mi-12. Python でニューラルネットワークを動かしてみる

（人工知能）

https://www.kkaneko.jp/cc/mi/index.html

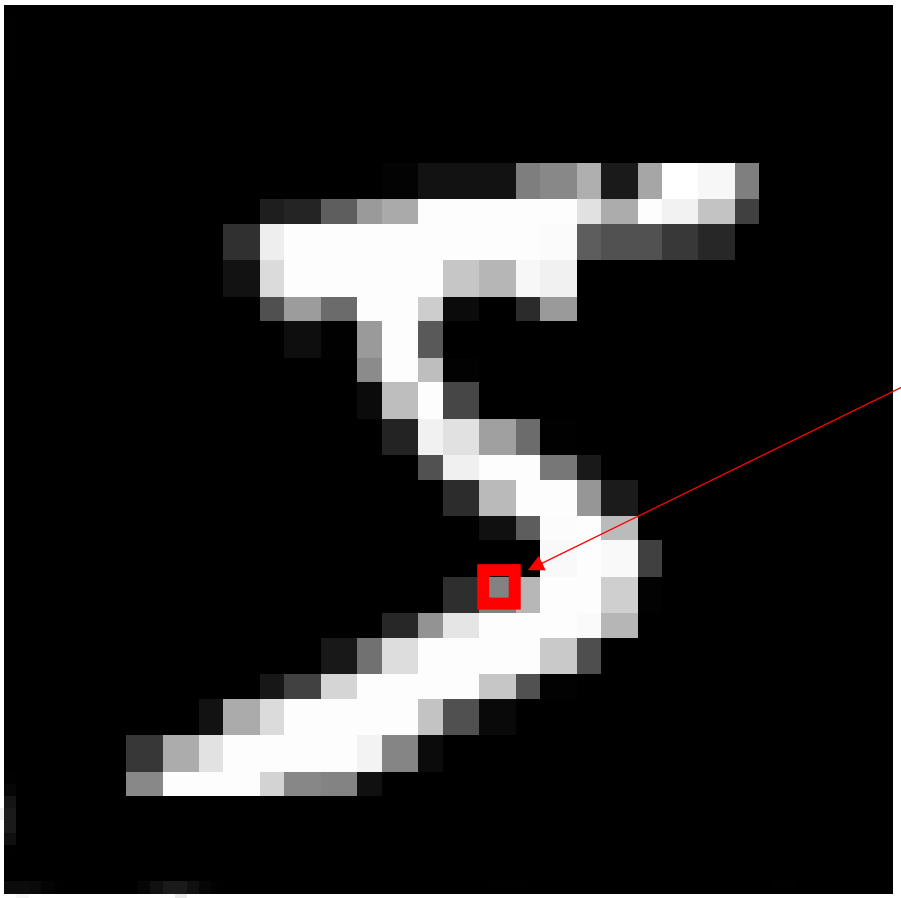
金子邦彦

配列（アレイ）とは

0	1	2	3
180	**20**	**250**	**40**

配列（アレイ）とは，**データの並び**で，
０から始まる番号（添字）が付いている

画像と画素



MNISTデータセット（手書き文字のデータセットで，濃淡画像）

画像サイズ: 28 × 28

画素

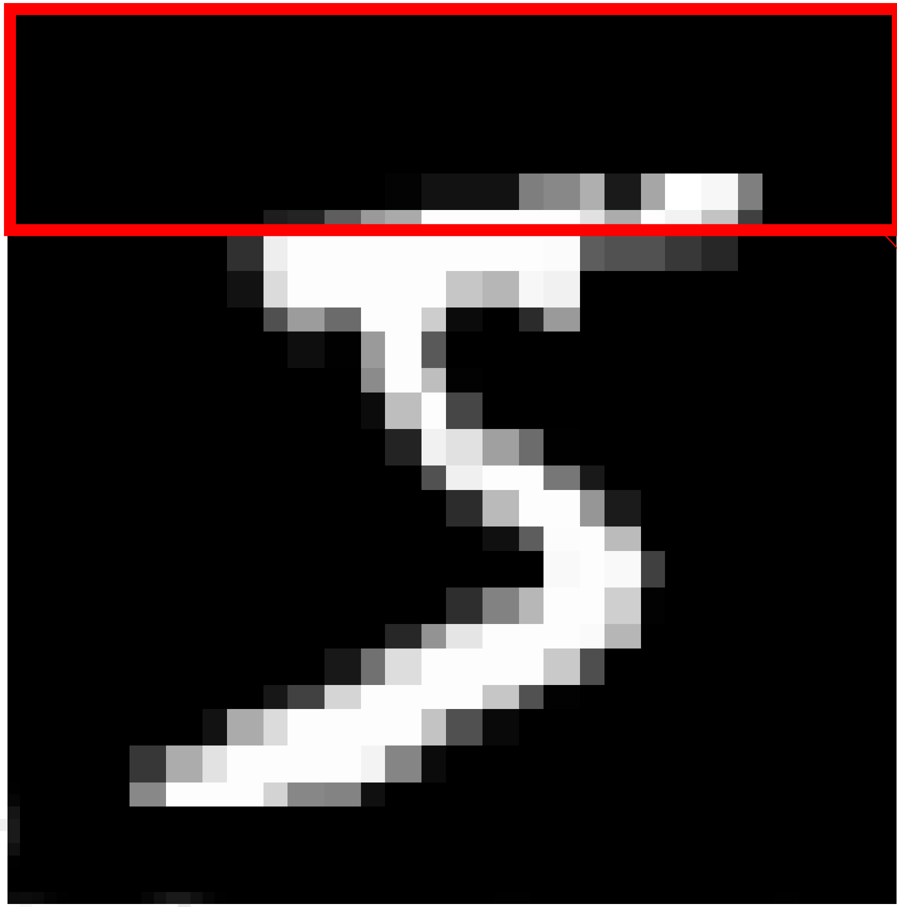
画素値

白　255

黒　0

画素値は，**画素**の明るさに応じた **0 から 255 の数値**

画像と配列（アレイ）



MNISTデータセット（手書き文字のデータセットで，濃淡画像）

画像サイズ: 28 × 28

画像全体は，サイズ 28 ×28 の**配列（アレイ）**

```
[[   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
   175  26 166 255 247 127   0   0   0   0]
 [   0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
   225 172 253 242 195  64   0   0   0   0]
```

