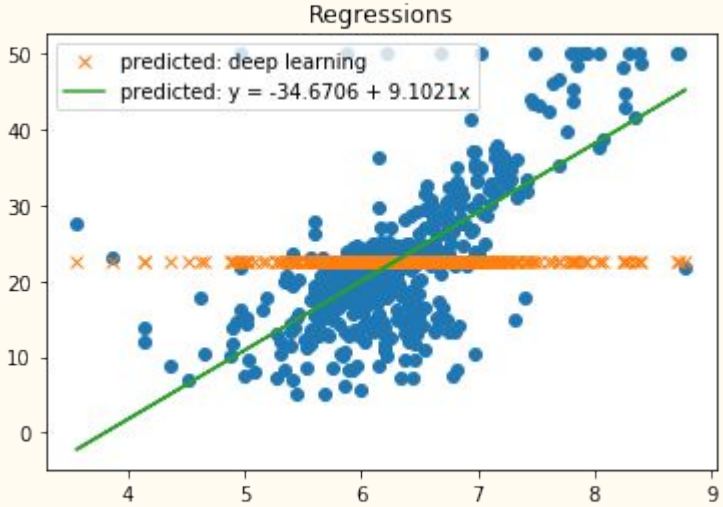
1980年代に開発されたモデルでは「勾配消失」と「過学習」の問題を引き起こしてしまった・・・

第二次ブームの終焉

シグモイド活性化関数の勾配消失 (回帰例)

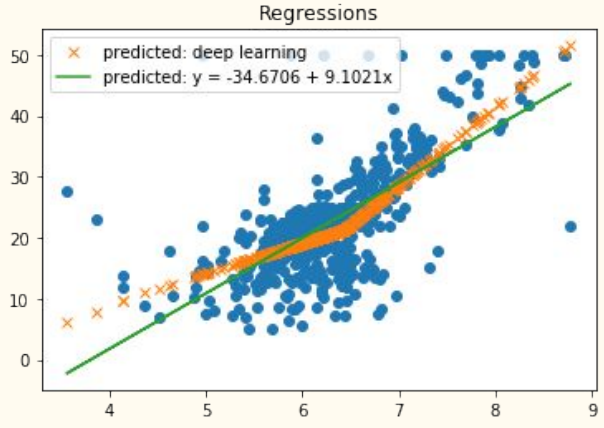


・隠れ層：**5**
・リンク数：100
・活性化関数：sigmoid

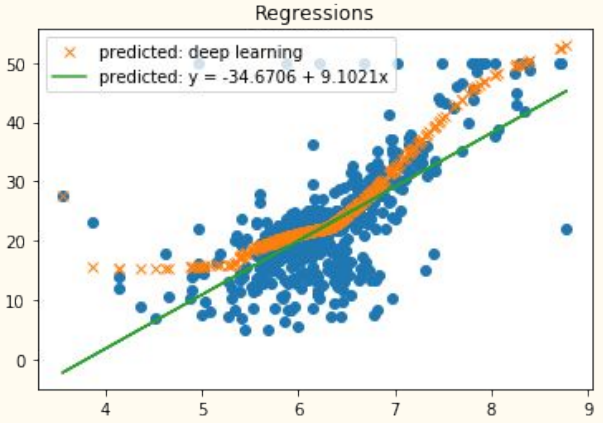


・隠れ層：**10**
・リンク数：100
・活性化関数：sigmoid

層を深くしたことによる過学習 (回帰例)



・隠れ層：5
・リンク数：10
・活性化関数：relu



・隠れ層：**10**
・リンク数：1000
・活性化関数：relu

2010年代~:第三次ブームの到来

- シグモイド関数の代わりにReLu (Rectified linear units)を利用
- Dropout (ランダムにニューロンをマスキングしながら学習すると精度が上昇)

ディープラーニングとして研究が本格化

NNの歴史のまとめ

・第一次NNブーム

- パーセプロトンは優秀だったが、いくつか弱点があった → ニューラルネットワーク
- 効率的に順伝播型NNを学習する方法が無かった → 誤差逆伝播

・第二次NNブーム

- ニューラルネットワークは優秀だが、勾配消失問題が発生 → ReLu活性化関数
- ニューラルネットワークは優秀だが、過学習の問題が発生 → ドロップアウト
- ニューラルネットワークは優秀だが、適切な初期値の問題 → オートエンコーダ (説明していない)

