社会人向けAI・IoT組込み技術を大学で学ぶ講座 AI基礎 (1日目)

二宮 崇

2020年12月5日

スケジュール

- 12月5日（土） 9:00-16:00
  - 深層学習の仕組みを学ぶ【講義】
  - 深層学習器作成のための準備を行う（環境構築とPythonの基礎の学習）【実習】

- 12月6日（日） 9:00-16:00
  - 深層学習の学習方法、畳み込みニューラルネットワークを学ぶ【講義】
  - Pytorchを使いPCで動作する深層学習器を作成する【実習】

自己紹介(1/2)

- 二宮 崇 (にのみや　たかし)
- 愛媛大学 大学院理工学研究科 電子情報工学専攻 教授
- ninomiya@cs.ehime-u.ac.jp

- 略歴
  - 1992-1996 東京大学理学部情報科学科卒業
  - 1996-2001 東京大学大学院理学系研究科情報科学専攻修了(博士)
  - 2001-2006 JST研究員
  - 2006-2010 東京大学情報基盤センター講師
  - 2010-2017 愛媛大学大学院理工学研究科電子情報工学専攻准教授
  - 2017- 現職

自己紹介(2/2)

- 自然言語処理の研究(1995～2006 東京大学 辻井研)
  - 言語学的文法(HPSG)を用いた構文解析の研究
- 機械学習と自然言語処理の研究(2006～2010 東京大学 中川研, 2010～愛媛大学 人工知能研究室)
  - 言語学的文法(HPSG)を用いた構文解析の研究
  - オンライン学習、特徴選択の研究
- 深層学習と自然言語処理の研究(2014～)
  - 深層学習を用いた評判分析、述語項構造の意味表現獲得
  - 機械翻訳、自動要約
  - シンボルグラウンディング、マルチモーダル機械翻訳

本講座の概要

- 深層学習ツールであるPyTorchを用い、IoT環境においてリアルタイムに学習する深層学習の技術について学ぶ。

  前半はPythonコードを基に深層学習について学び、後半は、深層学習ツールであるPyTorchをインストールし、実際にPC上で動作する深層学習器を作成することで、深層学習の技術を学ぶ。

12/5(土)の内容

1. 導入：講座の目的を説明し、機械学習の基礎について学ぶ
2. 環境構築：PCの環境を構築する
3. 深層学習 (1)：ニューラルネットワークの仕組み、活性化関数、推論について学ぶ
4. Python: PythonとPyTorchの基礎を演習を通して学ぶ

12/6(日)の講義内容

1. 深層学習 (2)：ニューラルネットワークの学習，損失関数，勾配法について学ぶ
2. 深層学習 (3)：誤差逆伝搬法，計算グラフ，畳込みニューラルネットワークについて学ぶ
3. PyTorch演習：PyTorchを用いた深層学習の演習を行う．PyTorchを用いて，ニューラルネットワークを実装し，学習，解析を行う．
4. PyTorchによる総合演習 (1)：PC上で動作するPyTorchを用いて，学習/解析する深層学習器を設計，実装する．
5. PyTorchによる総合演習 (2)：実装した深層学習器を改良する．

教材

- 授業は配布資料をベースに進める
- ソースコード

http://aiweb.cs.ehime-u.ac.jp/~ninomiya/enpitpro/

- 参考書
  1. 斎藤康毅, ゼロから作るDeep Learning――Pythonで学ぶディープラーニングの理論と実装, オライリージャパン, 2016.
  2. Guido van Rossum, Pythonチュートリアル 第3版, オライリージャパン, 2016.（https://docs.python.jp/3/tutorial/）
  3. 岡谷貴之, 深層学習 (機械学習プロフェッショナルシリーズ), 講談社, 2015.
  4. 神嶌 敏弘, 機械学習のPythonとの出会い, 2017. (http://www.kamishima.net/mlmpyja/)
  5. Pytorch公式サイト(https://pytorch.org/)

導入

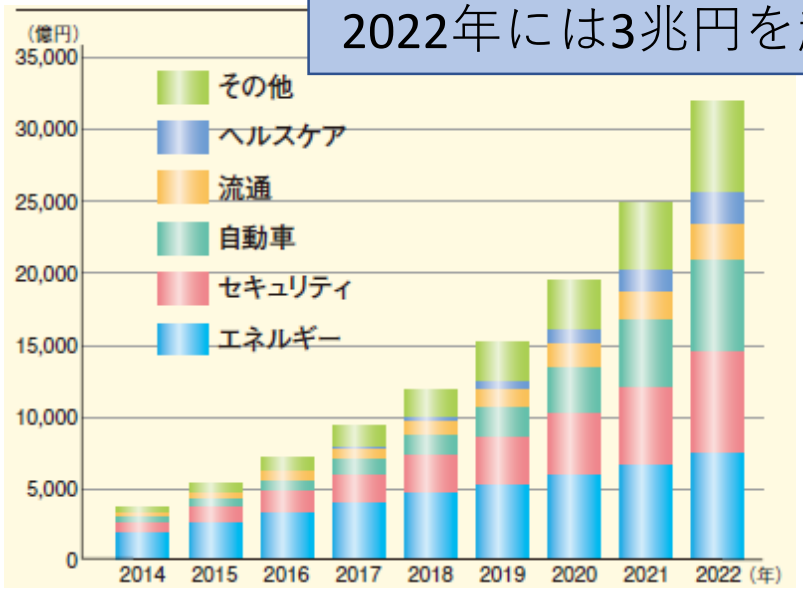
背景 (1/3)

- IoT (Internet of Things; モノのインターネット）市場の急速な拡大
  - ネットワークインフラ、インターネット技術の発展
  - センサー等のハードの低廉化、小型化

  など

IoT市場の分野別規模予測（野村総合研究所 2017）

2022年には3兆円を超える？

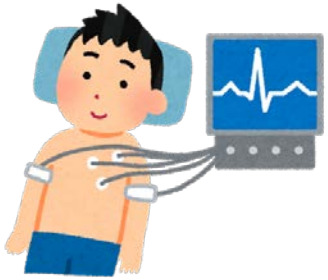


https://www.nri.com/~/media/PDF/jp/opinion/teiki/m_review/2017/nmr38-1.pdf

背景 (2/3)

- 人工知能(Artificial Intelligence; AI)が様々な場面で活躍しはじめている

将棋・囲碁

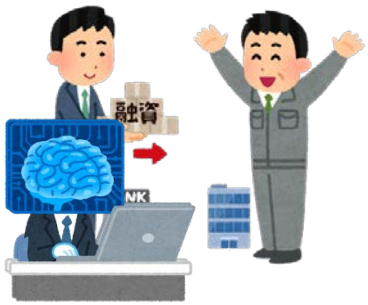


医療診断

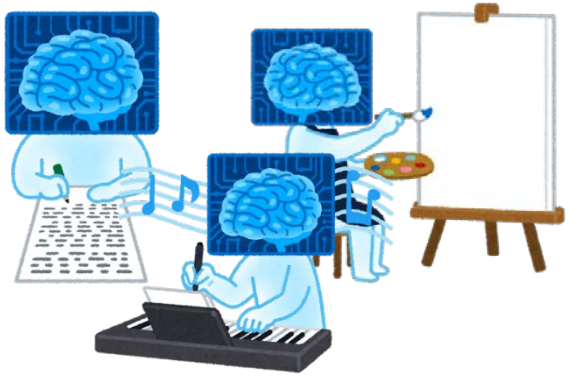
家事手伝いロボット

対話ロボット

証券取引



融資

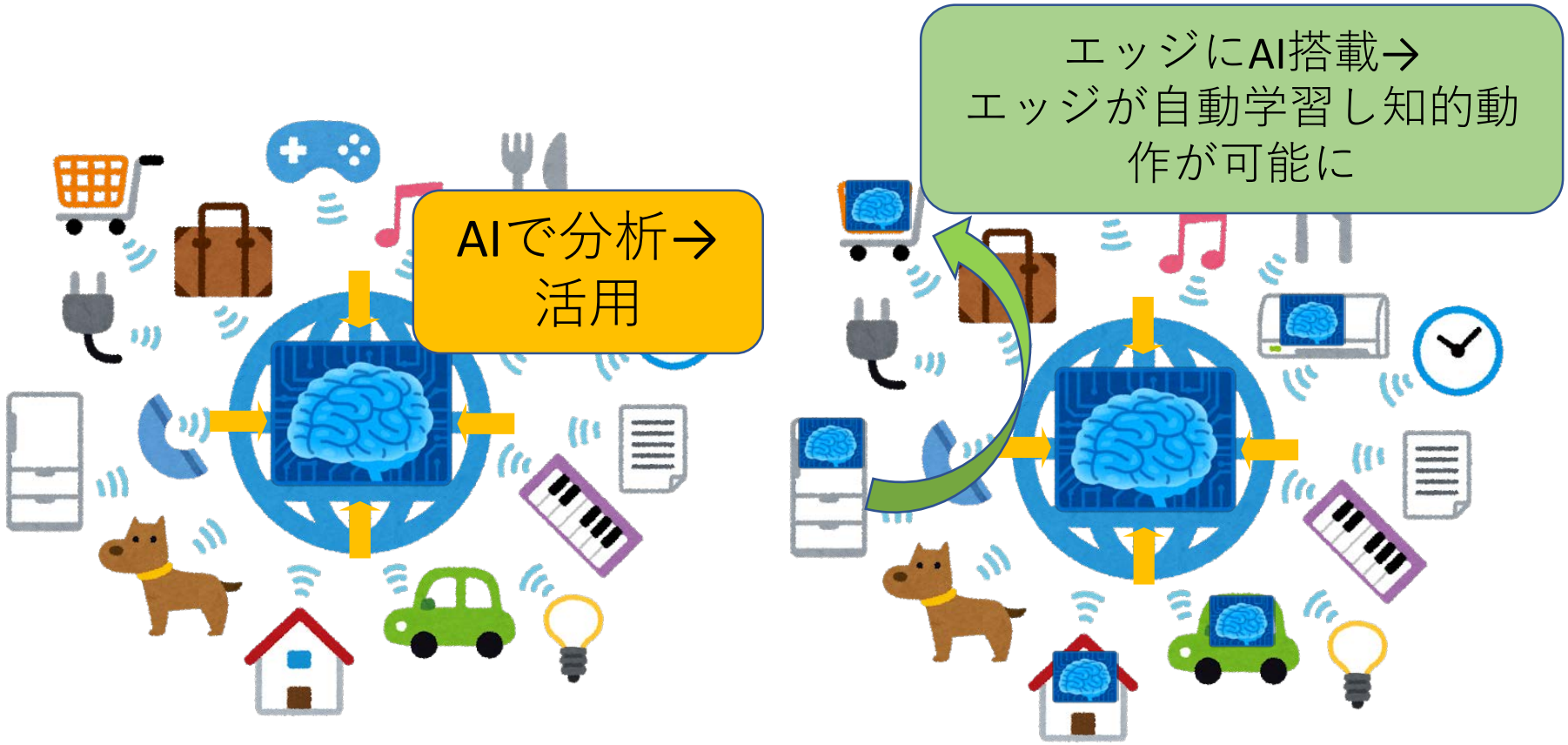


芸術

自動運転

背景 (3/3)

- IoT＋AIが注目を集めている



知的IoT環境

IoT+AIの事例 (1/2)

IoTで見える化

- 人工衛星、ドローン、圃場に設置した各種IoTセンサーのデータから圃場の見える化

＋

AIで分析

- AIによる生育シミュレーションから農業行動のアドバイス、収穫量などの予測

→ 加工用トマトの生産性を10%高めることに成功

https://wisdom.nec.com/ja/events/2017121301/index.html

IoT+AIの事例 (2/2)

- AI搭載の専用EV車両により無人宅配（公道を封鎖しての実証実験成功）
  - AIによる配送ルートの最適化
  - AIによる自動運転

→人件費削減、顧客満足度向上

https://www.roboneko-yamato.com/

本講座では、

IoT+AI

↑理論と技術を学ぶ

講義：AI（深層学習）の理論を学習
実習：PC上でAI（深層学習器）を作成
オプション実習：Jetson Nano上でAI（深層学習器）を作成

※IoT環境下を想定し、低価格で超小型なシングルボードPCでAIを実現する（深層学習器を作成する）

Jetson Nano
大きさ：100 x 80 x 29mm
価格：約15,000円

人工知能(Artificial Intelligence; AI)とは？

コミュニケーションロボット

自動翻訳機／プログラム

形態は問わない

パズルを解くプログラム（候補の全探索）

	2		3	8	5		7	
	5	1				9	8	
			4		1			
	7	4		3		6	5	
	1		6		7		3	
6		3		5		4		7
	8			6			9	
7	4		9		8		6	2
	6		7		3		4	

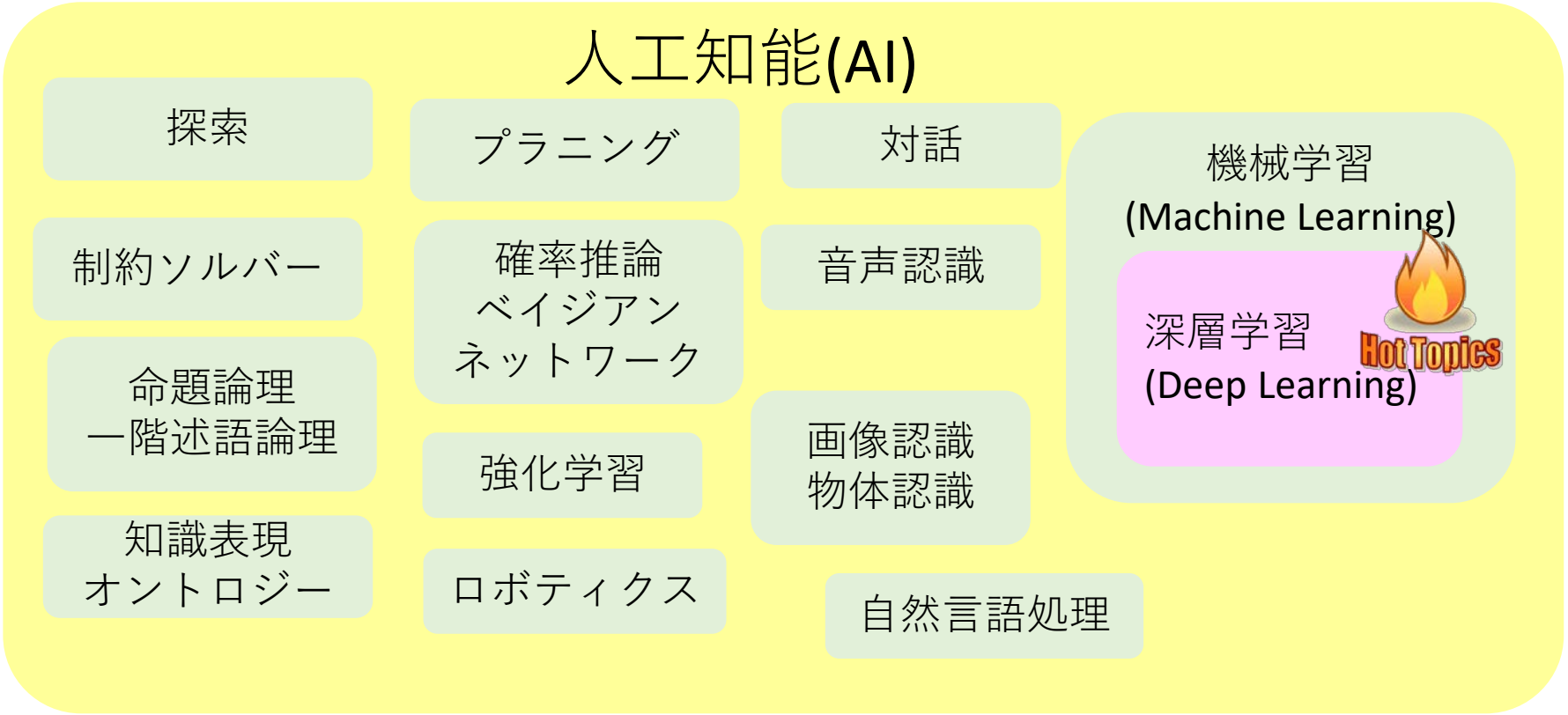
振る舞いが知的であればよい

電卓

「人間が知能を使って行うことを実行できるモノ(機械、プログラム、技術)」

※ 研究者間で共有された明確な定義はない
※ 知性や知能の定義自体がない

人工知能分野



機械学習：データから法則を見つけ出す手法

深層学習：ニューラルネットワーク(NN)を複数層重ねたモデル

Stuart Russell, Peter Norvig (2010) **Artificial Intelligence: A Modern Approach, 3rd Edition.**

AIの歴史

- AIブームと冬の時代

1950年代　1960年代　1970年代　1980年代　1990年代　2000年代　2010年代

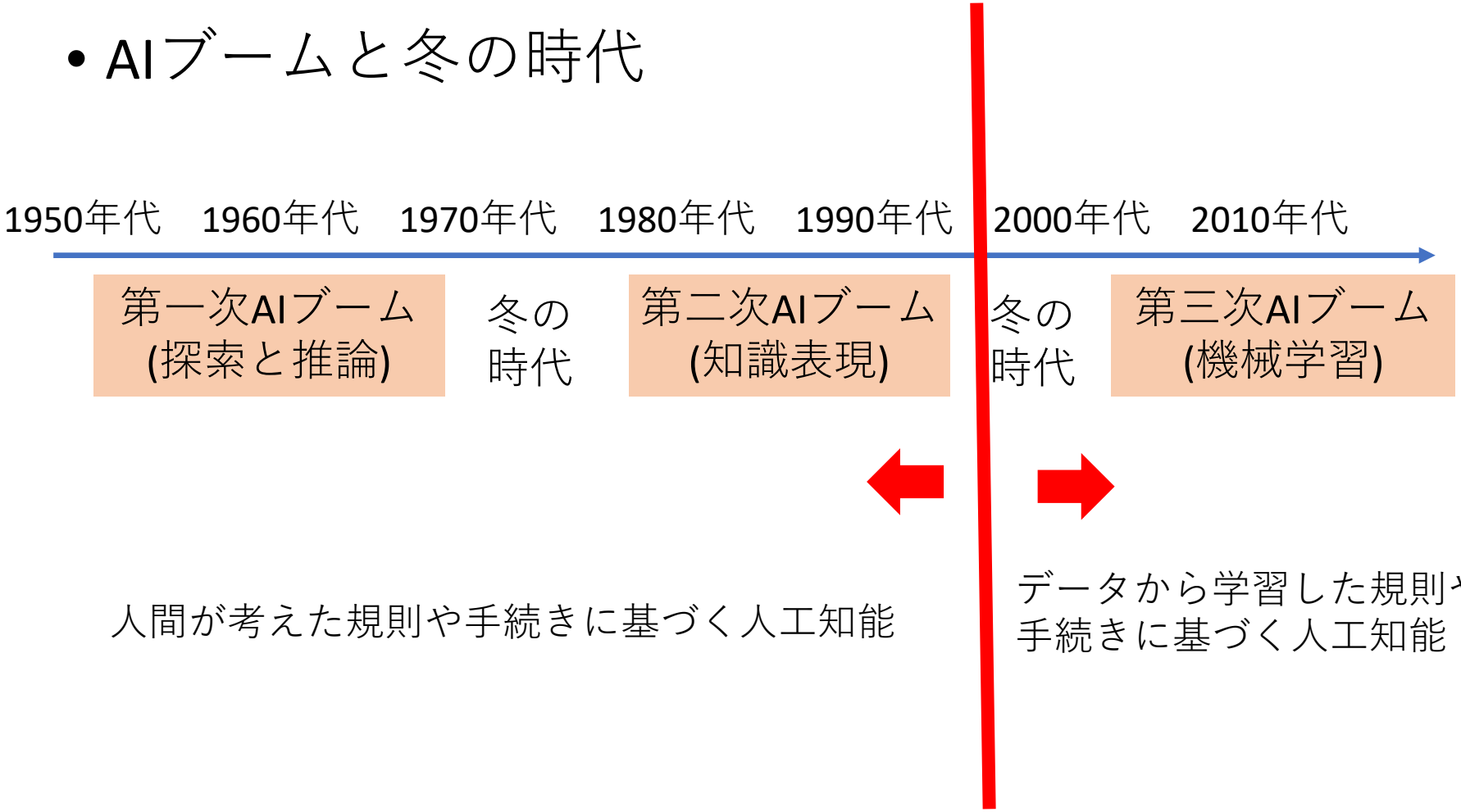
第一次AIブーム (探索と推論)　冬の時代　第二次AIブーム (知識表現)　冬の時代　第三次AIブーム (機械学習)