画像の**上 7行分**の画素値を表示したところ（28 ×7分）

画像データセット MNIST

- 濃淡画像 **70000**枚．画像サイズは 28 × 28

  うち学習用 **60000**枚

  配列（アレイ）の形：**60000 × 28 × 28**

  テスト用 **10000**枚

  配列（アレイ）の形：**10000 × 28 × 28**

11-1. 配列（アレイ）の形と次元

サイズ 28 × 28 の濃淡画像は

配列（アレイ）の形：**28 × 28**

次元: 2

ノートページ

サイズ 28 × 28 の濃淡画像 60000 枚の画像データセットは

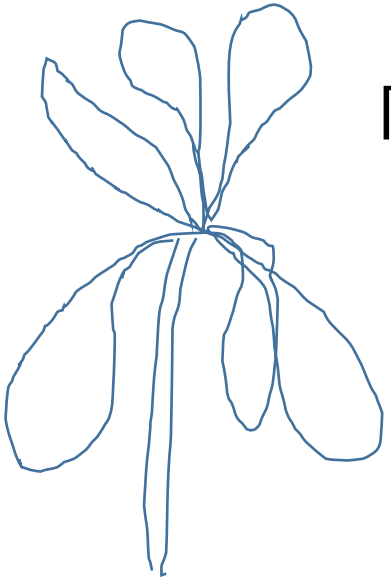
配列（アレイ）の形：**60000 × 28 × 28**

次元: 3

配列（アレイ）の形と次元

データ	配列（アレイ）の形	次元数
[8 5 4 1 3]	5	次元数は 1
1枚の画像	28 × 28	次元数は 2
60000枚の画像（同じ大きさ）	60000 × 28 × 28	次元数は 3

アヤメ属 (Iris)



内花被片

外花被片

◆多年草

◆世界に 150種. 日本に 9種.

◆花被片は 6個

外花被片（がいかひへん） **Sepal**

3個（大型で下に垂れる）

内花被片（ないかひへん） **Petal**

3個（直立する）

Iris データセット

Iris データセット（データ数は 50 × 3）
のうち、先頭 10 行

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa

外花被片(**Sepal**)の長さと幅　**内花被片**(**Petal**)の長さと幅　**種類**

特徴量　**ラベル**

- 3種のアヤメの外花被辺、内花被片を計測
- 種類も記録
  - setosa
  - versicolor
  - virginica
- データ数は 50 × 3

作成者：Ronald Fisher

作成年：1936

Iris データセットと、配列（アレイ）

Iris データセット（データ数は 50 × 3）のうち、先頭 10 行

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
```

```
[0
 0
 0
 0
 0
 0
 0
 0
 0
 0
```

特徴量（数値）が入った**配列**
サイズ：**150 × 4**
次元数: **2**

ラベル（数値）が入った**配列**
サイズ：**150**
次元数: **1**

setosa → 0
versicolor → 1
virginia → 2

ラベルの数値化

Iris データセットをグラフにする

Iris データセット（データ数は 50 × 3）のうち、先頭 10 行

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa

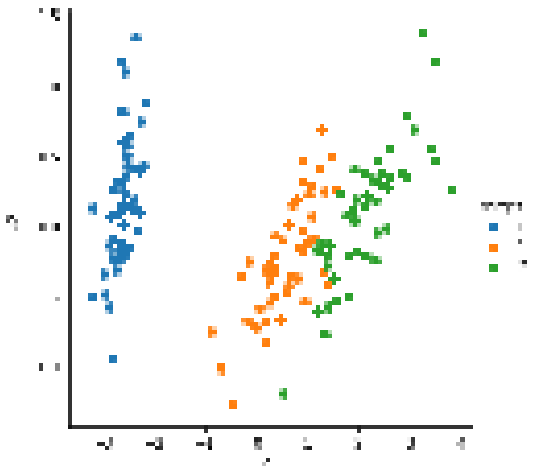
```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

サイズ：**150 × 4**

```
[0
0
0
0
0
0
0
0
0
0
```

サイズ：**150**

主成分分析



```
[[-2.68412563  0.31939725]
 [-2.71414169 -0.17700123]
 [-2.88899057 -0.14494943]
 [-2.74534286 -0.31829898]
 [-2.72871654  0.32675451]
 [-2.28085963  0.74133045]
 [-2.82053775 -0.08946138]
 [-2.62614497  0.16338496]
 [-2.88638273 -0.57831175]
 [-2.6727558  -0.11377425]]
```

