知能プログラミング演習 I

第 1 回: Python 入門

梅津 佑太
2 号館 404A: umezu.yuta@nitech.ac.jp

この講義について

講義の目的

- フリーの言語 python を使って深層学習の基礎を学ぶ
- 実際に自分で手を動かしてアルゴリズムを習得する
- 講義ノート (工事中) は Moodle で随時更新します

成績評価

- 出席 (**必須**)+レポート (100%)

オフィスアワー (**要事前連絡**)

- 梅津 (2 号館 404A): umezu.yuta@nitech.ac.jp

講義内容 (予定)

※シラバスから若干の修正があります※

1. python **入門**
2. 3 **層ニューラルネットワーク**
3. **深層ニューラルネットワーク**
4. **オートエンコーダー**
5. **リカレントニューラルネットワーク**
6. **たたみ込みニューラルネットワーク I**
7. **たたみ込みニューラルネットワーク II**
8. **発展的な話題**

今日の講義内容

1. python の使い方
2. パーセプトロンの学習

python の起動と終了

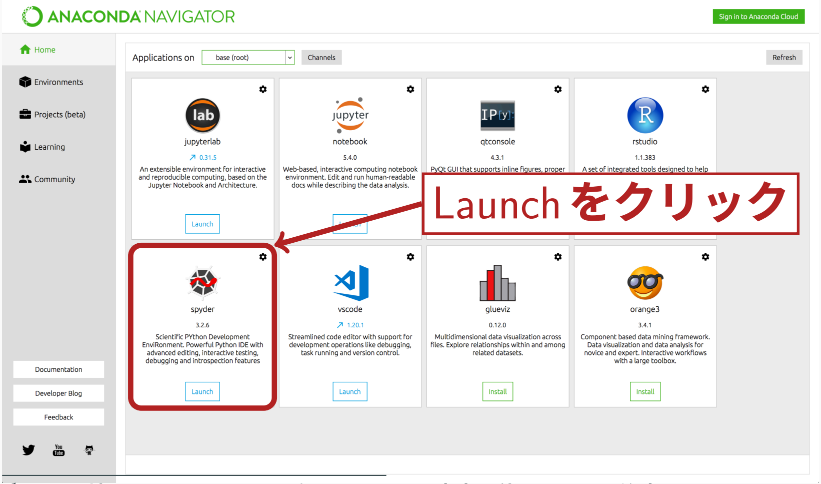
- ターミナルを起動し, `python` または `ipython` を実行することで利用可能.
  - ✓ バージョンも含めて起動する場合は `python3` を実行する
- ファイルをターミナルから実行する場合,

  `python　(ファイル名).py`

  などとすれば良い.
- python を終了する場合は `quit()` または `exit()` を実行

自前の PC で python を利用する場合

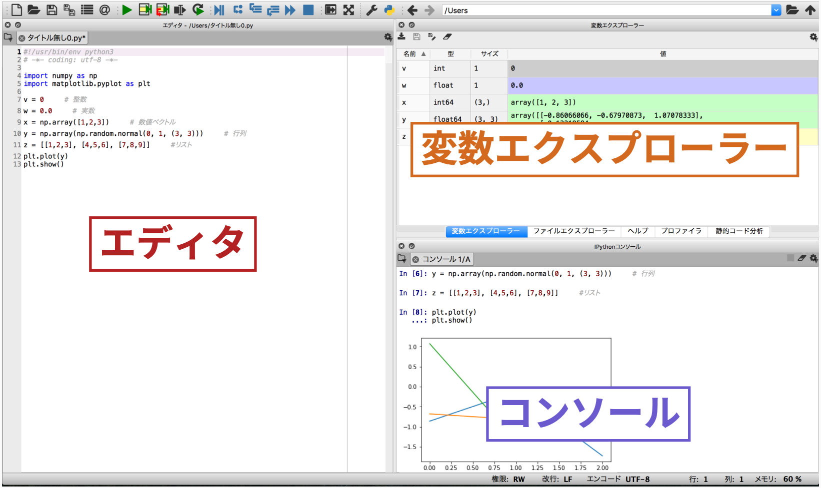
- Anaconda を自前の PC にインストールするのが便利で良い [1]
- Anaconda, spyder の順に起動し python を立ち上げる