社会の期待 >> 技術的限界

冬の時代を迎えたくない…

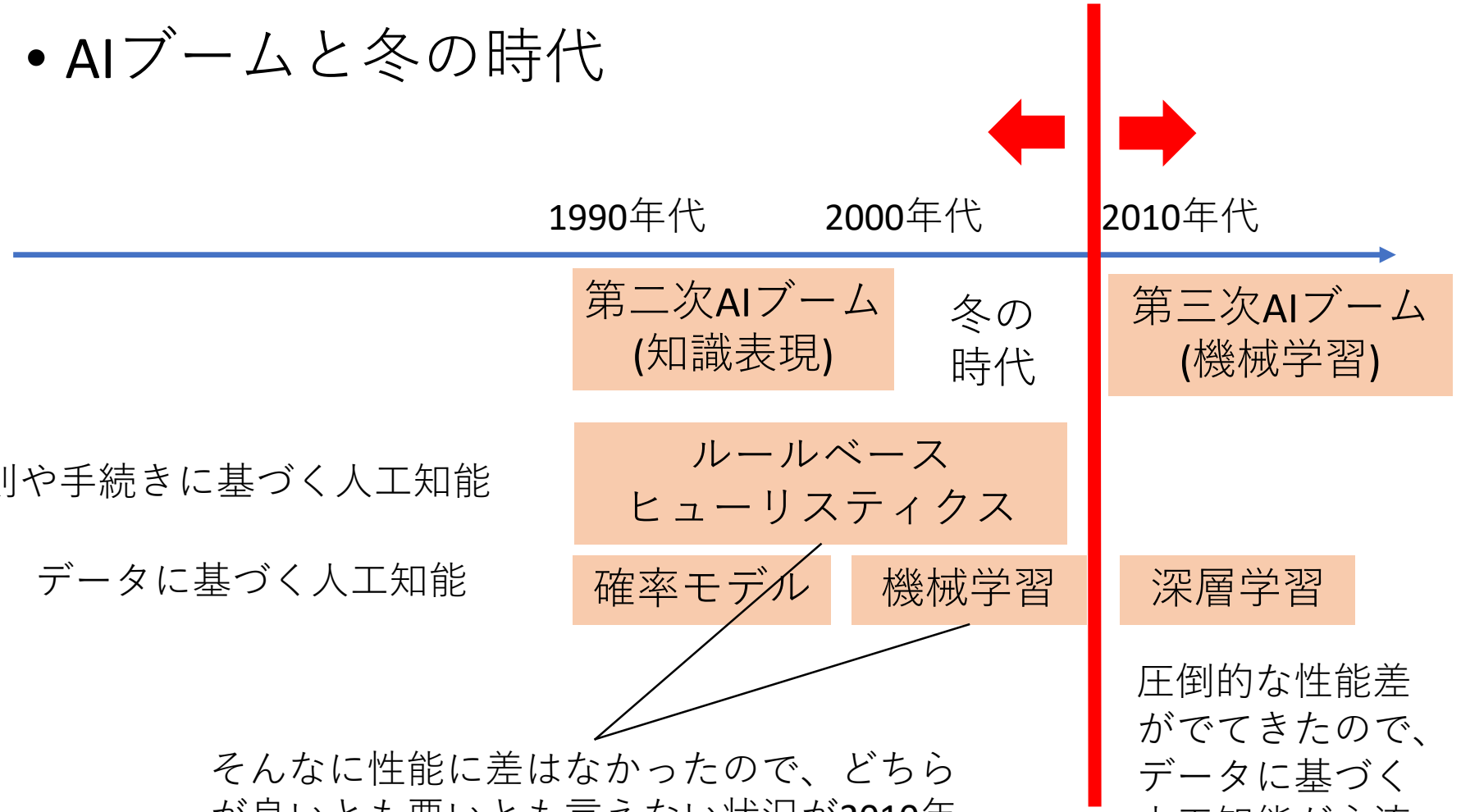
AIの歴史

- AIブームと冬の時代



AIの歴史

- AIブームと冬の時代

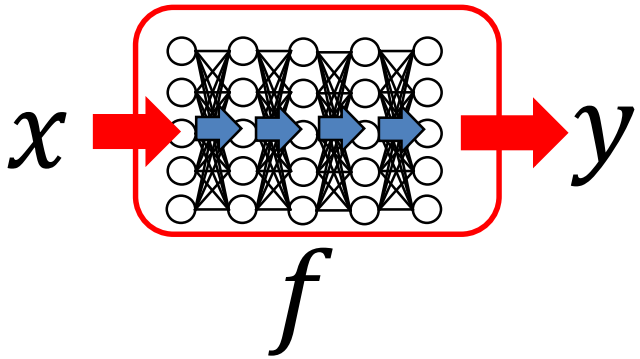


機械学習

- 機械学習 = データから関数を学習する学問分野

大量のデータ  $f(x)$

- 深層学習も機械学習の一種

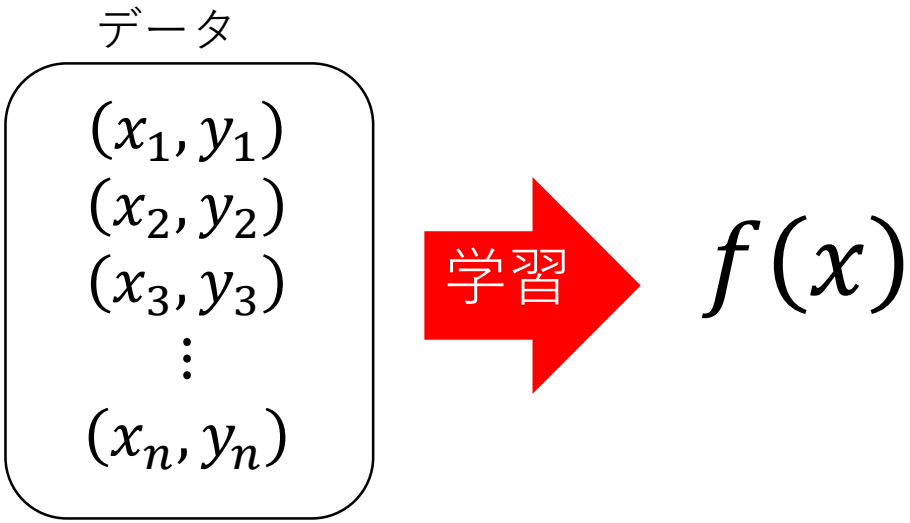


機械学習

- データから関数を学習
  - 関数は、入力(x)と出力(y)の関係を表す

$$x \xrightarrow[f]{} y$$

  - 大量の入出力ペア(x, y)の集まり(データ)から関数fを自動的に学習！



機械学習

例：犬猫判別機 $f(x)$

入力x

f

出力y

犬

データ


：猫

：犬


：犬

：猫


：犬 …

大量のデータから関数f を学習する　f: 画像 → {犬/猫}

機械学習

- データから学習
  - 入力 $\boldsymbol{x} = (x_1, x_2, \cdots, x_m)$ ←m次元ベクトル
    (画像の場合は、赤、緑、青の画像に対応する3つの行列)
  - データDは入力$\boldsymbol{x}$と出力$\boldsymbol{y}$のペアの集合

  - 関数fはパラメータ(重み変数)の集合 $\boldsymbol{w} = (w_1, w_2, \cdots w_m)$から成る
    例: $f(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m$
    - fは$f(\boldsymbol{x}, \boldsymbol{w})$と書くとわかりやすい

  - 学習 = データDに対する誤差を最小にする重み変数$\boldsymbol{w}$を求める
    データ全体の誤差の例 $E(\boldsymbol{w}) = \sum_{(\boldsymbol{x},y)\in D} \big(y - f(\boldsymbol{x}, \boldsymbol{w})\big)^2$

機械学習の教科書

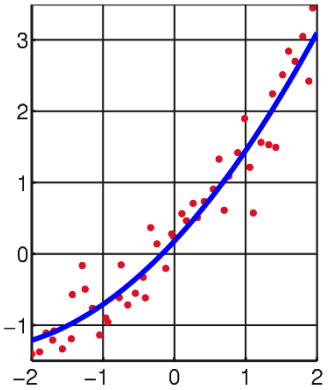
- 機械学習
  - Christopher M. Bishop, Pattern Recognition and Machine Learning
    - (邦訳) C. M. ビショップ他 パターン認識と機械学習 上・下
  - Nello Cristianini, John Shawe-Taylor, An Introduction to Support Vector Machines and other kernel-based learning methods
  - Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction
    - (邦訳) Trevor Hastie, Robert Tibshirani, Jerome Friedman他,統計的学習の基礎 ―データマイニング・推論・予測―
  - 中川裕志,東京大学工学教程 情報工学 機械学習
  - 岩田具治, トピックモデル (機械学習プロフェッショナルシリーズ)
- 数値最適化
  - Jorge Nocedal, Stephen Wright, Numerical Optimization
  - Dimitri P. Bertsekas, Nonlinear Programming
  - 金森 敬文, 鈴木 大慈, 竹内 一郎, 佐藤 一誠, 機械学習のための連続最適化 (機械学習プロフェッショナルシリーズ)

回帰問題と分類問題と構造予測

関数: $y = f(\boldsymbol{x})$

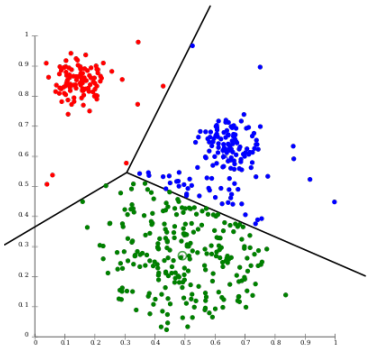
- 回帰問題 (regression)
  - $y \in R$ (実数)
  例: 年齢予測、降水確率の予測、気温の予測

回帰



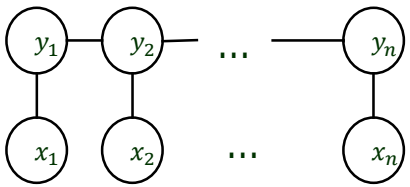
- 分類問題 (classification)
  - $y \in \{C_1, C_2, \cdots, C_K\}$ (ラベル集合)
  例: 文書分類(政治、経済、スポーツ等)

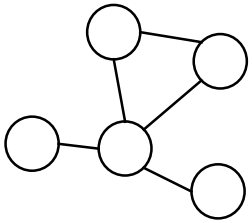
分類



- 構造予測 (structured prediction)
  - $y \in G$ (グラフ集合)

構造予測





教師付き学習と教師無し学習

- 教師付き学習
  - 入出力ペア$(\boldsymbol{x}, y)$の集まり(データ)から関数fを予測
- 教師無し学習
  - 入力xの集まり(データ)から関数fを予測

教師付き学習
(supervised learning)

$(\boldsymbol{x}_1, y_1)$
$(\boldsymbol{x}_2, y_2)$
$(\boldsymbol{x}_3, y_3)$
$\vdots$
$(\boldsymbol{x}_n, y_n)$

学習 → f

学習には、入力(x)と出力(y)のペアが揃ったデータが必要！

教師無し学習
(unsupervised learning)

$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$
$\vdots$
$\boldsymbol{x}_n$

学習 → f

そんなに高い精度を実現するわけではない

機械学習における学習と推論

- 学習 (learning)
  - データから良いパラメータを推定すること
  - 誤差関数を最小化することによりパラメータが得られる
  - 推定 (estimation)、パラメータ推定 (parameter estimation) ともいう
  - 最小二乗法、最尤推定、MAP推定、ベイズ推定、マージン最大化が有名

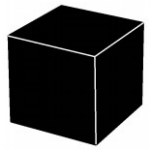
- 推論 (inference)
  - 未知のデータに対し、学習した関数fを適用し、出力を予測すること
  - 予測ともいう

機械学習のあれこれ

- 10年前、ニューラルネットワークはまったく見込みのない技術だった

- 機械学習は数学的に説明がつくエレガントな体系だったが、ニューラルネットワークは数学的に何をやっているのかよくわからない。

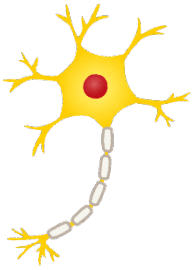
NN = ブラックボックス?



- 何か役に立つのか？
  - 10年前は顔認識ぐらいが唯一のアプリ？
  - 精度100%には(事実上)ならない。いつかかならず判断を間違える。(つまり、クリティカルな仕事には使えない。)

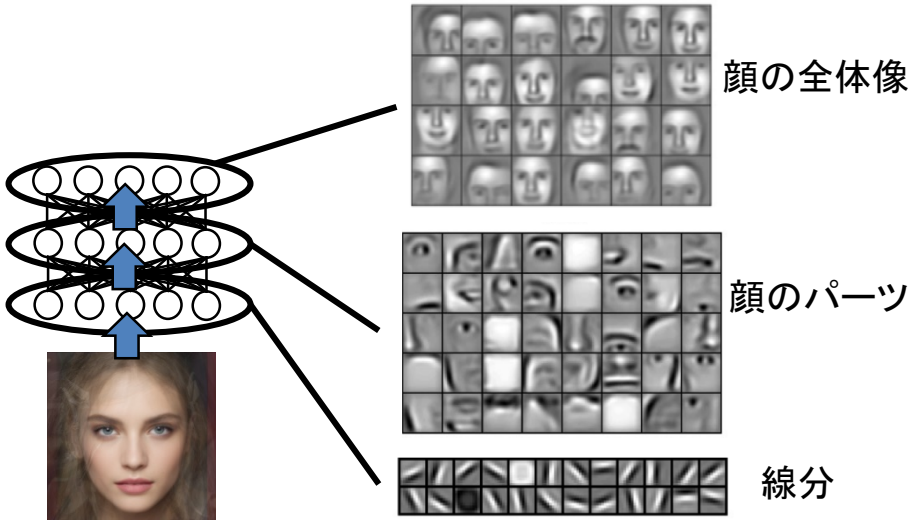
深層学習 (ディープラーニング)

深層学習



- 深層学習 ＝　多層ニューラルネットワーク(NN)による学習
  - ニューラルネットワーク
    - 人間の脳の神経細胞（ニューロン)の仕組みを模した計算モデル
      - パーセプトロン (Rosenblatt 1958)
      - 畳み込みNN (福島 1980)(LeCun+ 1989)
  - 特徴抽出、表現学習

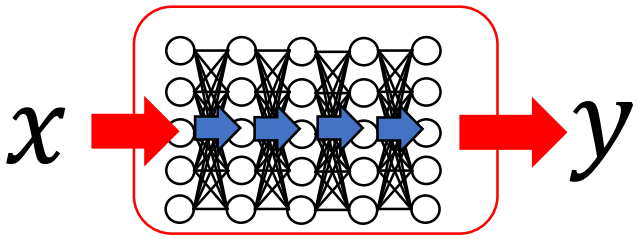
深層学習では入力の特徴が自動的に学習されることがわかってきた

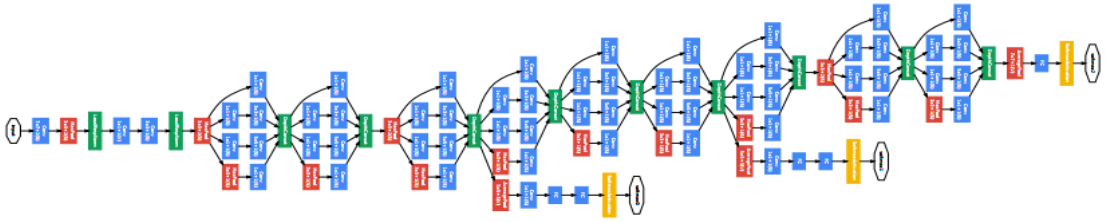


Honglak Lee, Roger Grosse, Rajesh Ranganath, Andrew Y. Ng (2011) Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks, **Communications of the ACM, Vol. 54 No. 10, Pages 95-103 より引用**

深層学習

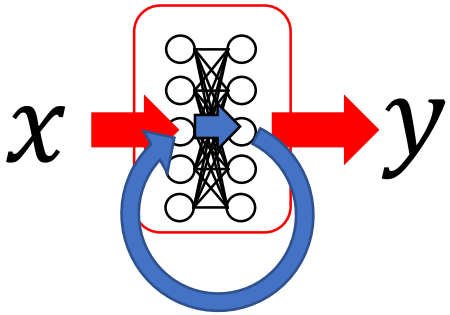
- 関数：重み付き線形和と非線形変換を多層化したもの
  - フィードフォワードニューラルネットワーク

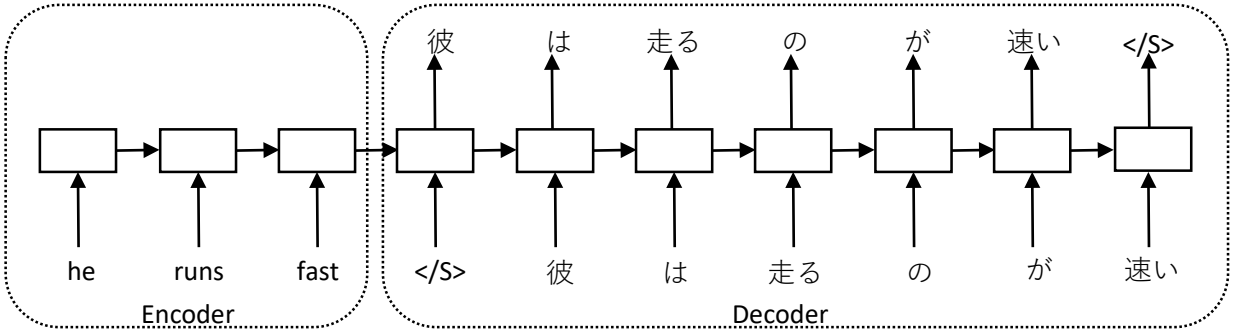




GoogLeNet (22層)　※画像分類に有効なネットワーク

  - 再帰型ニューラルネットワーク



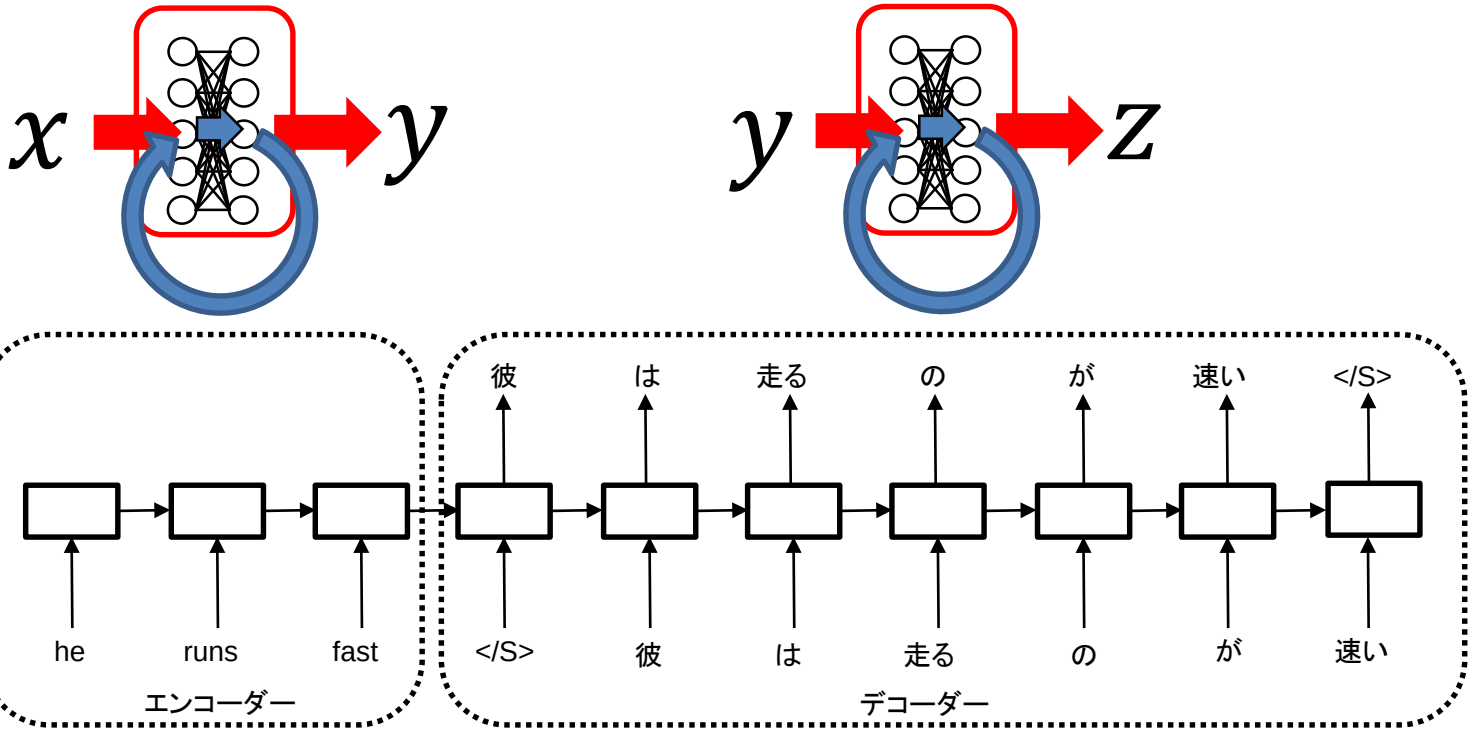


Encoder-Decoderによる機械翻訳

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich (2014) Going Deeper with Convolutions, CVPR 2015