サイズ：**150 × 2**

x 座標と y 座標

```
[0
0
0
0
0
0
0
0
0
0
```

サイズ：**150**

色

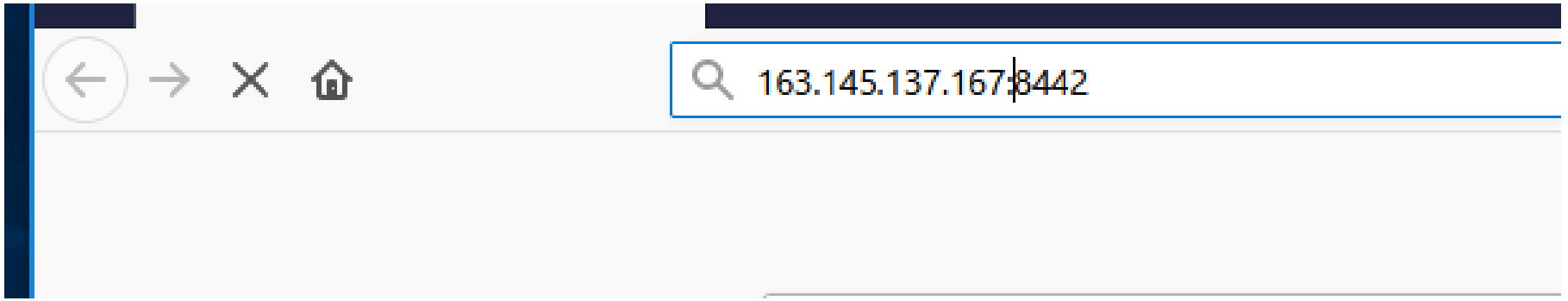
11-1 Python の配列（アレイ）と画像データセット

パソコン実習

① Web ブラウザを使う

② Web ブラウザで，次の URL を入れる

http://163.145.137.167:8442/



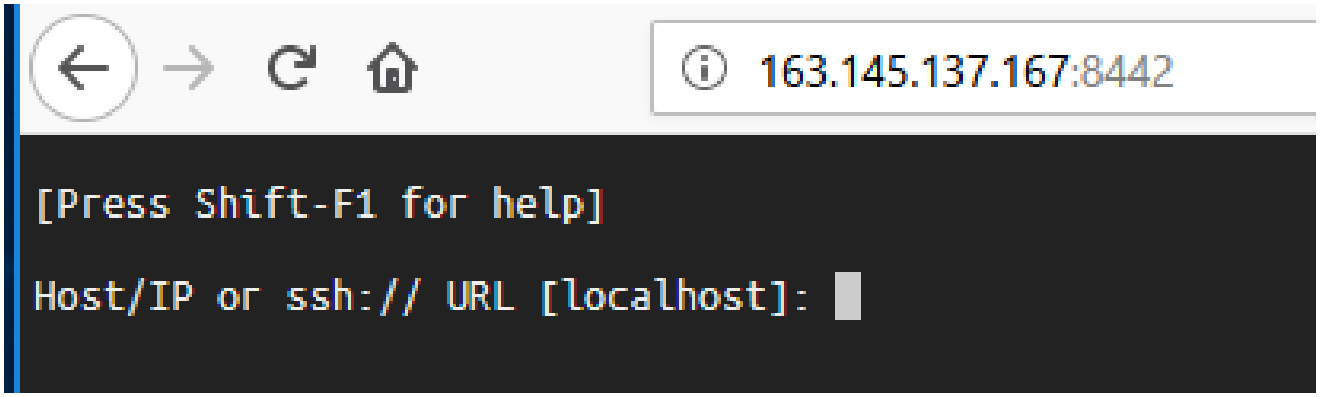
リモート接続するための準備をしている

IE よりも Google Chrome Web ブラウザを勧める

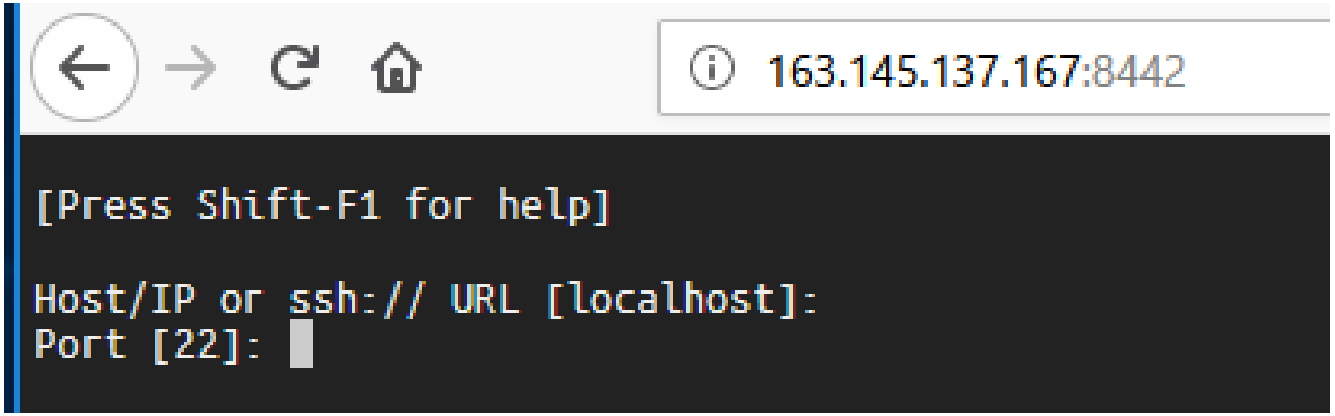
③ 画面が変わる．「**Terminal: SSH**」をクリック

④ 画面が変わる.