・第三次NNブーム

- 様々なディープラーニングのモデルが開発されている

プログラミング

・線形回帰

- github
- 中間層の活性化関数をsigmoidに変更し、層を深くする
- 過学習を起こし、ドロップアウトを過学習を回避する
- 回帰直線と曲線を出力し結果を比較する

本講義でお話できなかったこと

・誤差逆伝播の計算

- 連鎖率を利用した誤差関数のパラメータに関する微分
- 確率的勾配法を利用したパラメータの更新

・RNN / AutoEncoder / CNNなどのDeep Learningの応用的な使い方

・勾配消失問題やドロップアウトの数学的な記述

・Relu関数には改良版がある Leaky ReLU / Parametric ReLU

・勾配法にも様々な種類がある adadelta・・・ (学習率を大きすると学習が進まない)

参考文献

- 仕事ではじめる機械学習 (有賀康顕)
- 多変量解析入門 (小西貞則)
- http://scikit-learn.org/stable/
- 深層学習 (岡谷貴之)
- 深層学習 (Ian goodflow)
- はじめてのパターン認識 (平井 有三)

ご静聴ありがとうございました！

おまけ

最尤法

-パラメータの決め方-

データの正規化

■ 学習データに対して、値が0~1の範囲に収まるように加工を施すこと

- 学習データの中でもっも大きな値(あるいは最も大きな絶対値)を取り出し、全てのデータをその値で割

$$\alpha = \max\{x_1, \cdots, x_p\}$$

$$\{\frac{x_1}{\alpha}, \cdots, \frac{x_p}{\alpha}\}$$

最小2乗法と最尤法

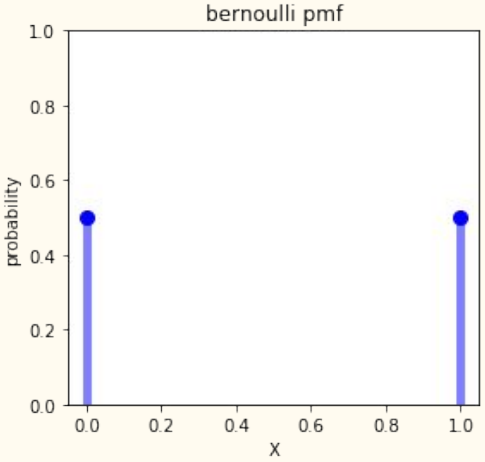
・線形回帰・ロジスティック回帰・ニューラルネットワークなどのパラーメータを推定する際に利用する機械学習アルゴリズム・

・与えられた関数の最小値 (または最大値)を探すことができる

・教師データとのモデルの出力の誤差を計算する損失関数を考えれば。その最小値を探すことで教師データに出力を近づけることができる。

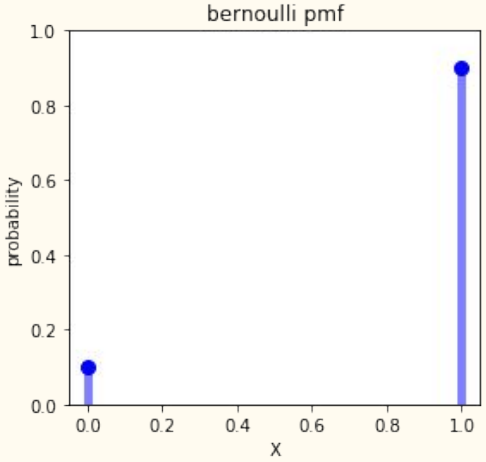
・真の最小値(または最大値)が見つかることが保証されて、局所的な最適解に陥る可能性がある

データは分布に基づいて生成 (2値分類)



$$p = 0.5$$

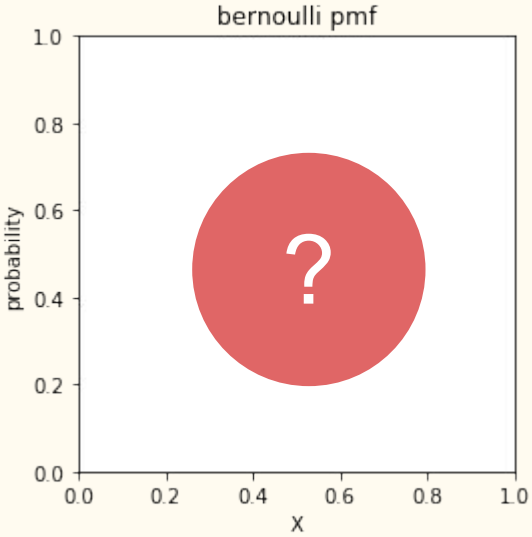
$$y_1 = 1$$
$$y_2 = 0$$
$$y_3 = 0$$
$$y_4 = 1$$
$$y_5 = 1$$
$$y_6 = 1$$
$$y_7 = 0$$
$$y_8 = 0$$
$$y_9 = 0$$
$$y_{10} = 0$$



$$p = 0.9$$

$$y_1 = 1$$
$$y_2 = 1$$
$$y_3 = 1$$
$$y_4 = 1$$
$$y_5 = 1$$
$$y_6 = 1$$
$$y_7 = 0$$
$$y_8 = 1$$
$$y_9 = 1$$
$$y_{10} = 0$$