ニューラル機械翻訳

- 2つの再帰型ニューラルネットワークを用いた機械翻訳
  - エンコーダー: 入力文(英語)を中間表現(数百次元のベクトル)に変換
  - デコーダー: 中間表現から出力文(日本語)に変換



深層学習

- 機械学習、深層学習が脚光を浴びるようになった理由
  1. 学習データの大規模化
  2. 高精度化
  3. 高速化

- 現在の深層学習は、他の機械学習手法に比べて、素人でも扱いやすく(End-to-End)、性能も良い

学習データの大規模化

- 大昔(20年以上前)
  - パラメータは手で調整、学習はしない
  - 学習データなし(専門家が作り込んだ少量(数千)の高品質な評価用データのみ)
- 昔(20年前～10年前)
  - 専門家が作り込んだ少量(数万～数十万規模)の高品質データ
  - 大量(数百万、数億規模)の生データ
- 最近
  - 素人が作った大量(数百万、数億規模)の中品質データ
    - クラウドソーシング(Amazon Mechanical Turk)など
  - 自動的に収集できる大量のデータ
    - 特許やEU議事録などの翻訳データ

深層学習のキーテクノロジー：高精度化

- 解きたい仕事の特徴にあったNN
  - 画像認識には畳み込みニューラルネットワーク(CNN)
  - 系列データ(テキストなど)には注意型長・短期記憶(Attention-based LSTM)やTransformer
- 事前学習
  - Stacked Auto Encoder 多層NNをいきなりに学習するのではなく、入力に近い層から順に1層ずつ学習していく手法
  - BERT マスク付き言語モデル
- 正則化(Weight Decay, Dropout)
- 正規化(Batch Normalization)
- 種々の活性化関数(ReLU, Maxout)
- Residual Net
- アンサンブル

深層学習の特長：高精度

- ILSVRC (ImageNet Large Scale Visual Recognition Challenge)における画像分類の精度
  - 2010年 71.8% (NEC Labs America, Univ. of Illinois at Urbana-Champaign, Rutgers Univ.)
  - 2011年 74.2% (Xerox Research Center Europe, CIII)
  - 2012年 83.6% (Univ. of Toronto) … この年に深層学習が圧勝
  - 2013年 88.3% (Clarifai)
  - 2014年 93.3% (Google)
  - 2015年 96.4% (MSRA) … 人間の分類精度(95%)を超えたと言われる
  - 2016年 97.0% (公安調査庁第3研究所, 中国)
  - 2017年 97.75% (Momenta, Univ. of Oxford)

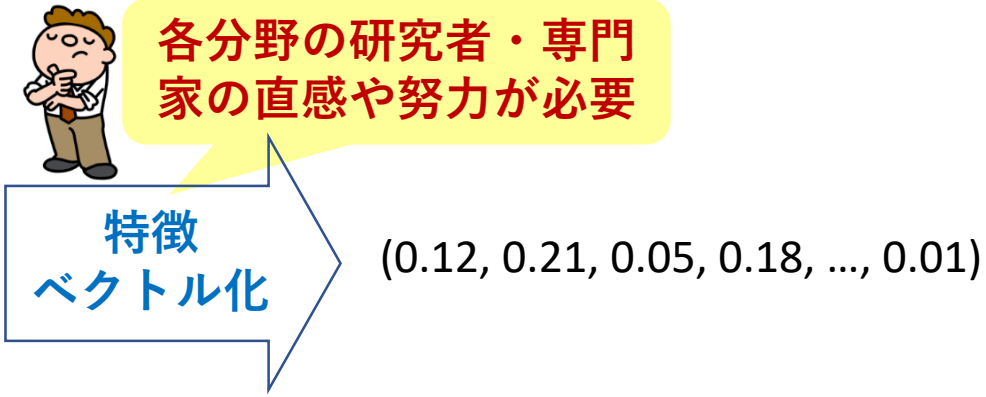
深層学習のキーテクノロジー：高速化

- 学習アルゴリズム
  - 計算グラフによる誤差逆伝搬法
  - オンライン学習(確率的勾配降下法、モーメンタム、AdaGrad、Adam)

- GPU (CPU1コアよりも10倍以上速い)
  - NVIDIA GeForce RTX 3080 (約10万円)
  - NVIDIA Tesla V100 32GB (約120万円)

深層学習の特長：End-to-End (1/2)

- 深層学習以前
  - 入力に対して、様々な処理を加えてから学習を行っていた
    - 画像認識: 画像(ビットマップ)→特徴量(SIFT, SURF, HOGなど)抽出
    - 機械翻訳: テキスト→単語列→品詞解析→構文解析
  - どのような特徴を抽出、利用するかは人手で決めていた



深層学習の特長：End-to-End (2/2)

- 深層学習
  - 解きたい仕事の入力（end）と出力（end）だけをNNに与えて全てまとめて学習する
    - 特徴もデータから自動的に学習
    - 複数のシステムをつなぐパイプライン処理を（あまり）しなくてよい

    例：日本語 → ニューラル機械翻訳 → 英語文

  - 開発がものすごく楽になった
    - 今まで敷居の高かった領域の研究がやりやすくなった

  - 専門家の技術が不要になりつつある

まとめ

- 本講座では、講義形式で深層学習の理論を学習して、実習形式でPC上に深層学習器を作成することで技術を学ぶ

- 深層学習
  - 機械学習の一種
  - 高精度
  - End-to-Endで学習可能

セットアップ編

深層学習のための開発・実行環境

- OS
  - Linux, Windows, Mac
  - Linuxの利用が多い
- **手軽に実行するためのローカル環境**
  - Windows, MacOS, Linux
- **本気で実行したいときはサーバー環境**
  - Windows/Mac (クライアント)
  - Linuxサーバー (CentOS/Ubuntu)
    - CentOSは設定がだんだんと面倒になってきているので、Ubuntuのほうが楽

環境設定

- 環境設定(Windows編)
  - Window PCを使っている人はこの環境設定を行う
  - Anaconda + PyTorch
- 環境設定(MacOS編)
  - Mac PCを使っている人はこの環境設定を行う
  - Anaconda + PyTorch
- 環境設定(Linux編)
  - Linux PCを使っている人はこの環境設定を行う
  - Anaconda + PyTorch
- 環境設定(Windows+Ubuntu編)
  - サーバー環境に近い環境で運用したい人はこの環境設定を行う
  - Python + PyTorch
- 環境設定(Jetson Nano編)
  - エッジ環境で運用したい人はこの環境設定を行う
  - Python + PyTorch

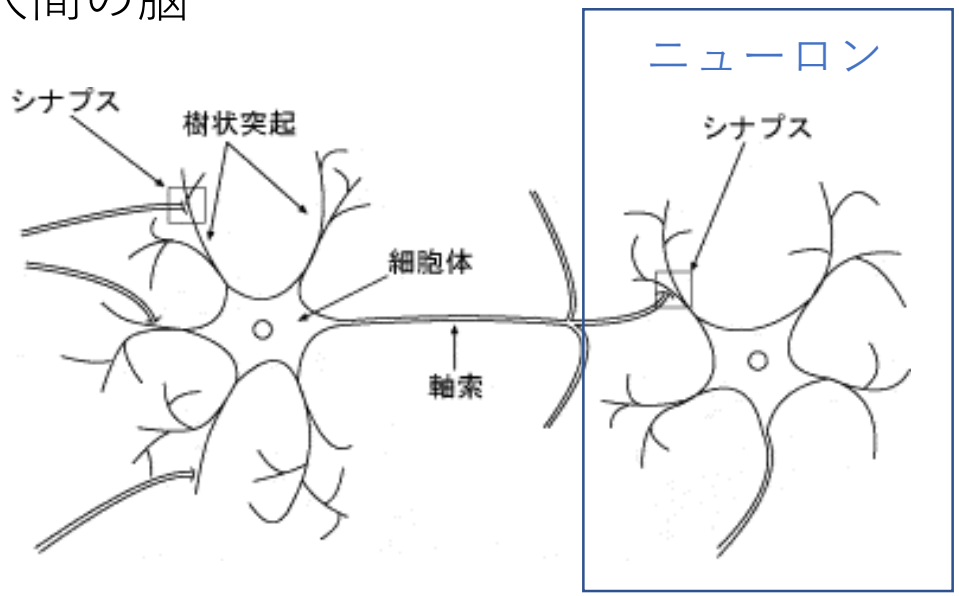
深層学習 (1)

-ニューラルネットワークの仕組み、活性化関数、推論 -

（単純）パーセプトロン (1)

- ニューラルネットワーク（深層学習）の起源
- 人間の脳の神経細胞（ニューロン）の仕組みを模したモデル

人間の脳



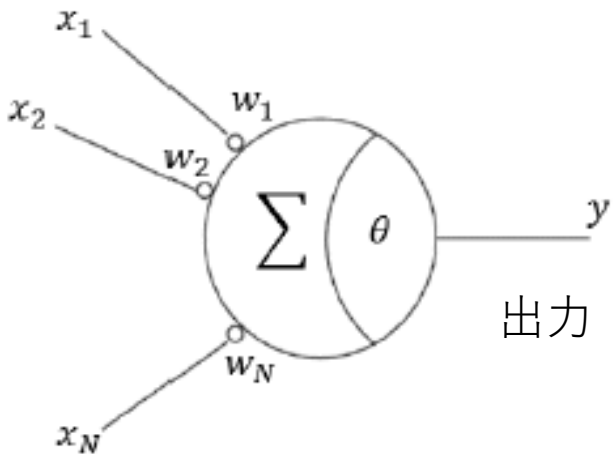
動作：

- シナプスを介して他のニューロンから電気信号を受け取る。
- 電気信号を受け取ったら電位が変化する。
- 電位差が閾値を超えると出力信号を発生する（発火する）。

（単純）パーセプトロン (2)

- ニューロンを模したもの
- 入力信号に対する固有の重みと閾値を持つ
- 動作
  - 多入力を受け付ける
  - 入力の重み付き和を計算する
  - 重み付き和が閾値を超える「1」、そうでなければ「0」を出力する

多入力



入力： $x_i \in \{0,1\}\,(i = 1 \cdots N)$

重み： $w_i \in R (i = 1 \cdots N)$

各入力の重要度をコントロール

閾値：$\theta \in R$

発火のしやすさをコントロール

出力： $y = \begin{cases} 1 & if\ \sum_{i=1}^{N} w_i x_i > \theta \\ 0 & otherwise \end{cases}$

パラメータは w_i と θ

2入力のパーセプトロンでANDゲートを実現してみましょう

ANDゲートの真理値表

パターン	x_1	x_2	y
A	0	0	0
B	1	0	0
C	0	1	0
D	1	1	1

※ 真理値表を満たすパラメータの組み合わせは無限にあります。

(w_1, w_2, θ)=(0.5, 0.5, 0.7) …

真理値表を満たす
パラメータ（w_1、w_2、θ）を設定する

例えば、w_1=1, w_2 = 1とすると…

重み付き和（パターンA）→ 0
重み付き和（パターンB）→ 1
重み付き和（パターンC）→ 1
重み付き和（パターンD）→ 2

$w_1x_1 + w_2x_2$

パターンDの時のみ発火させるように
θを決める
例えば1.5にしてやればよい

(w_1, w_2, θ)=(1.0, 1.0, 1.5)の
パーセプトロンでANDゲートを実現できる

2入力のパーセプトロンでORゲートを実現できるでしょうか？

真理値表を満たすパラメータはある？

ORゲートの真理値表

パターン	x_1	x_2	y
A	0	0	0
B	1	0	1
C	0	1	1
D	1	1	1

例えば、w_1=1, $w_2 = 1$とすると…

重み付き和（パターンA）→0
重み付き和（パターンB）→1
重み付き和（パターンC）→1
重み付き和（パターンD）→2

パターンB,C,Dの時に発火させる
例えばθを0.5にしてやればよい

(w_1, w_2, θ)=(1.0, 1.0, 0.5)の2入力の
パーセプトロンでORゲートを実現できる

2入力のパーセプトロンでNANDゲートを実現できる？

真理値表を満たすパラメータはある？

NANDゲートの真理値表

パターン	x_1	x_2	y
A	0	0	1
B	1	0	1
C	0	1	1
D	1	1	0

パラメータを適切に調整することで様々な動作をさせられる
- 教師データ（xとyのペア）から適切なパラメータを設定することを「学習」という
- 設定されたパラメータを使って入力に対する出力を推定することを「推論（識別、テスト）」という

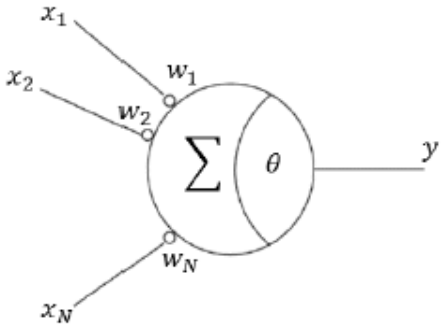
2入力のパーセプトロンでXORゲートを実現できる？

真理値表を満たすパラメータはある？

XORゲートの真理値表

パターン	x_1	x_2	y
A	0	0	0
B	1	0	1
C	0	1	1
D	1	1	0

バイアスの導入



入力： $x_i \in \{0,1\}\ (i = 1 \cdots N)$

重み： $w_i \in R (i = 1 \cdots N)$

各入力の重要度をコントロール

閾値：$\theta \in R$

発火のしやすさをコントロール

出力： $y = \begin{cases} 1 & if\ \sum_{i=1}^{N} w_i x_i > \theta \\ 0 & otherwise \end{cases}$

パラメータは
w_iとθ

(w_1, w_2, θ)=(1.0, 1.0, 0.5)の
パーセプトロンでORゲートを
実現できる

$-\theta$をbと考える

同じこと

重み： $w_i \in R (i = 1 \cdots N)$

各入力の重要度をコントロール

バイアス：$b \in R$

発火のしやすさをコントロール

出力： $y = \begin{cases} 1 & if\ \sum_{i=1}^{N} w_i x_i + b > 0 \\ 0 & otherwise \end{cases}$

パラメータは
w_iとb

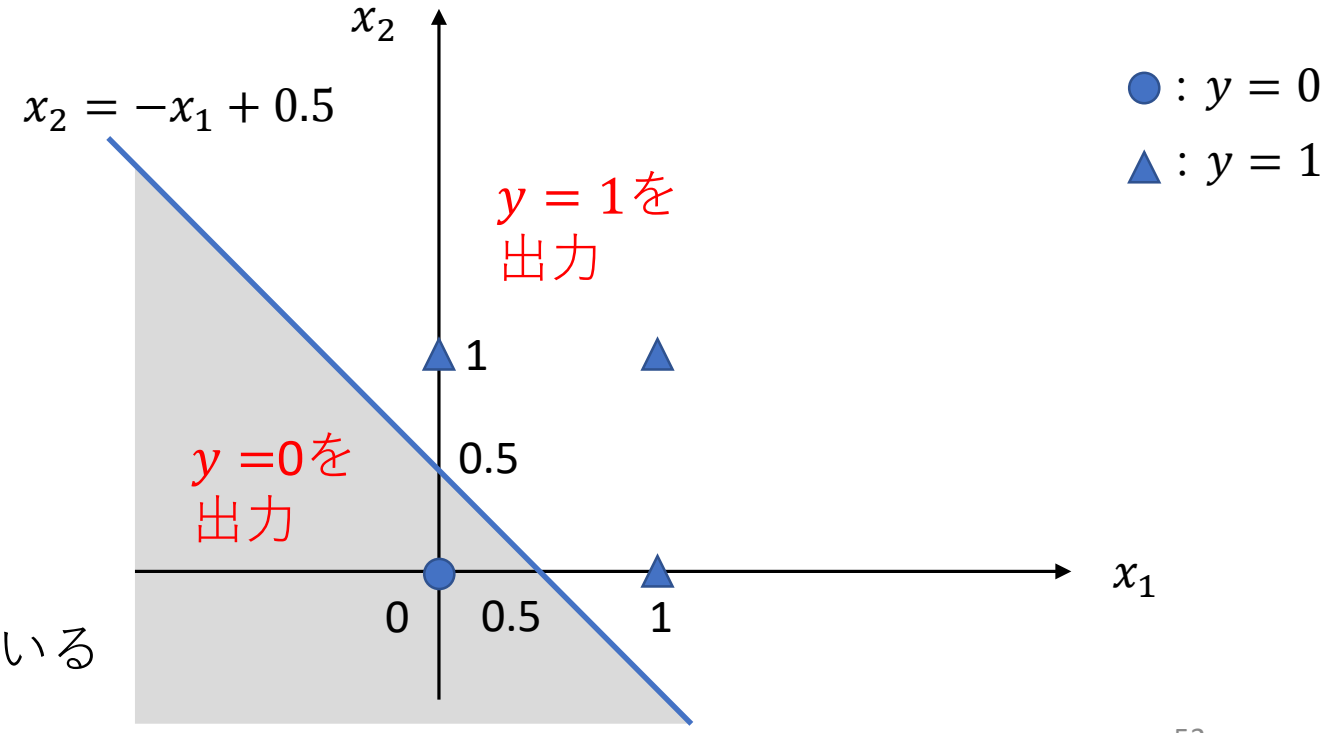
(w_1, w_2, b)=(1.0, 1.0, -0.5)の
パーセプトロンでORゲートを
実現できる

パーセプトロンの動作の視覚的解釈(推定時)

(w_1, w_2, b)=(1.0, 1.0, -0.5)の時

$$y = \begin{cases} 1 & (x_1 + x_2 - 0.5 > 0) \\ 0 & (x_1 + x_2 - 0.5 \leq 0) \end{cases} \Rightarrow y = \begin{cases} 1 & (x_2 > -x_1 + 0.5) \\ 0 & (x_2 \leq -x_1 + 0.5) \end{cases}$$