「Host/IP or ssh:// URL [localhost];」に対しては **Enter キー**

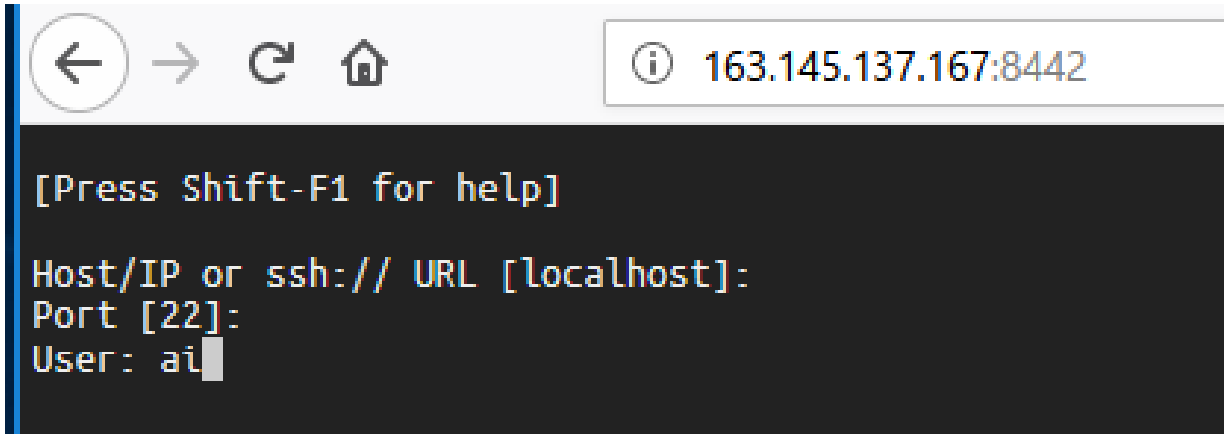


⑤ 「Port [22]:」に対しては **Enter キー**



⑥ 「User:」に対しては

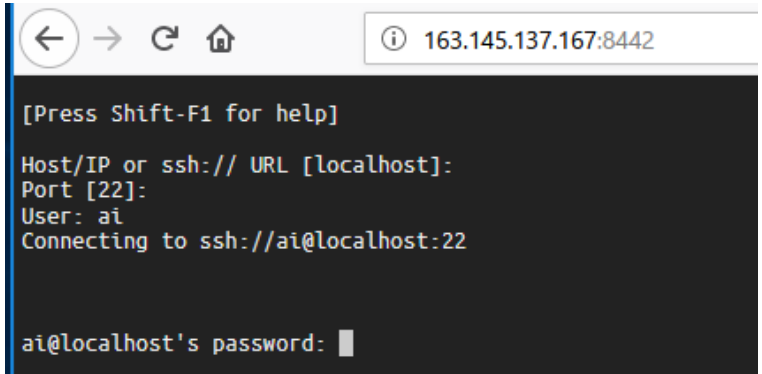
ai Enterキー　（エイ　アイ　Enter キー）



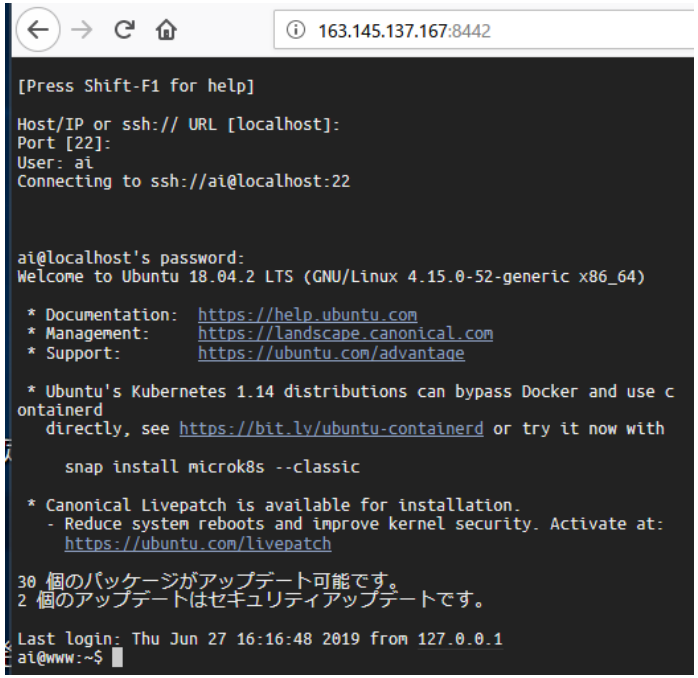
⑦ 次は

ai1234!!!!　Enterキー（エイ　アイ　１２３４！！！！）

※　**画面には表示されない**



⑧ 画面が変わるので確認



⑨ 次に

python3　Enterキー

※これは　⑪以降で使うので閉じないこと

①から⑨で行ったこと

- **授業用のマシン**を使用
- **インターネットを使ってリモート接続**
- 「**python3**」を起動　※ python3 は実習済み

パソコン演習

⑩ 資料の Web ページを，**別に**開く

https://www.kkaneko.jp/a/m.html

ケイ　かねこ　ドット　ジェイ　ピー

スラッシュ

エイ

スラッシュ

エム　ドット　エイチ　ティー　エム　エル

⑪「11-1」のところの**1つめ**のプログラムを，コピー，貼り付け

第11回授業

11-1. **Python の配列(アレイ) と画像データセット**

画像データセット MNIST を準備するプログラム

変数 X_train: サイズ 28 × 28 の 60000枚の濃淡画像

変数 y_train: 60000枚の濃淡画像それぞれの,種類番号(0 から 9 のどれか)

変数 X_test: サイズ 28 × 28 の 10000枚の濃淡画像

変数 y_test: 10000枚の濃淡画像それぞれの,種類番号(0 から 9 のどれか)

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Ubuntu's Kubernetes 1.14 distributions can bypass Docker and use c
ontainerd
   directly, see https://bit.ly/ubuntu-containerd or try it now with

     snap install microk8s --classic

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

30 個のパッケージがアップデート可能です。
2 個のアップデートはセキュリティアップデートです。

Last login: Thu Jun 27 16:16:48 2019 from 127.0.0.1
ai@www:~$ python3
Python 3.6.8 (default, Jan 14 2019, 11:02:34)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information
>>> from keras.datasets import mnist
Using TensorFlow backend.
>>> (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

コピーしたい範囲をマウスで選び,
マウスの**右クリックメニュー**で「**コピー**」が便利

⑨の画面に**張り付ける**.
張り付けたあと **Enter キー**

マウスの**右クリックメニュー**で「**貼り付け**」が便利

⑪ 「**11-1**」のところの**１つめ**のプログラムを，コピー，貼り付け

画像データセット MNIST を準備するプログラム

```
>>> from keras.datasets import mnist
Using TensorFlow backend.
>>> (X_train, y_train), (X_test, y_test) = mnist.load_data()
>>>
```

⑫ 「11-1」のところの**２つめ**のプログラムを，コピー，貼り付け

「サイズ 28 × 28 の 60000枚の濃淡画像」の形と次元を確認するプログラム

```
>>> print( X_train.shape )
(60000, 28, 28)
>>> print( X_train.ndim )
3
>>>
```

(60000, 28, 28),
3
のように表示されるので確認

⑬ 「11-1」のところの**３つめ**のプログラムを，コピー，貼り付け

ニューラルネットワークを使うために，データの形などを調整

```
>>> import tensorflow as tf
>>> import keras
>>> X_train = X_train.reshape(60000, 784)
>>> X_test = X_test.reshape(10000, 784)
>>> X_train = X_train.astype('float32')
>>> X_test = X_test.astype('float32')
>>> X_train /= 255
>>> X_test /= 255
>>> print( X_train.shape )
(60000, 784)
>>> print( X_train.ndim )
2
>>>
```

(60000, 784),
2
のように表示されるので確認

⑭ 「11-1」のところの**４つめ**のプログラムを，コピー，貼り付け

ニューラルネットワークを作るプログラム

```
>>> import tensorflow as tf
>>> import keras
>>> from keras.models import Sequential
>>> m = Sequential()
>>> from keras.layers import Dense, Activation, Dropout
>>> import keras.optimizers
>>> m.add(Dense(units=64, activation='relu', input_dim=784))
>>> m.add(Dropout(rate=0.8))
>>> m.add(Dense(units=10, activation='softmax'))
>>> m.compile(loss=keras.losses.categorical_crossentropy,
...           optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9, n
esterov=True))
>>>
```