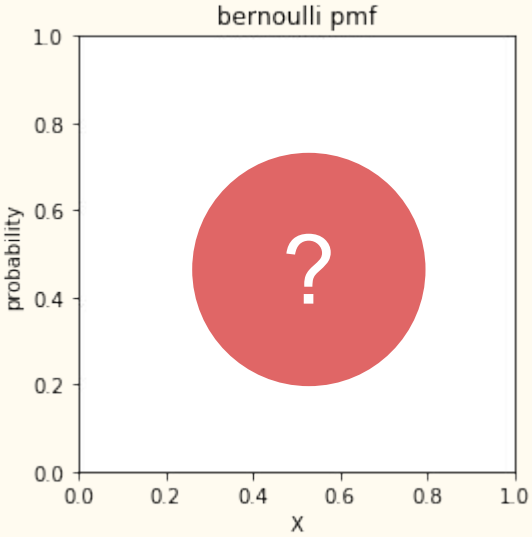
データから分布を特定するには・・・



$$p = ?$$

$$y_1 = 1$$
$$y_2 = 0$$
$$y_3 = 0$$
$$y_4 = 1$$
$$y_5 = 1$$
$$y_6 = 1$$
$$y_7 = 0$$
$$y_8 = 0$$
$$y_9 = 0$$
$$y_{10} = 0$$

データから分布を特定するには・・・



$$p = ?$$

最尤法

$$y_1 = 1$$
$$y_2 = 0$$
$$y_3 = 0$$
$$y_4 = 1$$
$$y_5 = 1$$
$$y_6 = 1$$
$$y_7 = 0$$
$$y_8 = 0$$
$$y_9 = 0$$
$$y_{10} = 0$$

特徴量 (イメージ)

5

255	0	0	0	255
255	0	255	255	255
255	0	0	0	255
255	255	255	0	255
255	0	0	0	255

特徴量	値
0の数	11
255の数	14
縦に0が三つ並んでいる場所の数	2
横に0が三つ並んでいる場所の数	3

1

255	0	0	255	255
255	255	0	255	255
255	255	0	255	255
255	255	0	255	255
255	0	0	0	255

特徴量	値
0の数	8
255の数	17
縦に0が三つ並んでいる場所の数	3
横に0が三つ並んでいる場所の数	1

特徴量 (イメージ)