[1] CSE 環境ではこのスライドと次のスライドの内容は使えないので注意

spyder

- **エディタでアルゴリズムを作成 → コンソールで実行**
- **定義済みの変数は変数エクスプローラーで表示される**



numpy と matplotlib

エディタ - /Users/タイトル無し0.py

タイトル無し0.py*

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt

```

- numpy: **ベクトルや行列演算のための数値計算モジュール**
- matplotlib: **グラフ描画ライブラリ**

python のデータタイプ

- **数値**

  v = 0　# **整数** (int),　w = 0.0　# **実数** (float)
- **ベクトル**

  x = np.array([1, 2, 3])　# **数値ベクトル**
- **行列**

  y = np.array(np.random.normal(0, 1, (3, 3)))　# **行列**
- **リスト**

  z = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]　# **リスト**

変数エクスプローラー

名前 ▲	型	サイズ	値
v	int	1	0
w	float	1	0.0
x	int64	(3,)	array([1, 2, 3])
y	float64	(3, 3)	array([[-0.86066066, -0.67970873, 1.07078333], [-0.13219584, - ...
z	list	3	[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

ベクトル演算 I

```
In [2]: x = np.array([1, 2, 3], float)

In [3]: x[0]
Out[3]: 1.0

In [4]: x + 0.5
Out[4]: array([1.5, 2.5, 3.5])

In [5]: 2*x     # 要素ごとの定数倍
Out[5]: array([2., 4., 6.])

In [6]: x**3     # 要素ごとのべき
Out[6]: array([ 1.,  8., 27.])

In [7]: x*x     # 要素ごとの積
Out[7]: array([1., 4., 9.])
```

ベクトル演算 II

```
In [8]: y = np.array([4, 5, 6], float)

In [9]: x + y      # 要素ごとの和
Out[9]: array([5., 7., 9.])

In [10]: np.dot(x, y)     # 内積
Out[10]: 32.0

In [11]: np.outer(x, y)     # 行列積
Out[11]:
array([[ 4.,  5.,  6.],
       [ 8., 10., 12.],
       [12., 15., 18.]])

In [12]: np.outer(y, x)     # 行列積
Out[12]:
array([[ 4.,  8., 12.],
       [ 5., 10., 15.],
       [ 6., 12., 18.]])
```

行列演算

```
In [2]: A = np.random.normal(0, 1, (3,3))     # 3*3行列

In [3]: B = np.random.normal(0, 1, (2,3))     # 2*3行列

In [4]: A
Out[4]:
array([[-0.54402056, -2.25872705, -0.84008799],
       [-0.45872658, -0.65348438,  0.14641837],
       [-1.9583066 ,  0.11243239,  1.5944507 ]])

In [5]: A.T      # 転置
Out[5]:
array([[-0.54402056, -0.45872658, -1.9583066 ],
       [-2.25872705, -0.65348438,  0.11243239],
       [-0.84008799,  0.14641837,  1.5944507 ]])

In [6]: np.dot(B, A)      # 積BA
Out[6]:
array([[-2.68199056,  0.70173615,  2.38960981],
       [-1.84463277, -2.56791097, -0.06458365]])

In [7]: np.dot(A, B)      # Aの列数とBの行数が異なるのでエラーが表示される
Traceback (most recent call last):

  File "<ipython-input-7-bb50fe1a139b>", line 1, in <module>
    np.dot(A, B)      # Aの列数とBの行数が異なるのでエラーが表示される

ValueError: shapes (3,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

for 文と if 文

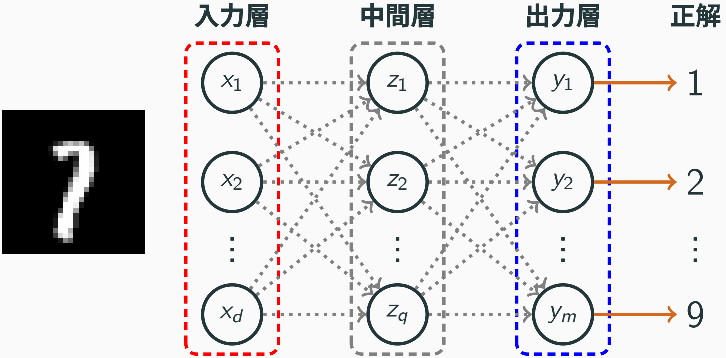
- $i = 0, 1, \ldots, 9$ まで順番に足す ($x = 0 + 1 + \cdots + 9$)

```
In [1]: x = 0
   ...: for i in range(0, 10):
   ...:     x = x+i
   ...:
   ...:

In [2]: x
Out[2]: 45
```