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

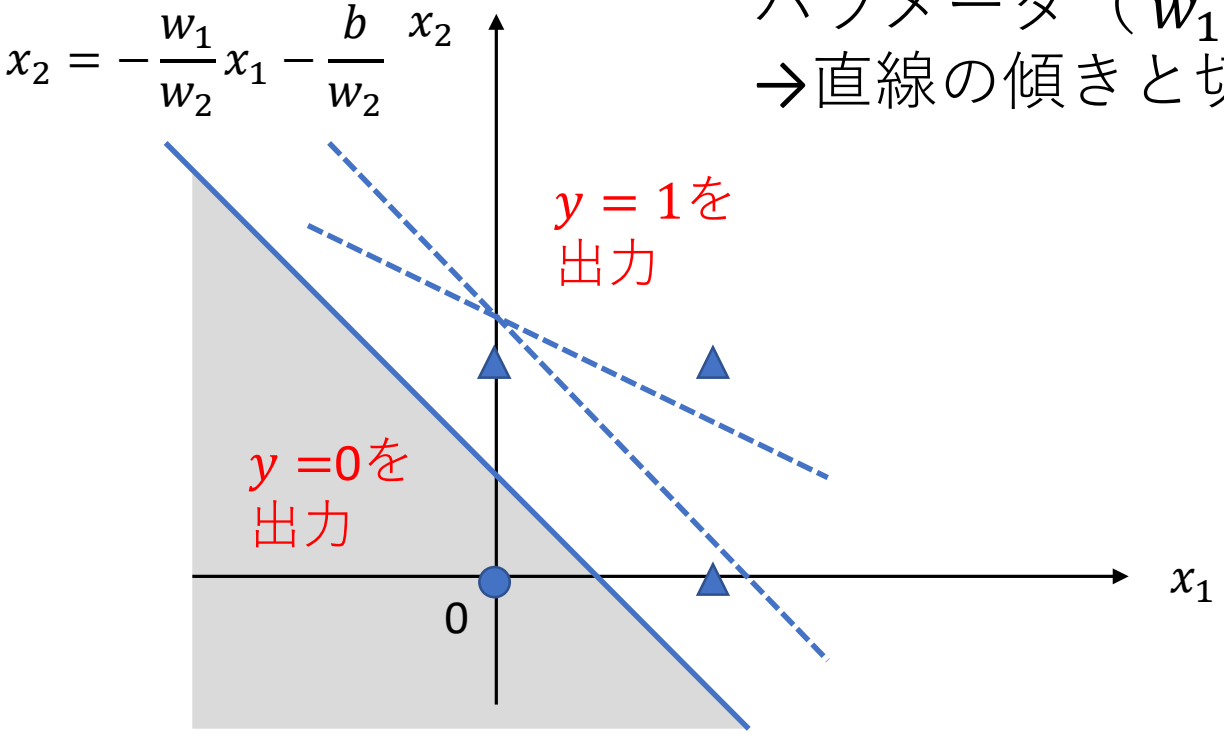


ORゲートを実現している

パーセプトロンの動作の視覚的解釈(学習時)

$$y = \begin{cases} 1 & (w_1 x_1 + w_2 x_2 + b > 0) \\ 0 & (w_1 x_1 + w_2 x_2 + b \leq 0) \end{cases} \Rightarrow y = \begin{cases} 1 & (x_2 > -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}) \\ 0 & (x_2 \leq -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}) \end{cases}$$

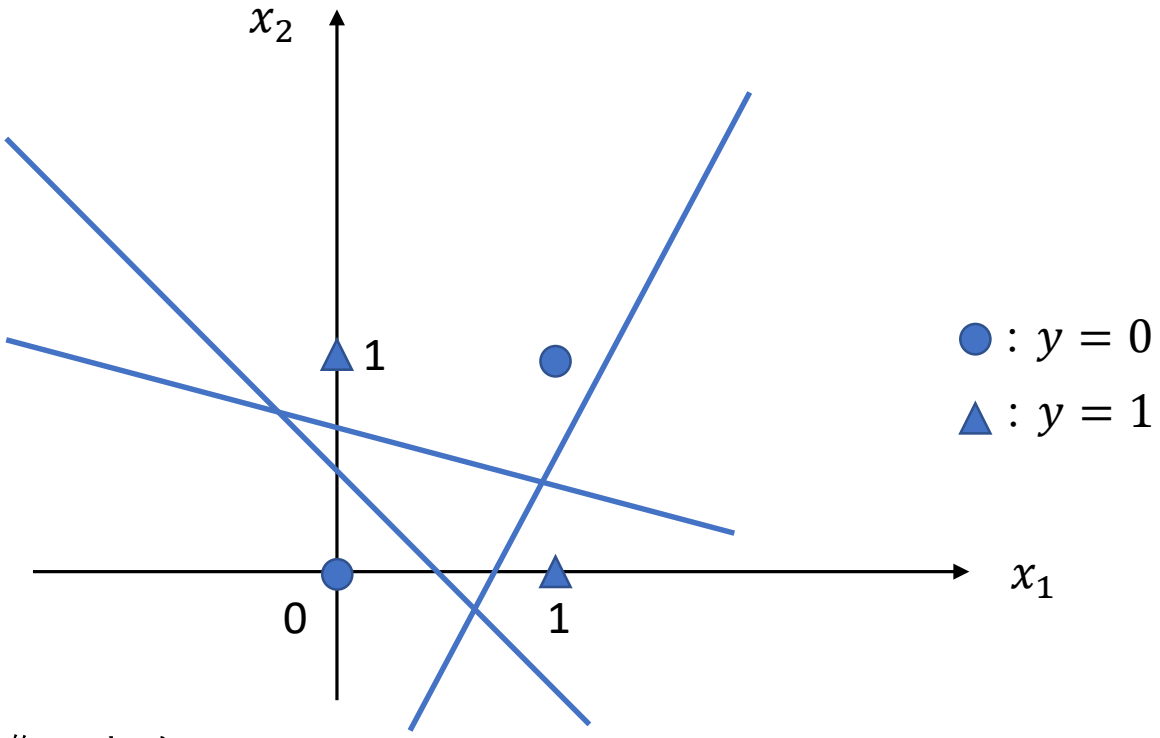
パラメータ（w_1、w_2、b）の調整
→直線の傾きと切片の調整



XORゲートの場合…

XORゲートの真理値表

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



● と▲ は直線で分離できない

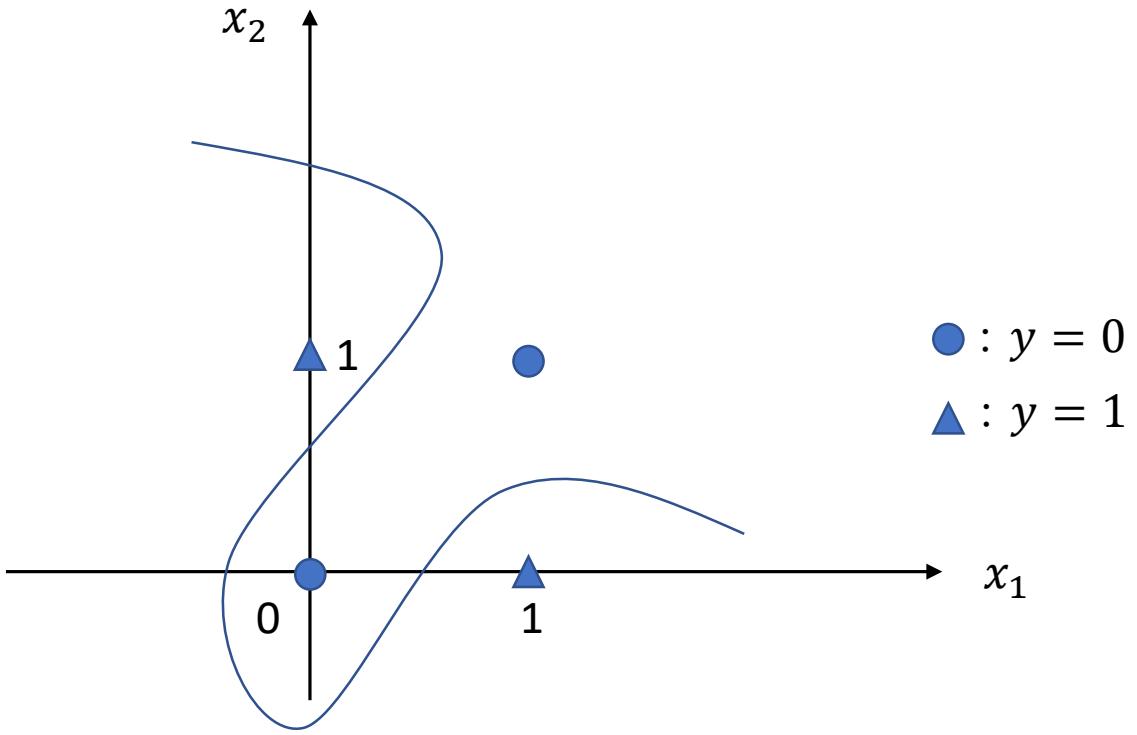
→ 単純パーセプトロンでXORゲートは実現できない

単純パーセプトロンは線形分離可能な問題しかとけない

XORゲートの場合…

XORゲートの真理値表

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



ちなみに、

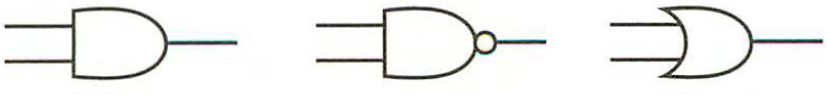
● と ▲ は曲線であれば分離できる

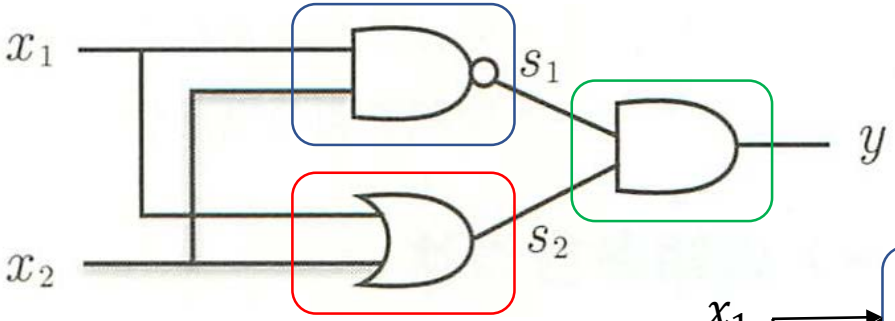
実は、

単純パーセプトロンでも層を重ねると非線形な分離が可能になる

単純パーセプトロンの多層化によるXORゲートの実現

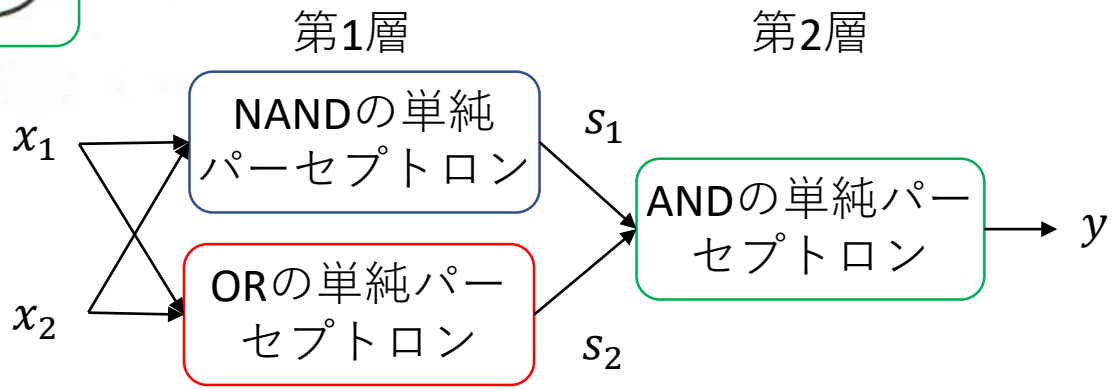
- XORゲートはAND、NAND、ORゲートの組み合わせで実現できる





x_1	x_2	s_1	s_2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

AND
NAND
OR



2層のパーセプトロン（多層パーセプトロン）

パーセプトロンのまとめ

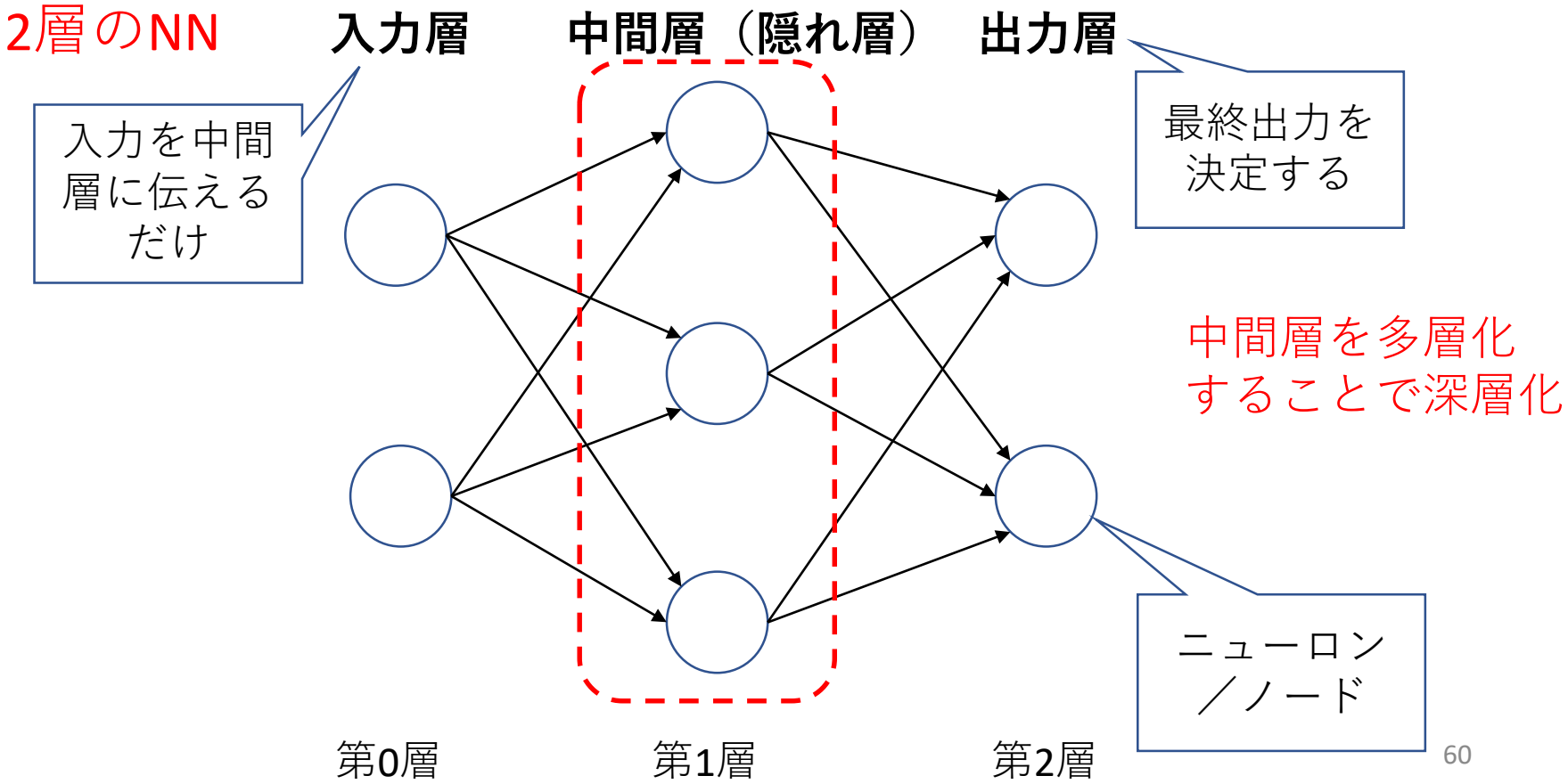
- 人間の神経細胞の仕組みを模したモデル
- 「重み」と「バイアス」のパラメータを持つ
- 入力、重み、バイアスに基づき出力を計算
- 単層パーセプトロンはパラメータを適切に設定することで線形分離可能な問題を表現できる（線形分離不可能な問題は対応不可能）
- 単層パーセプトロンを多層化することで非線形な問題にも対応できるようになる

ニューラルネットワーク（NN）

- パーセプトロンを発展させたもの
- 大きな違いは、
  - 中間層がある（多層）
  - 活性化関数が導入されている

標準的なNNの構造

- 順伝搬型（フィードフォワード）NN：情報が入力側から出力側に一方向に伝わるNN



活性化関数の導入

- パーセプトロン

入力： $x_i \in \{0,1\}\,(i = 1 \cdots N)$
パラメータ： $w_i, b \in R (i = 1 \cdots N)$

出力： $$y = \begin{cases} 1 & if \ \sum_{i=1}^{N} w_i x_i + b > 0 \\ 0 & otherwise \end{cases}$$

$$a = \sum_{i=1}^{N} w_i x_i + b$$
$$y = h(a)$$
$$h(x) = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

ステップ関数

入力信号の総和を出力信号に変換する関数$h(x)$を
活性化関数と呼ぶ

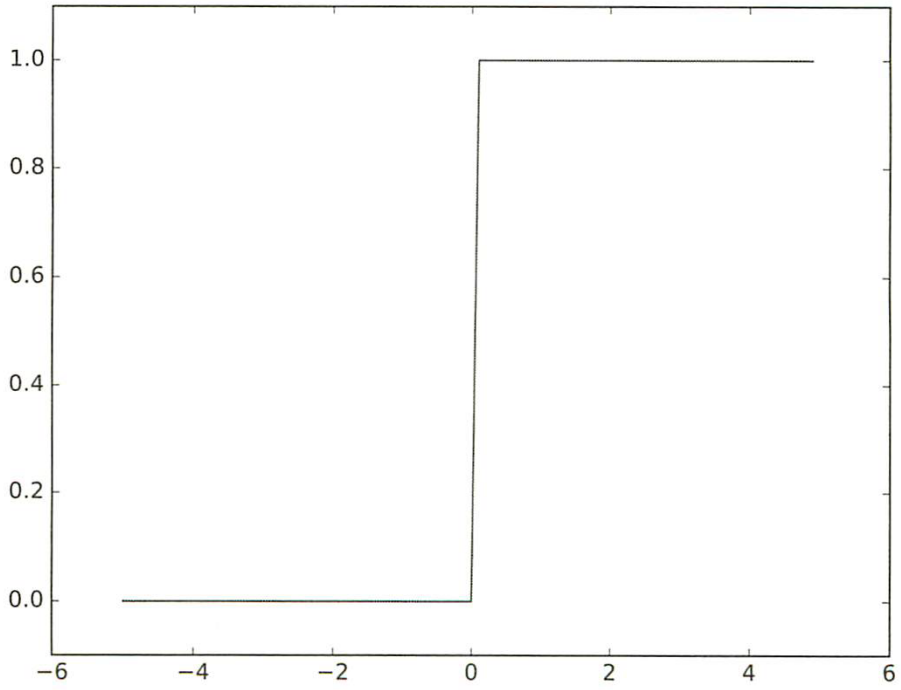
入力信号の総和がどのように発火（活性化）するかを決定する役割

NNの各ノードでは活性化関数としてステップ関数以外の関数も使う！

活性化関数：ステップ関数

パーセプトロンの活性化関数

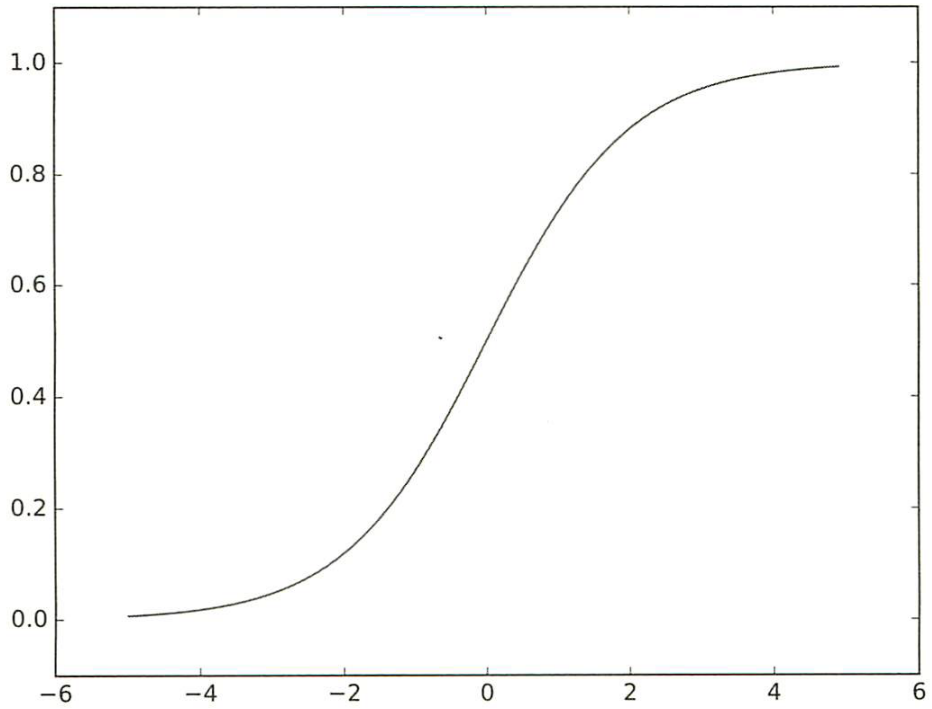
$$h(x) = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$