2層の**ニューラルネットワーク**を作成

⑮ 「11-1」のところの**５つめ**のプログラムを，コピー，貼り付け

ニューラルネットワークの確認表示

```
>>> print(m.summary())
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 64)                50240
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_12 (Dense)             (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
None
>>>
```

表示は 2層の**ニューラルネットワーク**が
できたことを示す

⑯ 「11-1」のところの**６つめ**のプログラムを，
コピー，貼り付け

ニューラルネットワークの学習を行うプログラム

```
60000/60000 [==============================] - 1s 22us/step - loss: 0.6131
Epoch 46/50
60000/60000 [==============================] - 1s 22us/step - loss: 0.6116
Epoch 47/50
60000/60000 [==============================] - 1s 22us/step - loss: 0.6160
Epoch 48/50
60000/60000 [==============================] - 1s 22us/step - loss: 0.6264
Epoch 49/50
60000/60000 [==============================] - 1s 22us/step - loss: 0.6164
Epoch 50/50
60000/60000 [==============================] - 1s 23us/step - loss: 0.6192
<keras.callbacks.History object at 0x7ffaa15d80f0>
>>>
```

学習にはしばらく時間がかかる.
（大人数が一斉に行うので，終わるまで待ってください）

⑰ 「11-1」のところの**7つめ**のプログラムを，コピー，貼り付け

ニューラルネットワークを使ってみる

```
>>> m.predict( X_test[0:1] )
array([[  3.63615728e-22,   6.20049770e-16,   3.22705318e-07,
          3.28984441e-07,   2.59300546e-18,   4.37388931e-10,
          3.32093604e-25,   9.99999285e-01,   2.03745702e-17,
          1.31196898e-12]], dtype=float32)
>>>
```

１０個の数字.
それぞれ，0, 1, 2, 3, 4, 5, 6, 7, 8, 9 であると予想される確率

上の結果では「7」が高い．乱数を使っているので，各自，結果が異なる.

⑱「11-1」のところの**８つめ**のプログラムを，コピー，貼り付け

正解表示

```
>>> print( y_test[0:1] )
[7]
>>>
```

上のように「7」が表示された場合，
正解は「7」だったという意味

11-2 强化学习

機械学習

ノート
ページ

- **与えられたデータ（教師データ）**を使い，**未知のデータに対しても**当てはまるパターンや規則を，コンピュータが抽出すること

ニューラルネットワークなど，機械学習を可能にする，多数の技術がある

強化学習

- **強化学習**は，機械学習の一種.
- コンピュータは一連の行動を行う

- 行動の結果に対して，人間は成功か失敗かなどを教える（報酬ともいう）
- 行動の途中で，報酬を与えることはしない

- そこから，コンピュータは，行動に至る過程について学習する.
- 行動を何度も繰り返すことによって，学習する

PyBarain

- 多数の機械学習の機能あり

  SVM, PCA, K-Measn Clustering, SOM,

  強化学習，探索，最適化

  ニューラルネットワーク(RBM, RNN, LSTM),

- 使いやすさに重点を置いて作成された

① 「**11-2**」のところの**１つめ**のプログラムを，
コピー，貼り付け

強化学習による迷路脱出プログラム.
「下に行くべき」のスコア算出

```
[[ 1.    1.    1.    1.    1.    1.    1.    1.    1.   ]
 [ 1.    0.94  6.68  1.    3.96  6.92  7.36  7.97  1.   ]
 [ 1.    6.59  6.86  1.    0.93  7.1   1.    8.13  1.   ]
 [ 1.    6.82  6.98  1.    6.56  5.64  1.    8.32  1.   ]
 [ 1.    4.01  7.29  1.    6.43  1.    1.    8.43  1.   ]
 [ 1.    0.9   5.16  5.96  5.05  1.87  1.    8.48  1.   ]
 [ 1.    1.    1.    1.    1.    1.    1.    8.75  1.   ]
 [ 1.    1.05  0.98  2.69  2.54  6.13  5.73  1.    1.   ]
 [ 1.    1.    1.    1.    1.    1.    1.    1.    1.   ]]
>>>
```

① 「**11-2**」のところの**２つめ**のプログラムを，
コピー，貼り付け

強化学習による迷路脱出プログラム.
「右に行くべき」のスコア算出

```
[[ 1.    1.    1.    1.    1.    1.    1.    1.    1.  ]
 [ 1.    1.54  2.91  1.    6.31  6.4   6.51  5.19  1.  ]
 [ 1.    5.2   2.53  1.    0.93  2.69  1.    5.1   1.  ]
 [ 1.    5.03  2.94  1.    6.15  5.68  1.    6.28  1.  ]
 [ 1.    0.87  1.1   1.    4.28  1.    1.    4.78  1.  ]
 [ 1.    5.78  5.91  6.    3.84  0.89  1.    5.83  1.  ]
 [ 1.    1.    1.    1.    1.    1.    1.    5.68  1.  ]
 [ 1.    5.43  6.22  6.68  7.06  7.07  6.87  1.    1.  ]
 [ 1.    1.    1.    1.    1.    1.    1.    1.    1.  ]]
>>>
```