5ピクセル x 5ピクセルのデータから、特徴量を抽出してみる

255	0	0	0	255
255	0	255	255	255
255	0	0	0	255
255	255	255	0	255
255	0	0	0	255

特徴量	値
0の数	11
255の数	14
縦に0が三つ並んでいる場所の数	2
横に0が三つ並んでいる場所の数	3

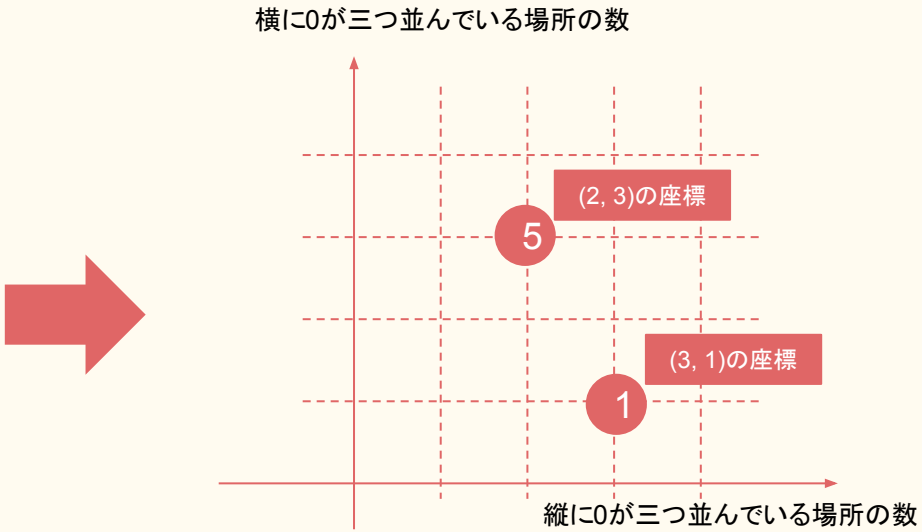
ここに列挙した特徴量を使って、この数値列が 5だと判断できるでしょうか？

特徴量を使った分類イメージ

1と5の特徴量を、特徴量空間にプロットし、線形分離面で分類できるか？を考えてみよう

特徴量	値
縦に0が三つ並んでいる場所の数	2
横に0が三つ並んでいる場所の数	3

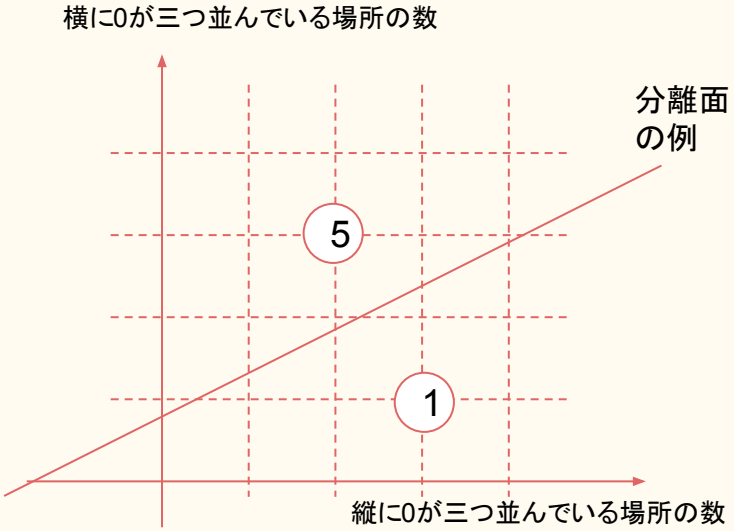
特徴量	値
縦に0が三つ並んでいる場所の数	3
横に0が三つ並んでいる場所の数	1



1と5の特徴量を、特徴量空間にプロットし、線形分離面で分類できるか？を考えてみよう

特徴量	値
縦に0が三つ並んでいる場所の数	2
横に0が三つ並んでいる場所の数	3

特徴量	値
縦に0が三つ並んでいる場所の数	3
横に0が三つ並んでいる場所の数	1



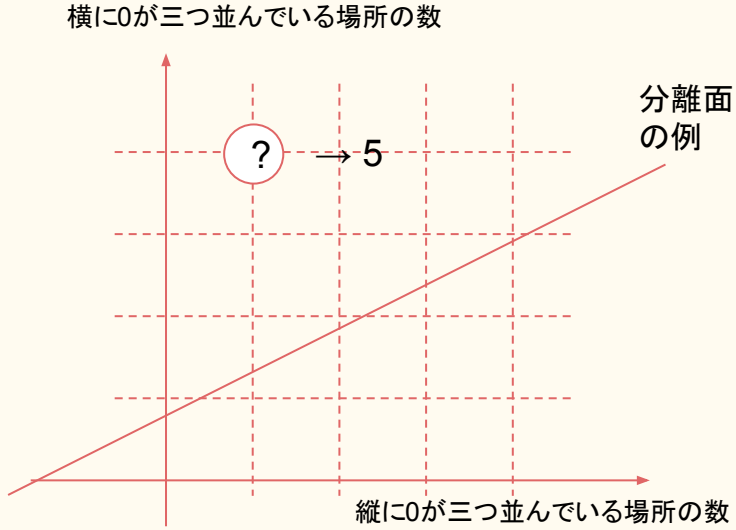
例えば、上記の分離面を使うと、分離面より上は 5
分離面より下は1と分類することができます
(この2つしかデータがない場合です。本当はもっと点があり、分離面も直線では分類できないなど、複雑になります)

特徴量を使った分類イメージ

先ほど学習した分離面を使って、未知データを分類してみる

ある数値を撮影したところ、
画像から以下の特徴量が得られた。
この数値は何か？

特徴量	値
縦に0が三つ並んでいる場所の数	1
横に0が三つ並んでいる場所の数	4



例えば、数値の画像を撮影すると、上記の特徴量が得られた場合、先ほど学習した分離面を使って分類することができます
データから、汎用的な分離面を見つけることを **機械学習** と言います