活性化関数：シグモイド関数

$$h(x) = \frac{1}{1 + \exp(-x)}$$

e^{-x}



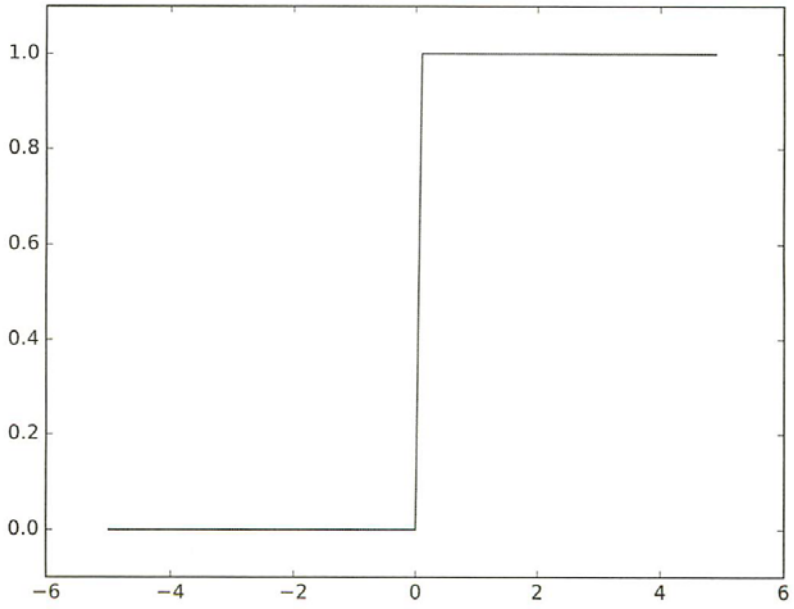
ステップ関数をスムーズにした形

ステップ関数とシグモイド関数の相違点

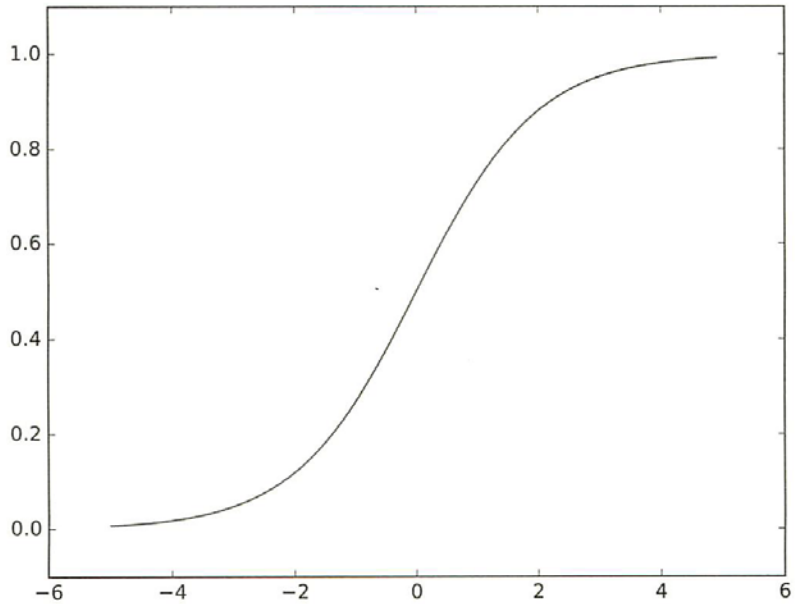
- ステップ関数は0か1の値、シグモイド関数は実数を出力

→ パーセプトロンでは0か1の信号、NNでは連続的な実数値の信号が流れる

ステップ関数

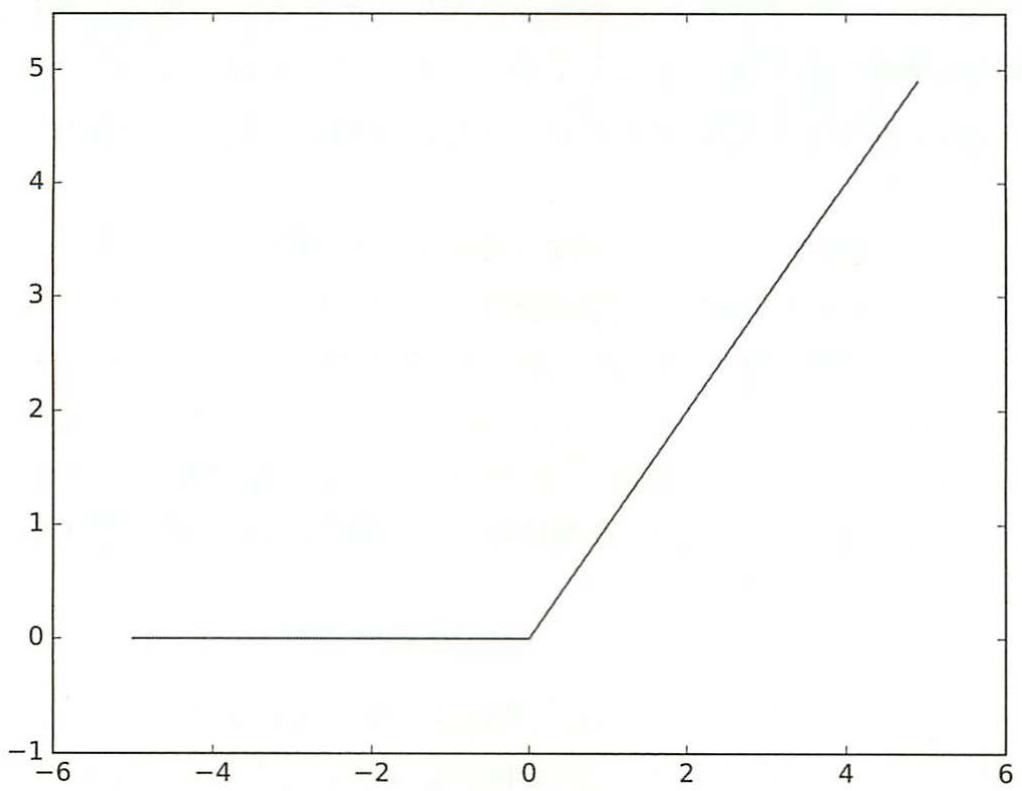


シグモイド関数



活性化関数：ReLU（Rectified Linear Unit）関数

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$



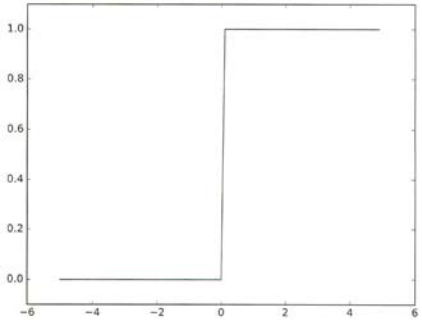
隠れ層の活性化関数の共通点

- 非線形関数

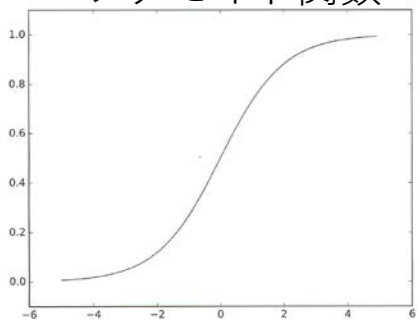
NNの隠れ層は非線形関数を活性化関数として用いないと深層化する意味がない！

仮に、線形関数$h(x) = cx$を活性化関数とすると、3層隠れ層を重ねた場合、その出力は、$y(x) = h\Big(h\big(h(x)\big)\Big) = c^3x$となる。
これは、$h(x) = ax$（ただし、$a = c^3$）を活性化関数とした1層の層と同等で、多層にするメリットなし

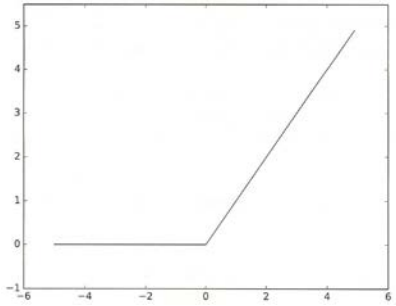
ステップ関数



シグモイド関数



ReLU関数



出力層の活性化関数

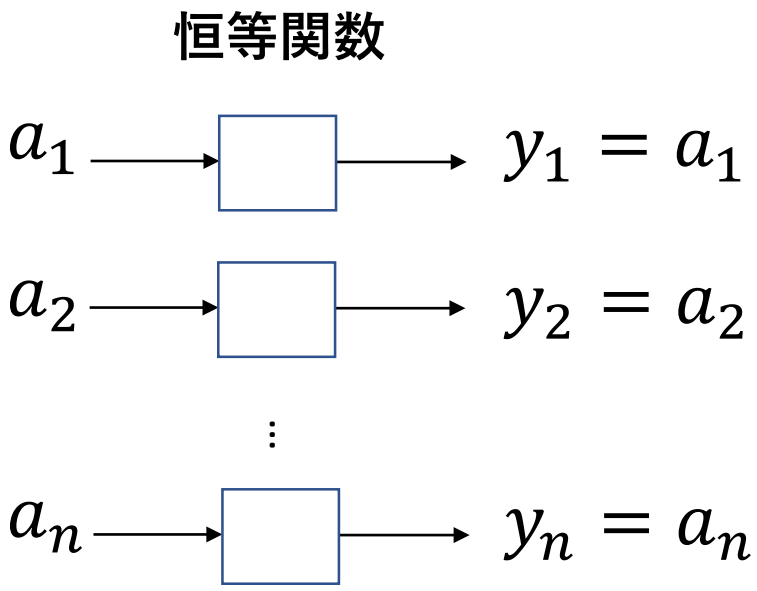
機械学習が扱う問題

- 分類問題：入力データを分類する問題
  - ※分類先のクラス数は予め決まっている
  - 例：人の画像からその人が男性か女性に分類
- 回帰問題：入力データから連続的な数値を予測する問題
  - 例：人の画像からその人の体重を予測