- j が偶数なら y に $j/2$ を加え，それ以外なら $(j+1)/2$ を加える

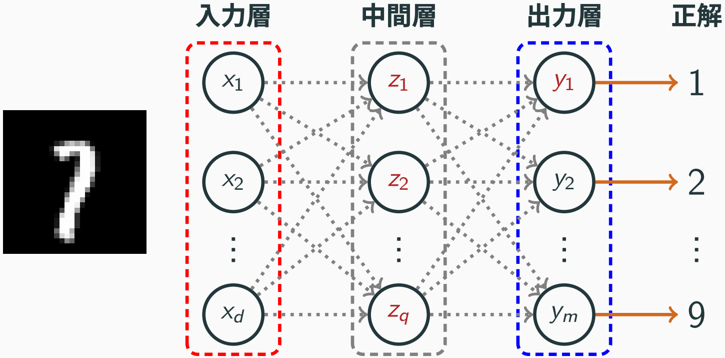
```
In [3]: y = 0
   ...: for j in range(0, 10):
   ...:     if j % 2 == 0:
   ...:      y = y + j/2
   ...:     else:
   ...:      y = y + (j+1)/2
   ...:
   ...:

In [4]: y
Out[4]: 25.0
```

ニューラルネットワーク



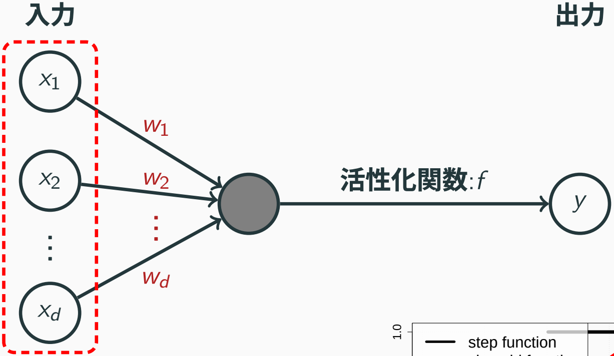
- ネットワークから得られた出力を正解と比較してモデルを学習

ニューラルネットワーク



- ネットワークから得られた出力を正解と比較してモデルを学習
- 各ユニットにつながるネットワークはパーセプトロンと呼ばれる

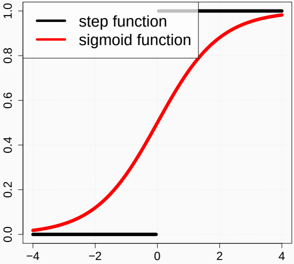
パーセプトロン



右図のような活性化関数 f を用いて

$$y = f(w_0 + w_1 x_1 + \cdots + w_d x_d)$$

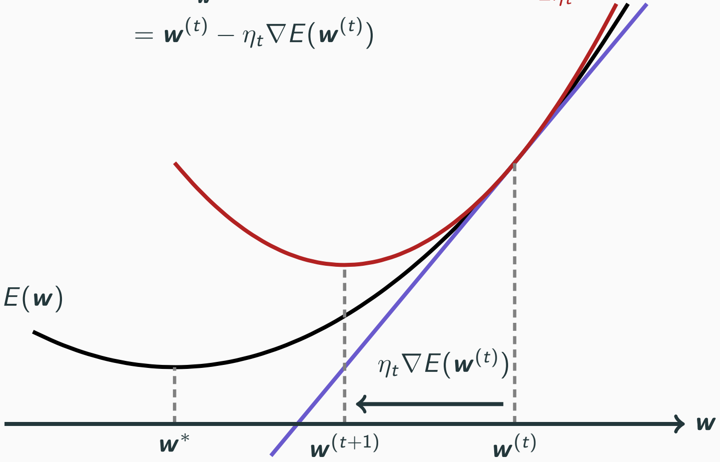
によって入力と出力の関係をモデル化し，データから重み $w_0, w_1, \ldots, w_d$ を学習