- 「下に行くべき」のスコア

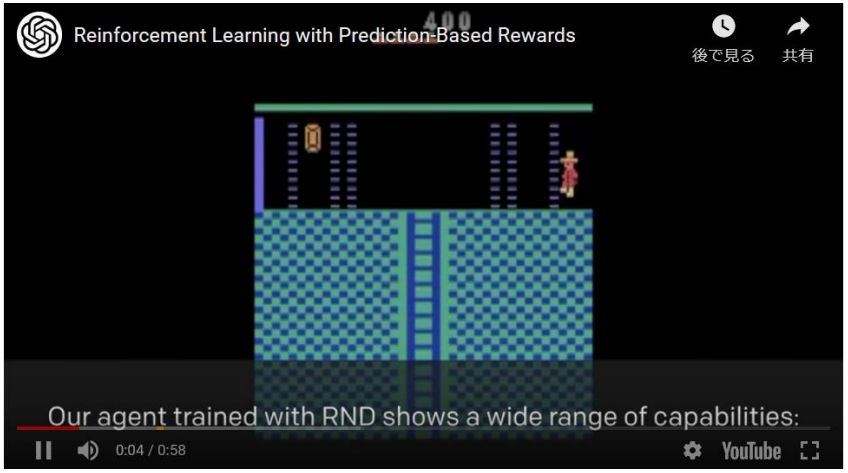
```
[[ 1.    1.    1.    1.    1.    1.    1.    1.    1.   ]
 [ 1.    0.94  6.68  1.    3.96  6.92  7.36  7.97  1.   ]
 [ 1.    6.59  6.86  1.    0.93  7.1   1.    8.13  1.   ]
 [ 1.    6.82  6.98  1.    6.56  5.64  1.    8.32  1.   ]
 [ 1.    4.01  7.29  1.    6.43  1.    1.    8.43  1.   ]
 [ 1.    0.9   5.16  5.96  5.05  1.87  1.    8.48  1.   ]
 [ 1.    1.    1.    1.    1.    1.    1.    8.75  1.   ]
 [ 1.    1.05  0.98  2.69  2.54  6.13  5.73  1.    1.   ]
 [ 1.    1.    1.    1.    1.    1.    1.    1.    1.   ]]
>>>
```

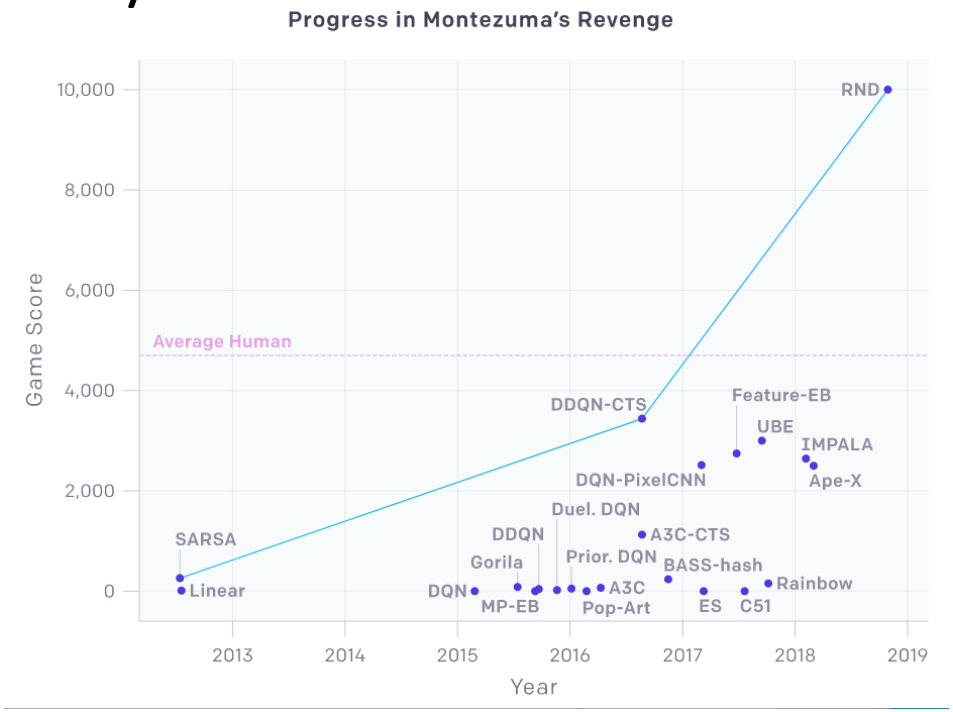
- 「右に行くべき」のスコア

```
[[ 1.    1.    1.    1.    1.    1.    1.    1.    1.   ]
 [ 1.    1.54  2.91  1.    6.31  6.4   6.51  5.19  1.   ]
 [ 1.    5.2   2.53  1.    0.93  2.69  1.    5.1   1.   ]
 [ 1.    5.03  2.94  1.    6.15  5.68  1.    6.28  1.   ]
 [ 1.    0.87  1.1   1.    4.28  1.    1.    4.78  1.   ]
 [ 1.    5.78  5.91  6.    3.84  0.89  1.    5.83  1.   ]
 [ 1.    1.    1.    1.    1.    1.    1.    5.68  1.   ]
 [ 1.    5.43  6.22  6.68  7.06  7.07  6.87  1.    1.   ]
 [ 1.    1.    1.    1.    1.    1.    1.    1.    1.   ]]
>>>
```