回帰問題 → 恒等関数
分類問題 → ソフトマックス関数

回帰問題の活性化関数

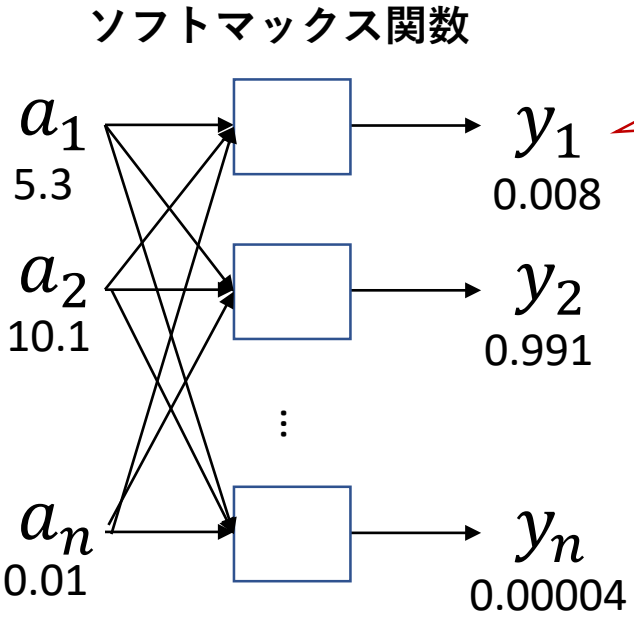
- 恒等関数
  - 入力をそのまま出力



分類問題の活性化関数

- ソフトマックス関数
  - 自分以外のノードの総和aの影響を受ける

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$



$$0 \leq y_k \leq 1 \ (1 \leq k \leq n)$$
$$\sum_{k=1}^{n} y_k = 1$$

出力層のノード数を「分類クラス数」にする
→ 各ノードの出力はそのクラスの分類確率として解釈可能
→ 分類確率の最も高いクラスに分類

本講義でのノードの書き方

入力： $x_i \in \{0,1\}\ (i = 1 \cdots N)$

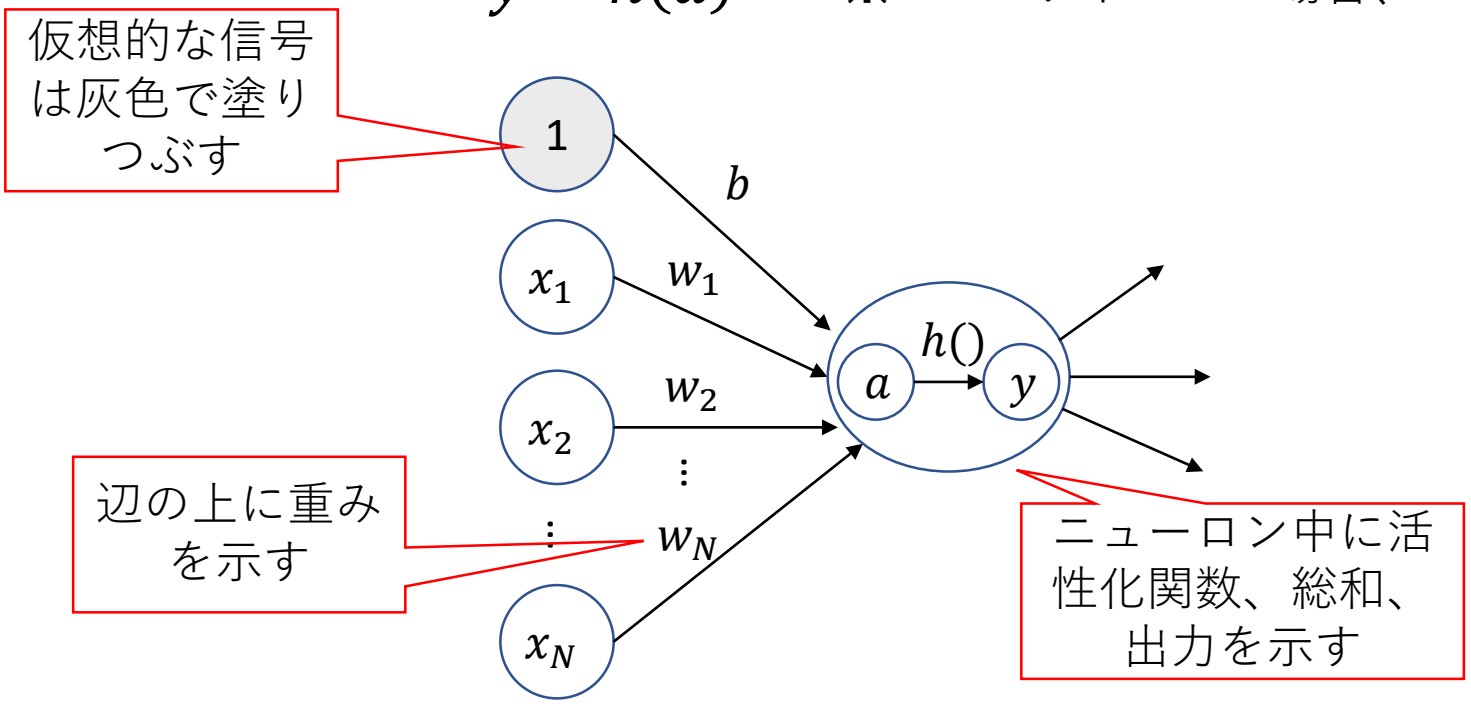
パラメータ： $w_i, b \in R (i = 1 \cdots N)$

出力：

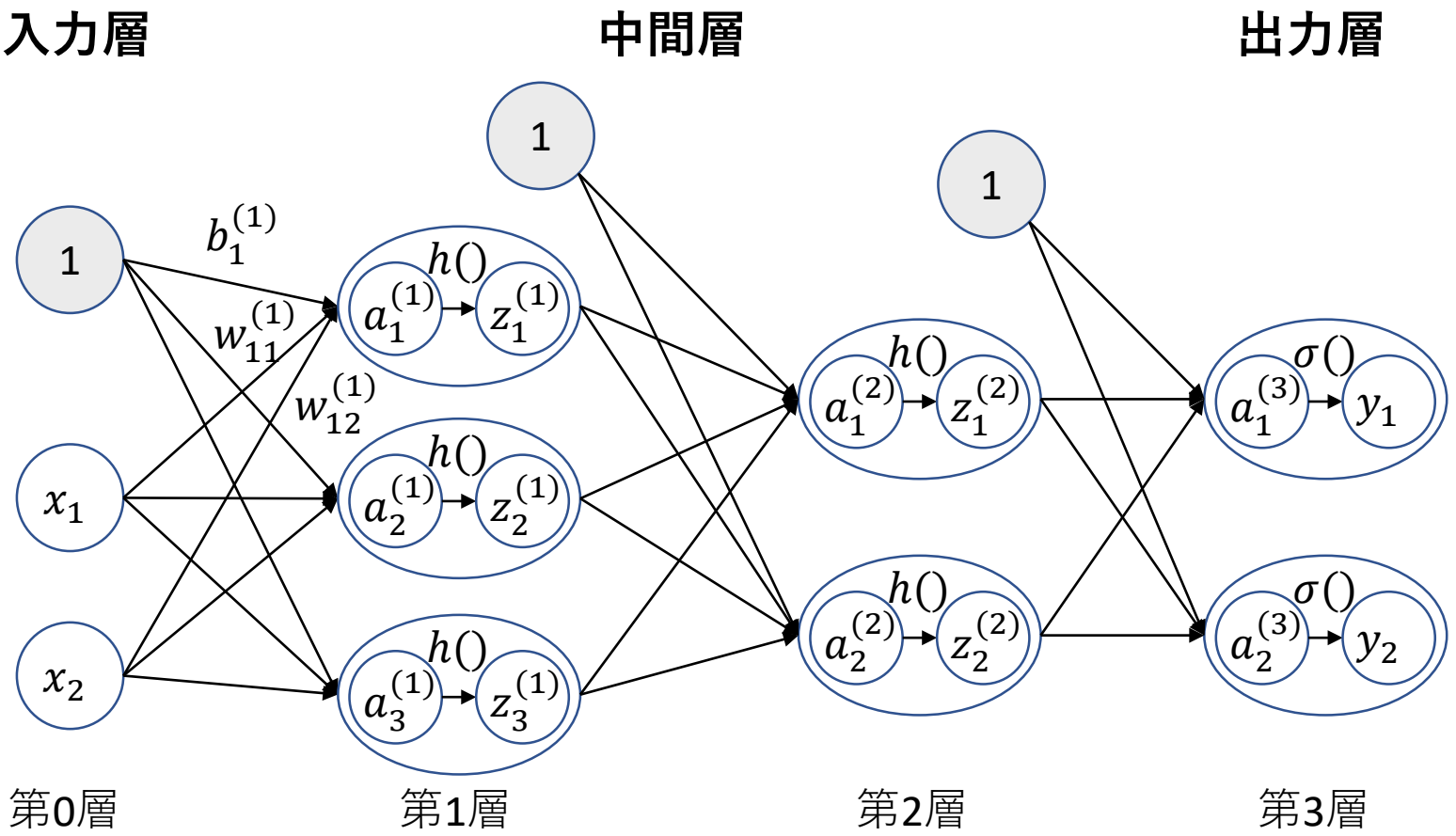
$$a = \sum_{i=1}^{N} w_i x_i + b$$

$$y = h(a)$$

※パーセプトロンの場合、ステップ関数



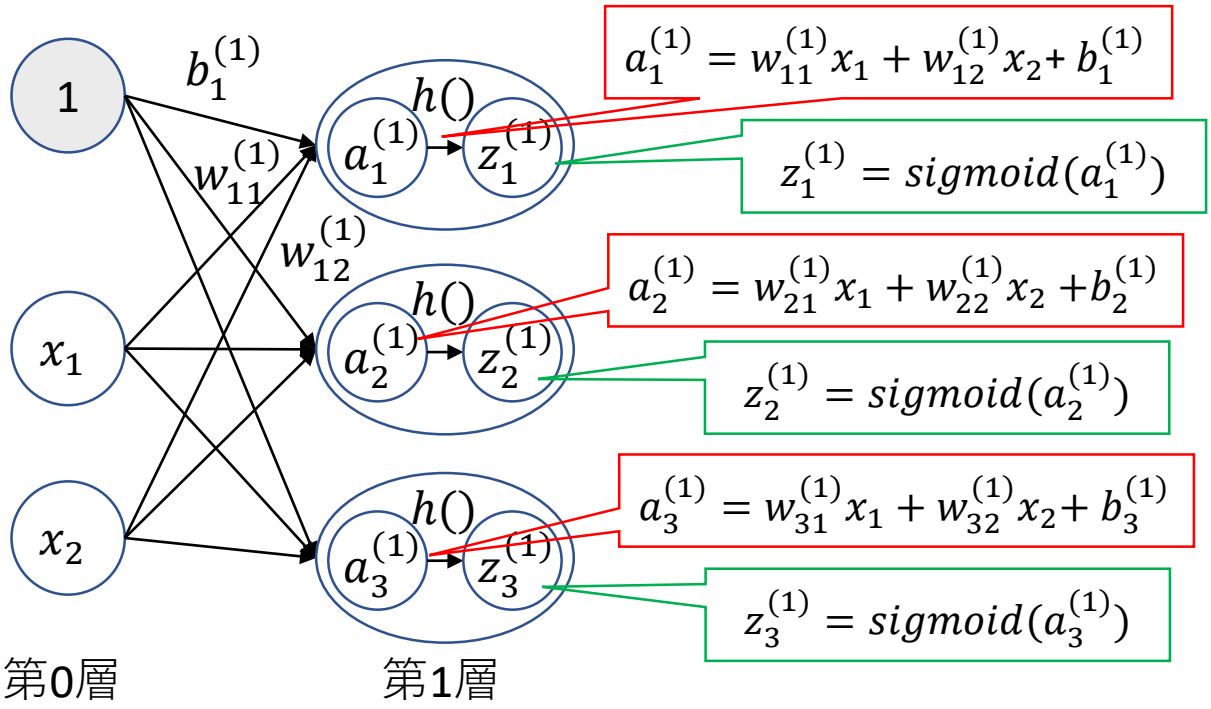
3層NNの推論



$h()$：シグモイド関数、$\sigma()$：恒等関数

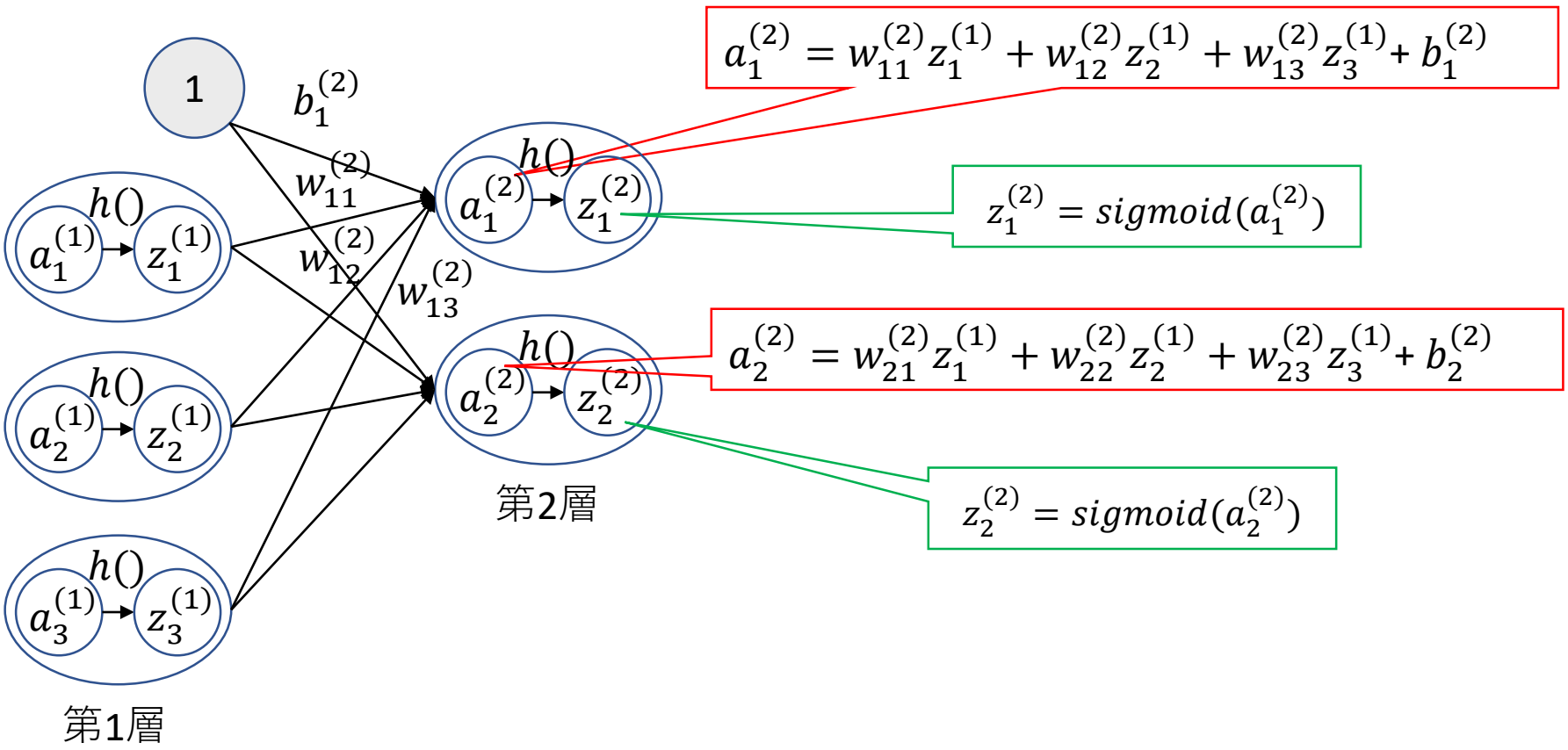
信号は低層から順に伝わる
→低層の演算から順に行う

1層目の動作



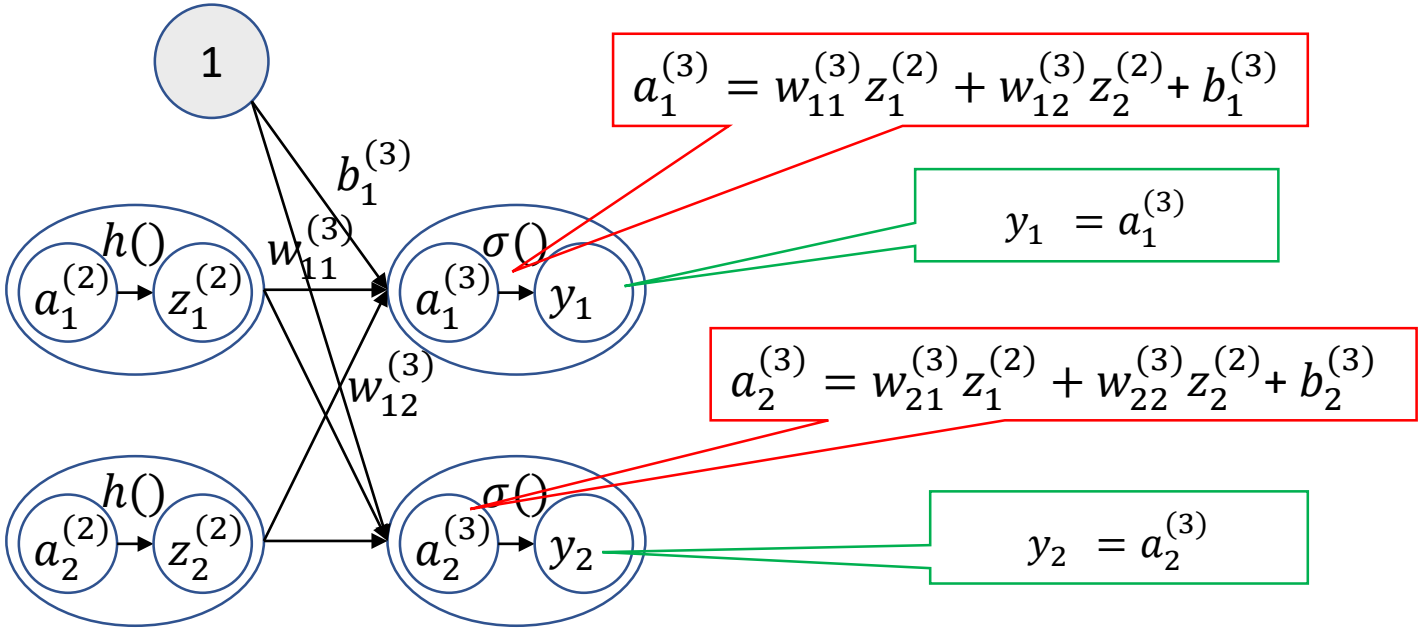
$h()$：シグモイド関数

2層目の動作



$h()$：シグモイド関数

3層目（出力層）の動作



$\sigma()$：恒等関数

NN推論のまとめ

- NNは入力層、中間層、出力層の多層構造であり、信号が階層的に伝わる
- 中間層の活性化関数としてシグモイド関数やReLU関数のような滑らかに変化する関数を利用する
- 出力層の活性化関数は、回帰問題では恒等関数、分類問題ではソフトマックス関数を利用する

実習編: Python

Pythonチュートリアル

PYTHON TUTORIAL

Python はじめに

- **Python https://www.python.jp/**
  - 簡単にプログラムを書きたいときのための軽量言語の一種
  - 関数型言語をベースにしているため、しっかりとした言語設計になっている
  - 基本的にインタプリタなので、プログラム開発が容易
  - 最もよく使われているプログラミング言語の一つ
    - 多くのプログラミング言語ランキングで1位～2位
  - ライブラリが充実しており、データ処理、データ解析、機械学習のためによく使われる言語になっている。特に深層学習のプログラムはPythonで書くことが多い。
  - ドキュメント https://docs.python.jp/3/

※Pythonには2系と3系があり、構文や同じ関数でも処理が違う場合があるので注意。本講義ではPython 3系を想定。

Python環境設定

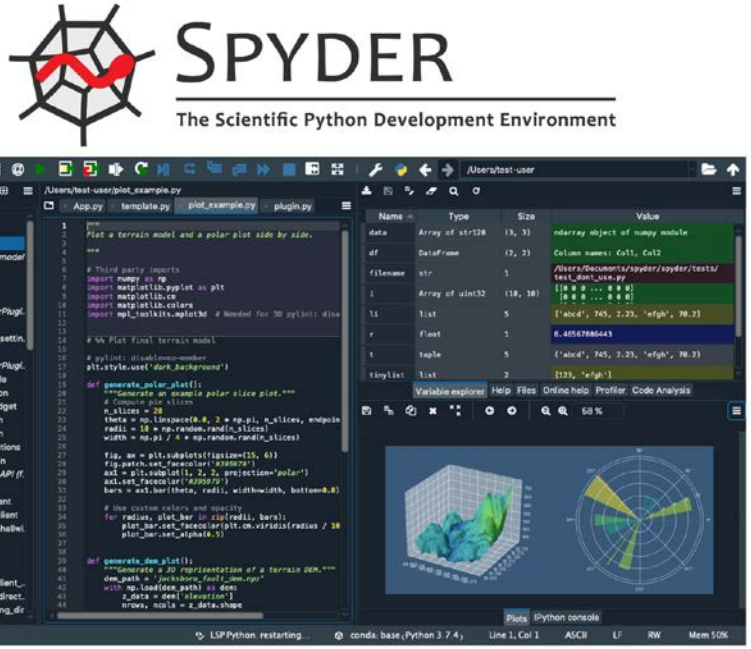
- 深層学習(ディープラーニング)を行うには？
  - クラウド環境を使う方法
    - Google Colaboratory (略称 Colab)
      - https://colab.research.google.com/
      - Googleが提供するクラウド環境の深層学習プラットフォーム
      - Googleアカウントがあれば無料で制限付きながら使える
      - Googleアカウントをもっていればおすすめ
    - Microsoft Azure Machine Learning (通称 Azure)
      - Microsoftアカウント+サブスクライブで使える。
      - 無料で1年間だけ制限付きながら使える。

Python環境設定

- 深層学習(ディープラーニング)を行うには？
  - 自分が持っているPCを使う方法
    - Pythonを使う
      - サーバー環境では、Python+PyPIがよく使われる。いろいろなものをインストールする必要があったり、バージョン不整合があったりして結構大変
    - Anacondaを使う
      - Python + 科学技術計算(データサイエンス)ライブラリ
      - データサイエンス向けライブラリが全部入りで最初から入っている
      - 最初から入っているライブラリ
        - NumPy ベクトル行列演算ライブラリ
        - SciPy 数値演算、最適化、信号処理、関数
        - Pandas 統計処理、データ分析、時系列解析
        - scikit-learn 機械学習

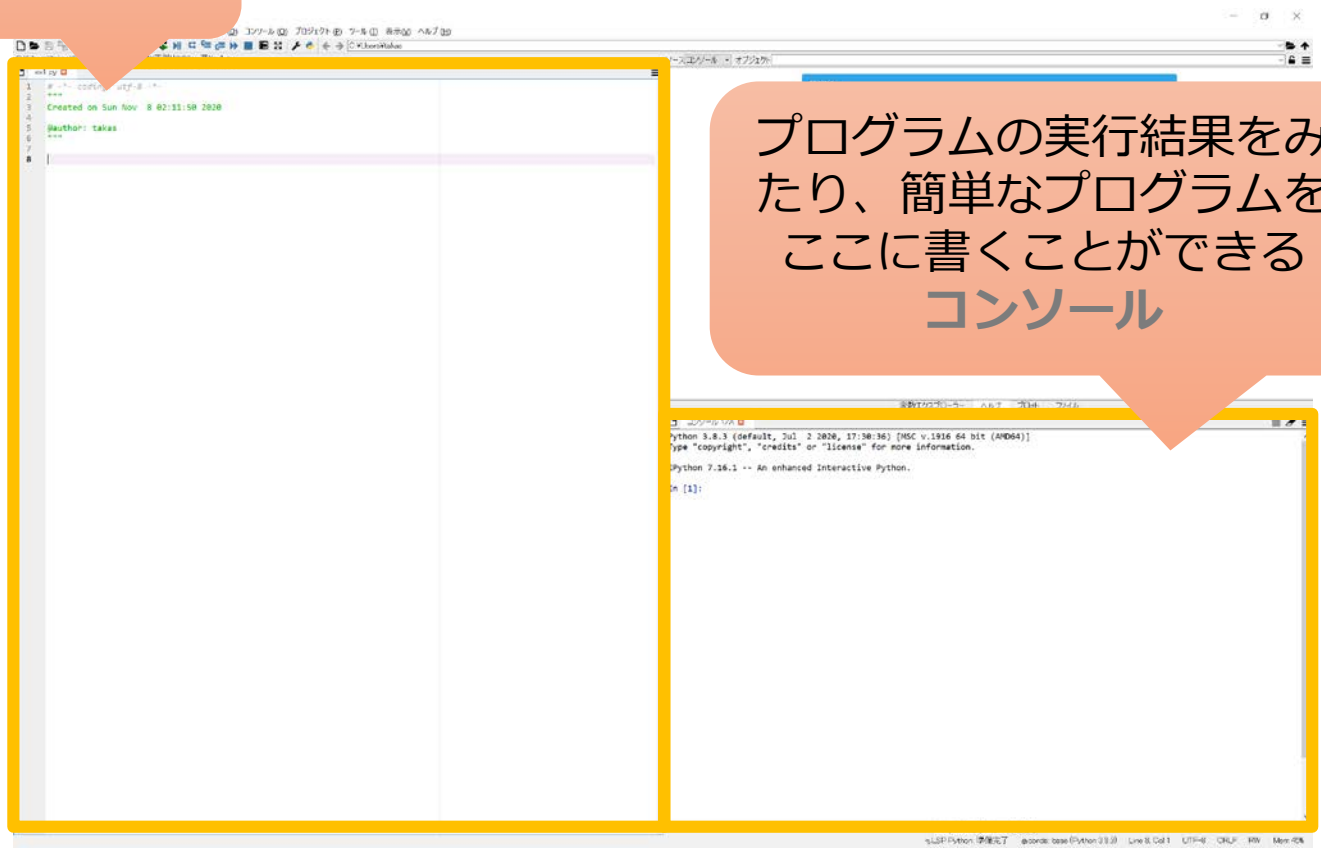
Spyder

- SpyderはPythonの統合開発環境
  - SpyderでPythonのプログラムを書いたり、プログラムを実行することができます
  - Anacondaをインストールした場合は、Spyderもついてくるのでこれを使うのが便利



Spyder

プログラムを書くための**エディタ**



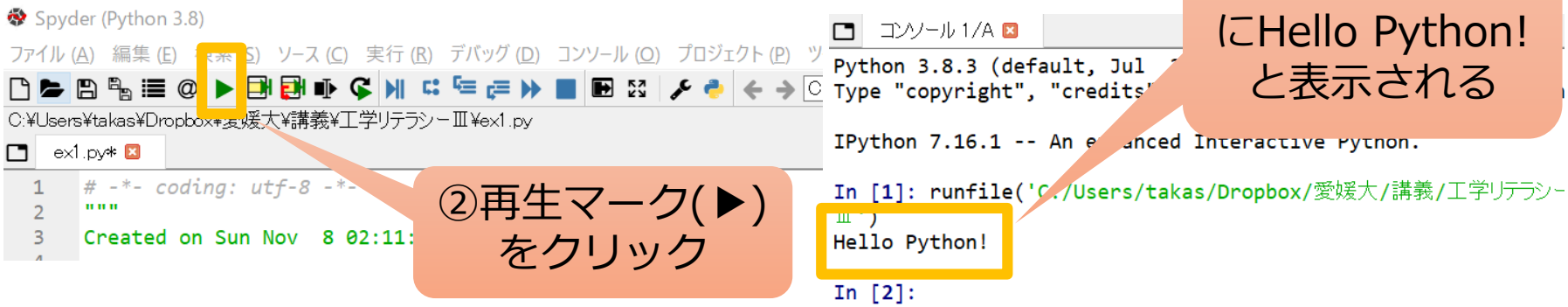
プログラムの実行結果をみたり、簡単なプログラムをここに書くことができる**コンソール**

Spyder

- プログラムの実行
  - エディタに「print(“Hello Python!”)」と書いて実行してみましょう。

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Nov  8 02:11:50 2020
4
5  @author: takas
6  """
7
8  print("Hello Python!")
9
```

①このように追記



②再生マーク(▶)をクリック

③コンソール部にHello Python!と表示される

VirtualBox+Ubuntuの場合

- テキストエディタでプログラムを書きましょう
  - gedit (初級者向け) ubuntuに最初から入っています
  - visual studio code (上級者向け)
  - emacs (上級者向け) セットアップ編でインストールしました

どれを使ってもかまいません。visual studio codeやemacsを使ったことがない人はgeditを使いましょう。左下のアプリボタンから起動するか、下記のとおり端末から起動することができます。

```
$ gedit &   (geditを起動したいとき)
$ emacs &    (emacsを起動したいとき)
$ code &     (visual studio codeを起動したいとき)
```

※ 後ろに&をつけると、別ウィンドウで起動します

(参考) Visual Studio Code (vscode)のインストール

- Visual Studio Codeのホームページにアクセス
  https://azure.microsoft.com/ja-jp/products/visual-studio-code/
- 今すぐダウンロードを選び、「↓.deb」(ubuntu用)を選択します。
- 「プログラムで開く」を選ぶとインストールが始まります。
- 左下のアプリボタンから「Visual Studio Code」を選択するか、端末から「code」で実行できます。

※Jetson Nanoではこの方法ではインストールできません。(アーキテクチャが違うため)

Python 算術演算

- Pythonでの算術演算
  - 足し算 +
  - 引き算 –
  - 掛け算 *
  - 割り算 /
  - 整数の割り算 //
  - 余りの計算 %
  - べき乗 **
- Spyderを起動して右のプログラムをコンソール(右下のIn [1]:と書かれているところ)で実行してみよう