勾配降下法

- 誤差関数 $E(\boldsymbol{w})$ を最小にする $\boldsymbol{w}$ を求めるためのアルゴリズム

$$\begin{aligned}\boldsymbol{w}^{(t+1)} &= \arg\min_{\boldsymbol{w}} \nabla E(\boldsymbol{w}^{(t)})^\top (\boldsymbol{w} - \boldsymbol{w}^{(t)}) + \frac{1}{2\eta_t}\|\boldsymbol{w} - \boldsymbol{w}^{(t)}\|^2 \\ &= \boldsymbol{w}^{(t)} - \eta_t \nabla E(\boldsymbol{w}^{(t)})\end{aligned}$$



準備: シグモイド関数とその微分

シグモイド関数

$$y = f(w_0 + w_1 x_1 + \cdots + w_d x_d) = f(\boldsymbol{w}^\top \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^\top \boldsymbol{x}}}$$

- **練習**:

$$f(x) = \frac{1}{1 + e^{-x}} \quad \Rightarrow \quad \frac{\mathrm{d}f(x)}{\mathrm{d}x} = \frac{e^{-x}}{(1 + e^{-x})^2} = f(x)(1 - f(x))$$

- **シグモイド関数の微分** [2]:

$$\frac{\partial f(\boldsymbol{w}^\top \boldsymbol{x})}{\partial \boldsymbol{w}} = \underbrace{\frac{\partial f(\boldsymbol{w}^\top \boldsymbol{x})}{\partial \boldsymbol{w}^\top \boldsymbol{x}} \frac{\partial \boldsymbol{w}^\top \boldsymbol{x}}{\partial \boldsymbol{w}}}_{\text{合成関数の微分}} = f(\boldsymbol{w}^\top \boldsymbol{x})(1 - f(\boldsymbol{w}^\top \boldsymbol{x}))\boldsymbol{x}$$

[2] 偏微分を ∇ で表すこともある

準備: 誤差関数とその微分

誤差関数

$$E(\boldsymbol{w}) = -y \log f(\boldsymbol{w}^\top \boldsymbol{x}) - (1-y)\log(1 - f(\boldsymbol{w}^\top \boldsymbol{x}))$$

- **誤差関数の微分**:

$$\begin{aligned}\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} &= -y \underbrace{\frac{1}{f(\boldsymbol{w}^\top \boldsymbol{x})}\frac{\partial f(\boldsymbol{w}^\top \boldsymbol{x})}{\partial \boldsymbol{w}}}_{\text{対数関数の微分}} - (1-y)\underbrace{\frac{-1}{1-f(\boldsymbol{w}^\top \boldsymbol{x})}\frac{\partial f(\boldsymbol{w}^\top \boldsymbol{x})}{\partial \boldsymbol{w}}}_{\text{対数関数の微分}} \\ &= (f(\boldsymbol{w}^\top \boldsymbol{x}) - y)\boldsymbol{x}\end{aligned}$$

確率的勾配降下法

- **ランダムに**データ $\{(y_i, \boldsymbol{x}_i)\}$ を並べ替える
- 順番にデータ $y, \boldsymbol{x}$ を読み，誤差関数

$$E(\boldsymbol{w}) = -y \log f(\boldsymbol{w}^\top \boldsymbol{x}) - (1-y)\log(1 - f(\boldsymbol{w}^\top \boldsymbol{x}))$$

  を最小にする重み $w_0, w_1, \ldots, w_d$ を確率的勾配降下法で求める
- 適当な初期値 $\boldsymbol{w}^0$ から始め，学習率を η_t として重みを更新[3]

確率的勾配降下法

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta_t \Big(f(\boldsymbol{w}^{t\top} \boldsymbol{x}) - y \Big) \boldsymbol{x}$$

[3] 誤差が減少する様子はエポック (データを 1 回スキャンすること) ごとに確認する