ニュース：強化学習は，ビデオゲーム Montezuma's Revenge で，人間の平均を上まわるスコアを出した（2018年12月）

https://openai.com/blog/reinforcement-learning-with-prediction-based-rewards/

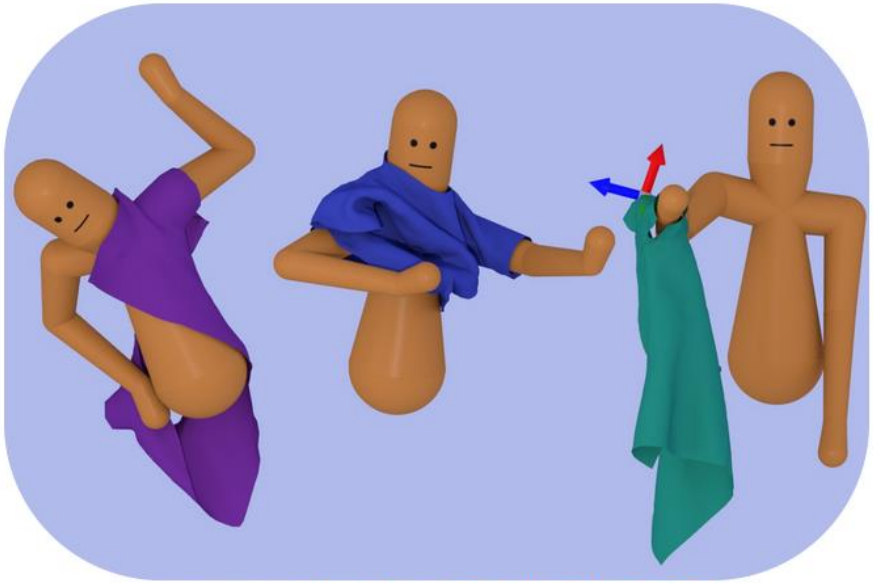




強化学習での Montezuma's Revenge スコアの進展

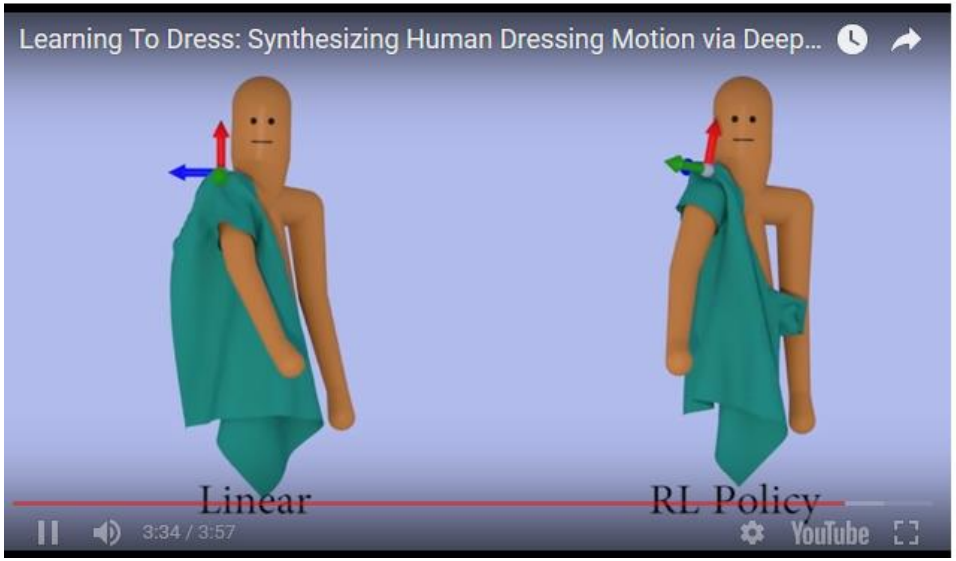
強化学習により，人間の着替え動作のシミュレーションに成功（２０１８年）

https://www.cc.gatech.edu/~aclegg3/projects/LearningToDress.html



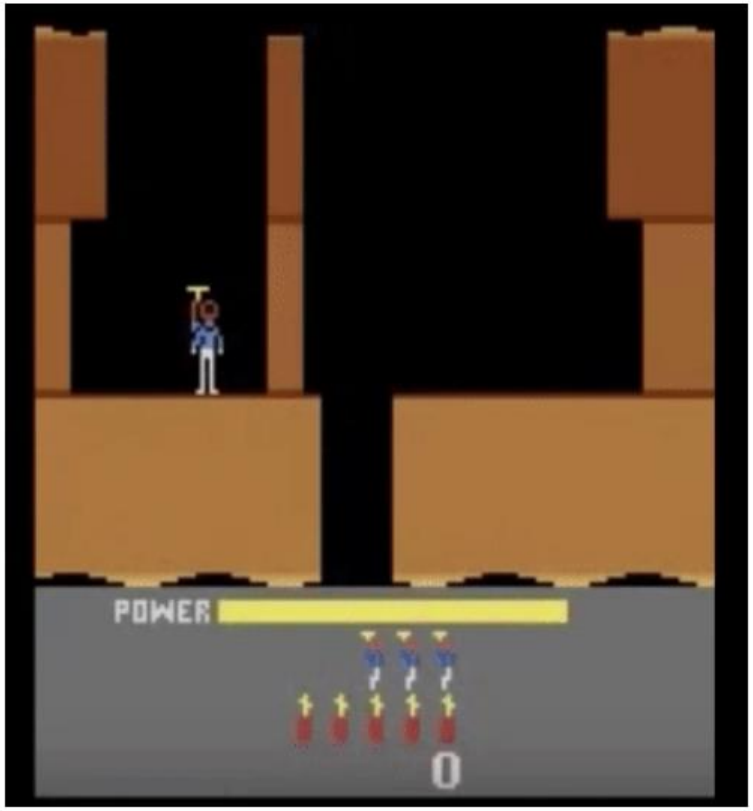
Alexander Clegg, Wenhao Yu, Jie Tan, C. Karen Liu, Greg Turk