```
In [1]: 1+3
Out[1]: 4

In [2]: 3-1
Out[2]: 2

In [3]: 5*2
Out[3]: 10

In [4]: 2**4
Out[4]: 16

In [5]: 7/5
Out[5]: 1.4

In [6]: 7//5
Out[6]: 1

In [7]: 7%5
Out[7]: 2
```

※ Ctrl+d, quit(), exit()のいずれかでコンソールを終了して最初から始めることができます

Python：算術演算

- **Ubuntuの端末で実行する場合**

```
$ python3
>>> 1+3
4
>>> 3-1
2
>>> 5*2
10
>>> 2**4
16
>>> 7/5 #浮動小数点
1.4
>>> 7//5
1
>>> 7%5
2
>>> quit()
```

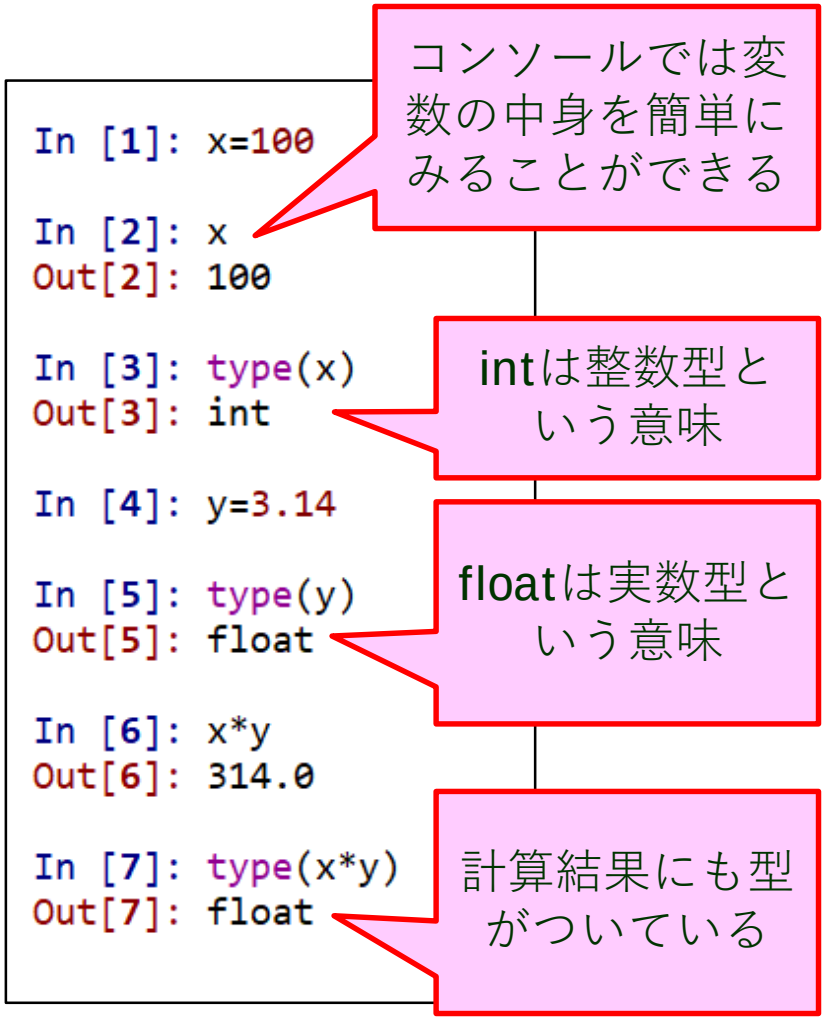
```
$ python3
>>>
```
pythonインタプリタが起動して対話モードでプログラミング可能

```
$ python3 test.py
```
ファイル名を引数にすることでpythonプログラムを実行可能

Ctrl+d、quit()、exit()のいずれかで対話モードを終了します

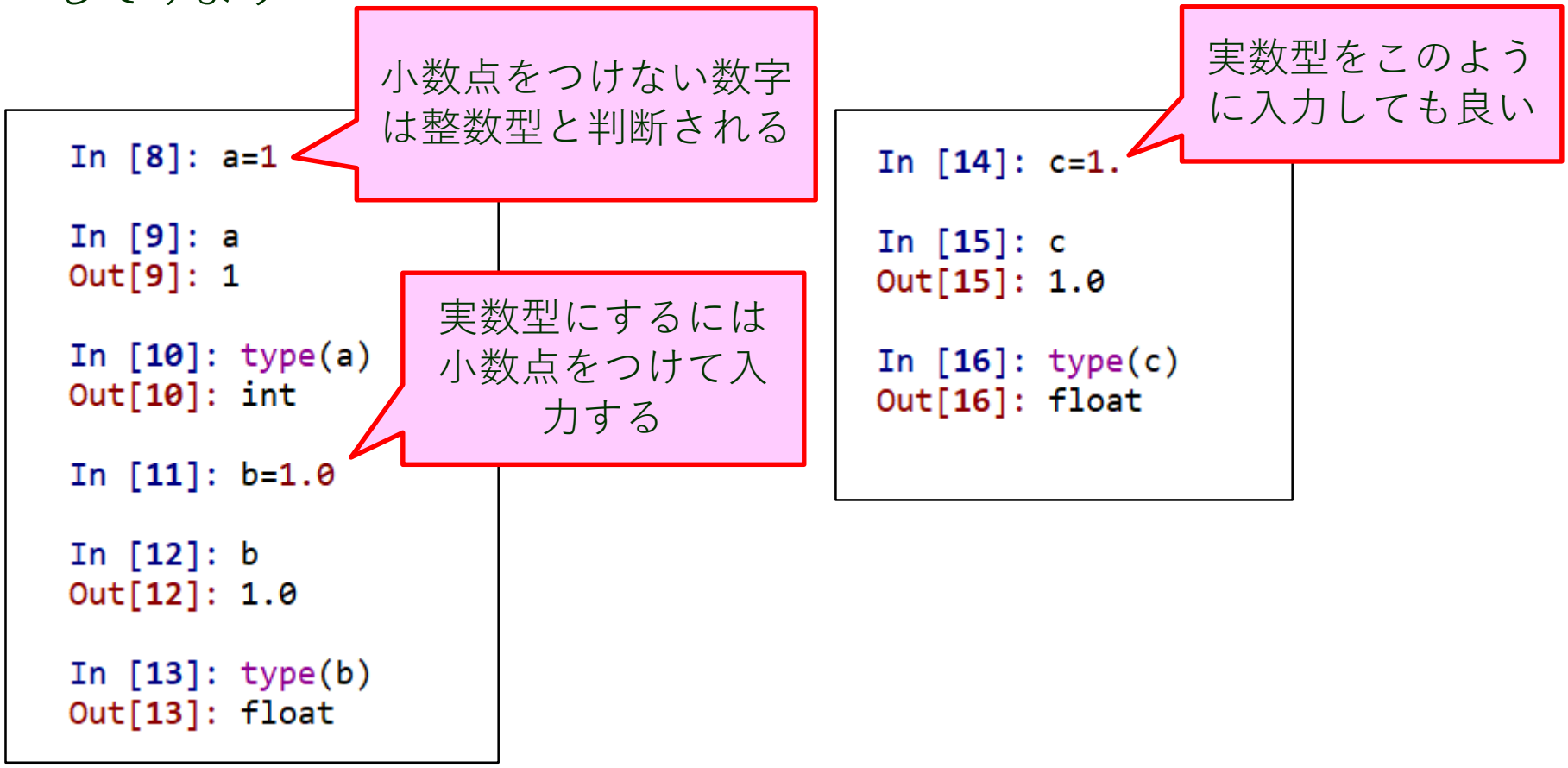
Python 変数

- アルファベットで定義
- 整数型(Int)や実数型(float)といった型は宣言せず自動的に決定される
- 変数への代入は「=」
- 変数の型や計算結果の型はtype関数で調べることができる
- 次のプログラムをコンソールで実行してみよう



Python 変数

- 次のプログラムをコンソールで実行してみよう



ブール型

- ブール型: TrueかFalseの値
- ブール演算
  - and演算: $x\ and\ y =$
    $\begin{cases} True & x\text{とy}\text{の両方とも}True\text{のとき} \\ False & \text{それ以外} \end{cases}$
  - or演算: $x\ or\ y =$
    $\begin{cases} True & x\text{とy}\text{のどちらかもしくは両方が}True\text{のとき} \\ False & \text{それ以外} \end{cases}$
  - not演算: $not\ x = \begin{cases} True & x\text{が}False \\ False & x\text{が}True \end{cases}$ (xのTrue/Falseの反転)

ブール型

- コンソールで次のPythonプログラムを書いてみよう

```
In [1]: hungry = True

In [2]: sleepy = False

In [3]: type(hungry)
Out[3]: bool

In [4]: not hungry
Out[4]: False

In [5]: hungry and sleepy
Out[5]: False

In [6]: hungry or sleepy
Out[6]: True
```

文字列型

- 文字列
  - ’(シングルクォート)で囲む
  - ”(ダブルクォート)で囲む
  - ”””(ダブルクォート3つ)で囲む
  - どの方法でも同じ文字列が得られる
  - 文字列中に「"」を使いたいときは「'」で囲む、といった使い分けをします。

  例: `print('he says "hello!".')`

- 文字列の連結は「+」で実行できる

```
In [1]: a = "こんにちは"

In [2]: b = 'こんにちは'

In [3]: c = """こんにちは"""

In [4]: a
Out[4]: 'こんにちは'

In [5]: b
Out[5]: 'こんにちは'

In [6]: c
Out[6]: 'こんにちは'

In [7]: a+b+c
Out[7]: 'こんにちはこんにちはこんにちは'
```

文字列の連結

リスト型

- 複数のデータをまとめる
- コンマ区切りの値の並びを角括弧で囲む

```
In [1]: a = [1, 2, 3, 4, 5]

In [2]: a
Out[2]: [1, 2, 3, 4, 5]

In [3]: len(a)
Out[3]: 5

In [4]: a[0]
Out[4]: 1

In [5]: a[1]
Out[5]: 2

In [6]: a[4]
Out[6]: 5

In [7]: a[4] = 99

In [8]: a
Out[8]: [1, 2, 3, 4, 99]
```

角括弧の中にリストの各要素を並べる。各要素はカンマ(,)で区切る

リストの長さは**len関数**で計算できる

a[n]と書くことでn番目の要素を得ることができる。ただし、**リストの最初の要素は0番目と数える**

a[n]=xと書くことでn番目の要素にxを代入することができる

リスト型

- リストの連結
  - +でリストを連結することができる (文字列の連結と似ている)

リストの連結

```
In [1]: a = [1, 2, 3]

In [2]: b = [4, 5, 6]

In [3]: a+b
Out[3]: [1, 2, 3, 4, 5, 6]
```

- リストの生成
  - [x]*n でxを要素とする長さnのリストを生成できる

要素1の長さ10のリスト

```
In [1]: a = [1]*10

In [2]: a
Out[2]: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

リスト型

- **スライシング[:]によりサブリストにアクセス可能**

```
>>> a[0:2] #0番目から2番目（2番目は含まない）まで
[1, 2]

>>> a[1:] #1番目から最後まで
[2, 3, 4, 99]

>>> a[:-1] #最初から最後の要素の一つ前まで
[1, 2, 3, 4]

>>> a[:] #全ての要素
[1, 2, 3, 4, 99]
```

(注) a[-1]でaの一番最後の要素にアクセスできる

リスト型

- **リストへの要素の追加**
- **リストの結合**

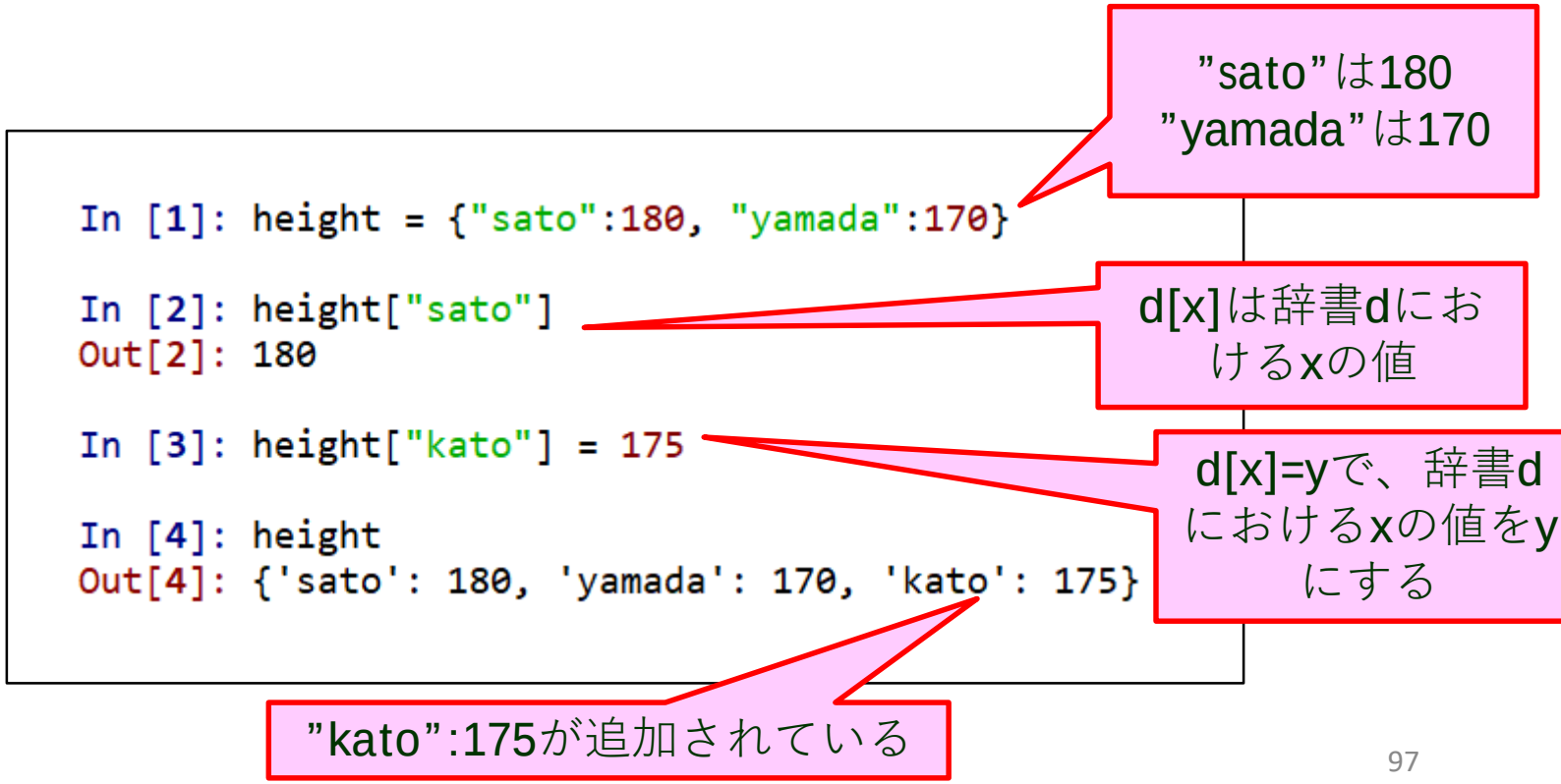
```
>>> a.append(10) #要素を追加

>>> a
[1, 1, 1, 10]

>>> a.extend([20, 30]) #リストの結合

>>> a
[1, 1, 1, 10, 20, 30]
```

リストの末尾に新しい要素を追加

リストとリストの結合

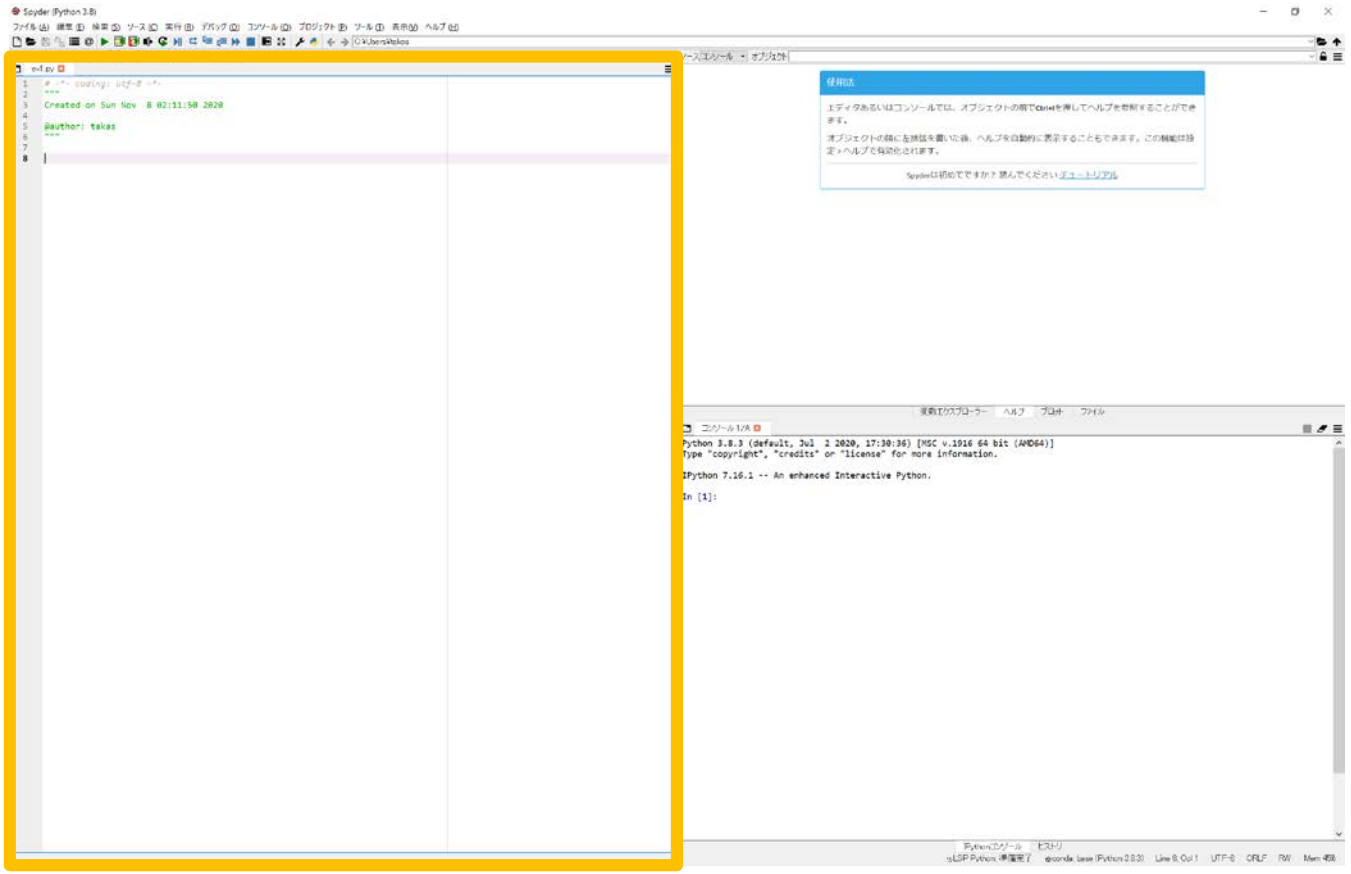
辞書型

- 一般には連想配列と呼ばれるデータ構造
  - キーと値のペアをたくさん格納したリスト
  - キーに対応する値をすぐに取り出すことができる
  - 「キー:値」を並べて中括弧で囲むことで生成する



エディタでプログラムを書こう！

- Spyder左側のエディタ部でプログラムを書きましょ
- Ubuntuの場合は、エディタ(gedit, emacs, vs code等)を起動しましょう



if文

- 条件分岐
  - ブロック: 一連の処理の範囲のことをブロックという。
  - C, JAVA, Processingなど普通の言語では中括弧{}を用いてブロックを指定する。
    - Processingの例

```
int a = 150;
if( a > 100) {
  background(255,0,0);
  ellipse(50, 50, 50, 50);
}
```

If文の条件式が真であったときのブロック

  - Pythonのブロックはインデントで指定する
    - Pythonの例

```
In [1]: a = 150

In [2]: if a > 100:
   ...:     a = 0
   ...:     print("1")
   ...: else:
   ...:     print("0")
   ...:
1
```

If文の条件式が真であったときのブロック

If文の条件式が偽であったときのブロック

空白4個かタブが一般的

if文

- エディタで次のようにif文を書いて実行

```
 8  a = 150
 9  if a > 100:
10      print("aは100より大きいです")
11  elif a > 0:
12      print("aは0より大きくて100以下です")
13  else:
14      print("aは0以下です")
15
```

aの値をいろいろと変えてどのように結果が変わるのかみてみよう

```
aは100より大きいです
```

if文の構文

条件式の後にコロン(:)が必要

```
if 条件式1:
        条件式1が真のときの処理
elif 条件式2:
        条件式1が偽で条件式2が真だったときの処理
…
else:
        上記の条件式がすべて偽だったときの処理
```

上から下に向かって実行

if文の条件式

- 条件式には次のようなものが使えます
  - 等号(等しい) ==
  - 不等号(より大きい) >
  - 不等号(より小さい) <
  - 不等号(以上) >=
  - 不等号(以下) <=
- また、ブール演算を使って、複雑な条件を書くこともできます
  - 例

```
 8   x = 150
 9   if 100 < x and x <= 200:
10       print("xは100より大きく200以下です")
11
```

for文

- Pythonのfor文はリストに対する繰り返し処理

```
 8  words = ["cat", "dog", "lion"]
 9  a = 1
10  for w in words:
11      print(w + ": " + str(a))
12      a = a + 1
13
```

文字列と数値の足し算はできない。
str関数を用いると数値を文字列に変換できる。

```
cat: 1
dog: 2
lion: 3
```

for文の構文

```
for 変数 in リスト:
        繰り返し処理のブロック
        （変数を参照しながら処理することができる）
```

for文

- Pythonのfor文はリストに対する繰り返し処理

```
 8  words = ["cat", "dog", "lion"]
 9  a = 1
10  for w in words:
11      print(w + ": " + str(a))
12      a = a + 1
```

①　②　③

繰り返し処理ブロック

①に対する処理　②に対する処理　③に対する処理

a=a+1の代わりにa+=1と書いてもよい
ただし、a++やa--は使えない

リストの先頭から順に各要素が変数に代入されて
繰り返し処理ブロックの処理が行われる

①w = ”cat”として繰り返し処理ブロックを実行
②w = ”dog”として繰り返し処理ブロックを実行
③w = ”lion”として繰り返し処理ブロックを実行

for文とrange関数

- 一定回数繰り返したいときは？→range関数を使う

- range関数
  - range(n)で[0, 1, 2, …, n-1]の(仮想的な)リストを作る
  - range(i, j)で[i, i+1, i+2, …, j-1]の(仮想的な)リストを作る

```
In [1]: list(range(20))
Out[1]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

In [2]: list(range(10, 20))
Out[2]: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

for文とrange関数

- for文とrange関数を用いた繰り返し

[0, 1, 2, …, 99]のリストがあると思えば良い

x = i + 1とすることで、xは1, 2, …, 100の値になる

```
 8  sum=0
 9  for i in range(100):
10      x = i + 1
11      sum = sum + x
12  print(sum)
13
```

```
5050
```

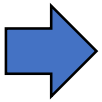
i = 0 としてx = 1, sum = sum + x
i = 1 としてx = 2, sum = sum + x
i = 2 としてx = 3, sum = sum + x
…
i = 99としてx = 100, sum = sum + x

これは$\sum_{x=1}^{100} x$ =1 + 2 + 3 + … + 100=5050の計算をしている

関数

- lenなど組み込みの関数を用いてきたが自分で新しく関数を定義することができる
- defで関数を定義

```
 7  def add(x, y):
 8      ans = x + y
 9      return ans
10
11  print(add(10, 30))
12
```



```
40
```

関数定義の構文

```
def 関数名(引数の列):
        関数内での処理
        return 返り値
        (返り値は関数が返す値)
```

Python演習(1)

- 次のプログラムを入力して実行してみましょう。

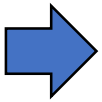
```
 8  def hello(x):
 9      print("hello "+ x + "!")
10
11  hello("cat")
```

```
In [1]: runfile('C:/User
講義/工学リテラシーⅢ')
hello cat!

In [2]:
```

Python演習(2)

- 次のプログラムを入力して実行してみよう

```
 8  def sign(x):
 9      if x < 0:
10          print("negative")
11          return -1
12      elif x == 0:
13          print("zero")
14          return 0
15      else:
16          print("positive")
17          return 1
18
19  x = sign(10)
20  y = sign(0)
21  z = sign(-5.3)
22  print(x, y, z)
23
```



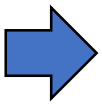
```
positive
zero
negative
1 0 -1
```

このような結果がでていたらうまく動いているということ

Python演習(3)

- 数値のリストを受け取ったとき、受け取ったリストの各要素を10倍にして返す関数を書きましょう。
  - まず、「ファイル」→「新規ファイル」を選んで新しいファイルを作成しましょう。
  - 続いて、「ファイル」→「保存」(もしくは「形式を指定して保存」)を選んでファイル名(tentimes.py)をつけて保存しましょう。どこのフォルダに保存されているのか注意して保存しましょう。

```
 8  def tentimes(numlist):
 9      #ここにプログラムを書く
10
11
12  print(tentimes([1,2,3,4,5]))
13
```



```
[10, 20, 30, 40, 50]
```

うまくプログラムが書けたらこのような結果が返ってくる

Python演習 解答例

- 解答例

```
 8  def tentimes(numlist):
 9      r = []
10      for n in numlist:
11          r = r + [10*n]
12      return r
13
14
15  print(tentimes([1,2,3,4,5]))
16
```

最初にrに空リストをいれる

rにnumlistの各要素を10倍にした値を追加していく

リストの連結はリスト同士でないとできないので、要素1個のリスト[10*n]を作ってから連結している

最後にrを返す

Python演習 解答例

- 別解

```
15  def tentimes(numlist):
16      r = [0]*len(numlist)
17      for i in range(len(numlist)):
18          r[i] = 10 * numlist[i]
19      return r
20
21  print(tentimes([1,2,3,4,5]))
22
```

最初にnumlistと同じ長さのダミーのリストを作っておく

i=0, .., (numlistの大きさ-1)まで繰り返す

r[i]をnumlist[i]の10倍の値にする

最後にrを返す

Python演習(4)

- 与えられたリストを逆順にして返す関数reverseを作成しよう

Python演習(5)

- **フィボナッチ数列を計算しよう**

$$fib(n) = \begin{cases} 0 & if\ n = 0 \\ 1 & if\ n = 1 \\ fib(n-1) + fib(n-2) & if\ n \geq 2 \end{cases}$$

フィボナッチ数列の関数を定義して
定義した関数を使って`fib(1)`から`fib(20)`を
順に表示してみてください

※時間が余った人は、フィボナッチ数列をリストで返すようにしてみよう

Python演習(6)

- **微分の近似計算**

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

※非常に小さなhを与えれば良い。

※高階関数で書けるとさらに良い。(上級者向け)

$f(x) = x^3$ とし，　$f'(5)$を計算，
表示してみてください

クラス

- **独自のデータ型の定義**
  - メソッド(クラス用の関数)や属性(クラス用の変数)の定義が可能

クラスの定義

man.py

```
class Man:
    def __init__(self, name):
        self.name = name
        print('Initialized!')

    def hello(self):
        print('hello ' + self.name + '!')

    def goodbye(self):
        print('good bye ' + self.name + '!')

m = Man('Taro') #Manクラスのインスタンス生成
m.hello()
m.goodbye()
```

__init__：コンストラクタ（初期化用メソッド）

.でメソッドやインスタンス変数にアクセス（クラス内ではself.）

メソッドの第一引数にはselfを明示的に書く

実行結果

```
$ python3 man.py
Initialized!
hello Taro!
Good bye Taro!
```

Torch

- **Pythonで高速に行列計算を行うためのライブラリ**
  - 行列演算は単純な足し算掛け算を繰り返すがpythonはこういった計算が遅くて苦手
  - Pythonの便利さと高速計算を同時に実現する
- **Torchの配列クラス(torch.tensor)には多次元配列を操作するための便利なメソッドが多数用意されている**
- **Torchは外部ライブラリなのでインポートして利用する**

```
>>> import torch
```

torch：配列の生成

- **torch配列の生成**
  - メソッドtensor()を使用する。引数はPythonのリスト。
  - 多次元配列も生成可能。2次元配列は行列と捉えられる。
  - 属性size()に配列の形状、dtypeに要素のデータ型が保存されている

2×3の行列生成

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
>>> import torch
>>> x = torch.tensor([1.0, 2.0, 3.0])
>>> x
tensor([ 1.,  2.,  3. ])
>> type(x)
<class 'torch.Tensor'>
>>> A = torch.tensor([[1, 2, 3], [4,5,6]])
>>> A
tensor([[1, 2, 3],
        [4, 5, 6]])
>>> A.size()
torch.Size([2, 3])
>>> A.dtype
torch.int64
```

Torch：配列の演算 (1)

- **四則演算子による演算は要素毎の演算**

$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, B = \begin{pmatrix} -1 & 1 \\ 5 & 6 \end{pmatrix}$

```
>>> x = torch.tensor([1., 2., 3.])
>>> y = torch.tensor([2., 4., 6.])
>>> x + y
tensor([3., 6., 9.])
>>> x - y
tensor([-1., -2., -3.])
>>> x * y
tensor([2., 8.,18.])
>>> x / y
tensor([0.5000,0.5000,0.5000])
```

```
>>> A = torch.tensor([[1, 2], [3, 4]])
>>> B = torch.tensor([[-1, 1], [5, 6]])
>>> A + B
tensor([[ 0,  3],
        [ 8, 10]])
>>> A * B
tensor([[-1,  2],
        [15, 24]])
>>> torch.matmul(A,B)
tensor([[ 9, 13],
        [17, 27]])
```

要素毎の積

$\begin{pmatrix} 1\times -1 & 2\times 1 \\ 3\times 5 & 4\times 6 \end{pmatrix}$

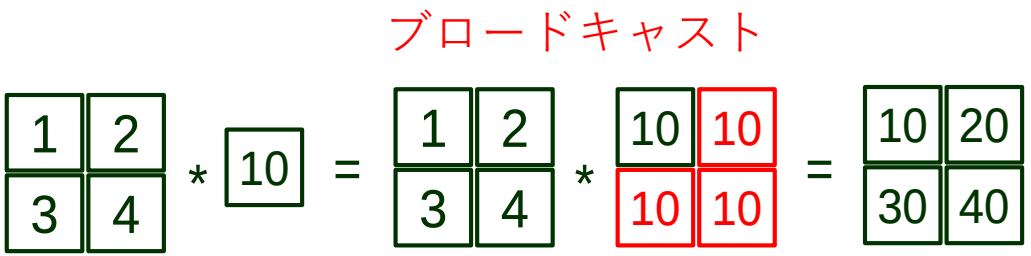
行列の積（ドット積）は matmulメソッド

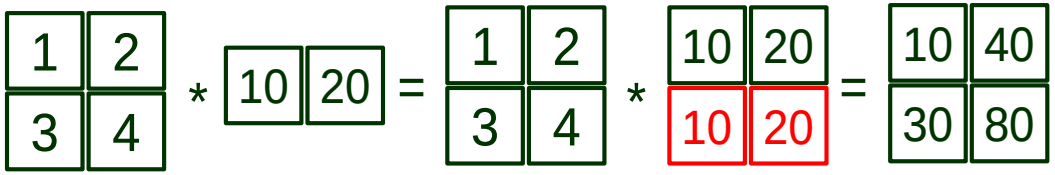
$\begin{pmatrix} 1\times -1+2\times 5 & 1\times 1+2\times 6 \\ 3\times -1+4\times 5 & 3\times 1+4\times 6 \end{pmatrix}$

Torch：配列の演算 (2)

- **配列の形状が異なる場合は、形状を揃えて要素毎の演算がされる(ブロードキャスト)。**

```
>>> A = torch.tensor([[1,
2], [3, 4]])
>>> A*10
tensor([[10, 20],
        [30, 40]])
>>> A+10
tensor([[11, 12],
        [13, 14]])
>>> B = torch.tensor([10,
20])
>>> A*B
tensor([[10, 40],
        [30, 80]])
```





要素へのアクセス

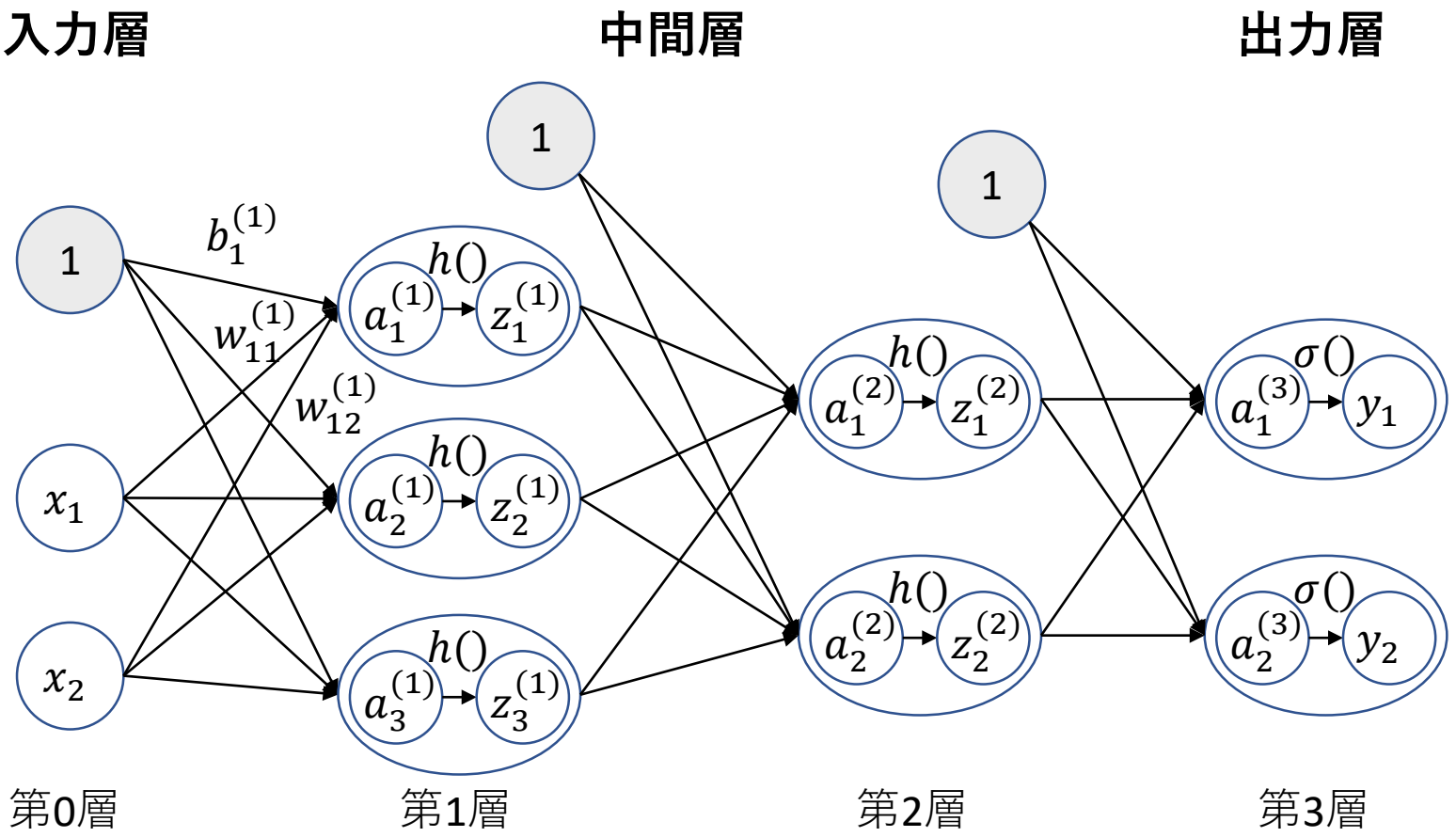
- **Pythonリストの要素へのアクセスと同様にインデックスによりアクセス可能**
- **加えて、tensor配列によるアクセスも可能**
  - インデックスの配列により指定番目の要素だけ取得
  - ブーリアンの配列によりTrueに対応する要素を取得

```
>>> X = torch.tensor([[10, 40], [15, 30], [1, 0]])
>>> X
tensor([[10, 40],
        [15, 30],
        [ 1 , 0]])
>>> X[0] #0行目
tensor([10, 40])
>>> X[0][1] #(0,1)の要素
tensor(40)
>>> X = X.flatten()
>>> X
tensor([10, 40, 15, 30,  1,  0])
>>> X[torch.tensor([0,2,4])]
tensor([10, 15,  1])
>>> X>20
tensor([False,  True, False,  True, False, False])
>>> X[X>20]
tensor([40, 30])
```

Xを1次元配列に変換

tensor配列に対して不等号演算を行うと配列の各要素に対して不等号演算が行われた結果のブーリアンの配列となる

3層NNの動作

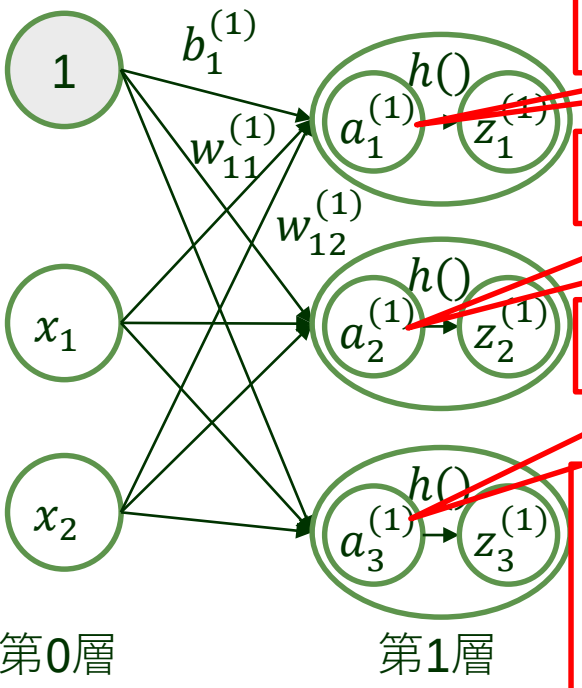


$h()$：シグモイド関数、$\sigma()$：恒等関数

信号は低層から順に伝わる
→低層の演算から順に行う

1層目の動作の実装

Torchの行列演算で実行すること
でfor文で回すより圧倒的に早い！



$h()$：シグモイド関数

$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

$$a_2^{(1)} = w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + b_2^{(1)}$$

$$a_3^{(1)} = w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + b_3^{(1)}$$

$$A^{(1)} = XW^{(1)} + B^{(1)}$$

$$A^{(1)} = \left(a_1^{(1)}, a_2^{(1)}, a_3^{(1)}\right),$$

$$X = (x_1, x_2),$$

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix},$$

$$B^{(1)} = \left(b_1^{(1)}, b_2^{(1)}, b_3^{(1)}\right)$$

```
x1, x2 = 1.0, 0.5
w1_11, w1_21, w1_31 = 0.1, 0.3, 0.5
w1_12, w1_22, w1_32 = 0.2, 0.4, 0.6
b1_1,b1_2, b1_3 = 0.1, 0.2, 0.3
X=torch.tensor([x1, x2])
W1 = torch.tensor([[w1_11, w1_21, w1_31], [w1_12, w1_22, w1_32]])
B1 = torch.tensor([b1_1, b1_2, b1_3])
A1 = torch.matmul(X, W1) + B1
print(A1)
#[0.3000, 0.7000, 1.1000]

Z1 = torch.sigmoid(A1)
print(Z1)
#[0.5744, 0.6682, 0.7503]
```

Python演習(7)：3層NNの推論の実装

infer.py

```
import torch

w1 = torch.tensor([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])
b1 = torch.tensor([0.1, 0.2, 0.3])
w2 = torch.tensor([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
b2 = torch.tensor([0.1, 0.2])
w3 = torch.tensor([[0.1, 0.3], [0.2, 0.4]])
b3 = torch.tensor([0.1, 0.2])


def sigmoid(x):
    #torch.exp()を使ってシグモイド関数を実装してみてください

def forward(x):
    #推論プロセスを実装してみてください

#実行
x = torch.tensor([1.0, 0.5])
y = forward(x)
print(y)
```

実行結果：

```
$ python3 infer.py
tensor([0.3168, 0.6963])
```

まとめ

- **PythonとPytorchの基本を演習を通して学んだ**