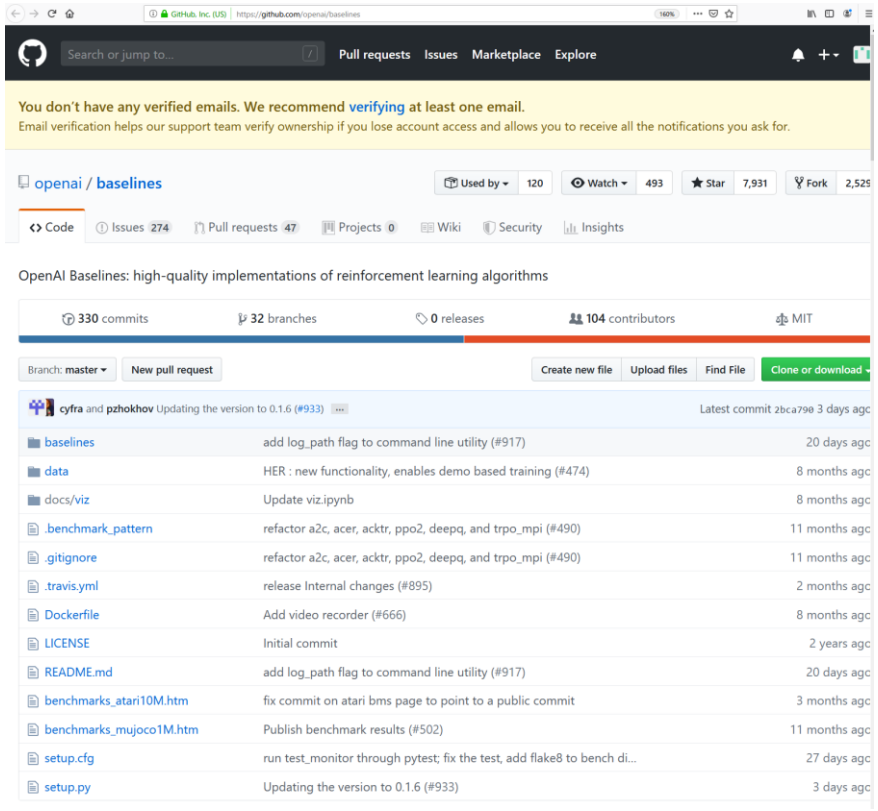
SIGGRAPH Asia 2018 | Preprint



人工知能の団体 OpenAI が，深層学習のプログラム・ソースコードを公開

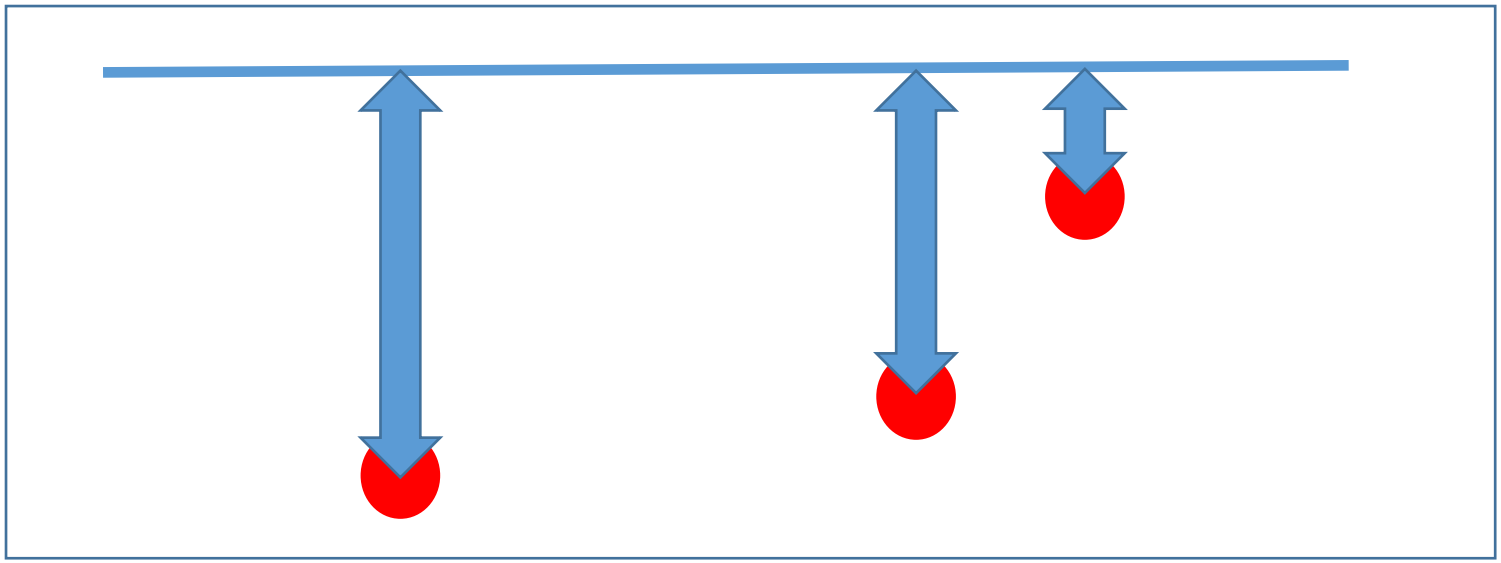
https://github.com/openai/baselines



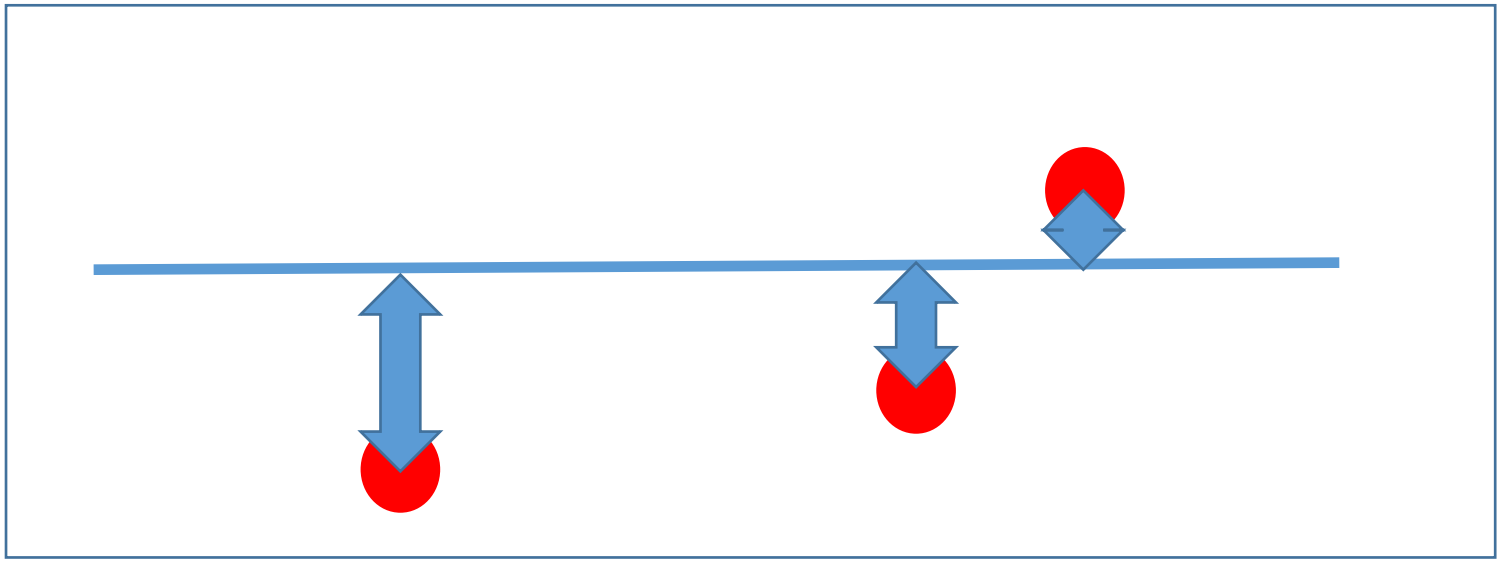


11-3 最適化の例

直線の上下移動による誤算の変化



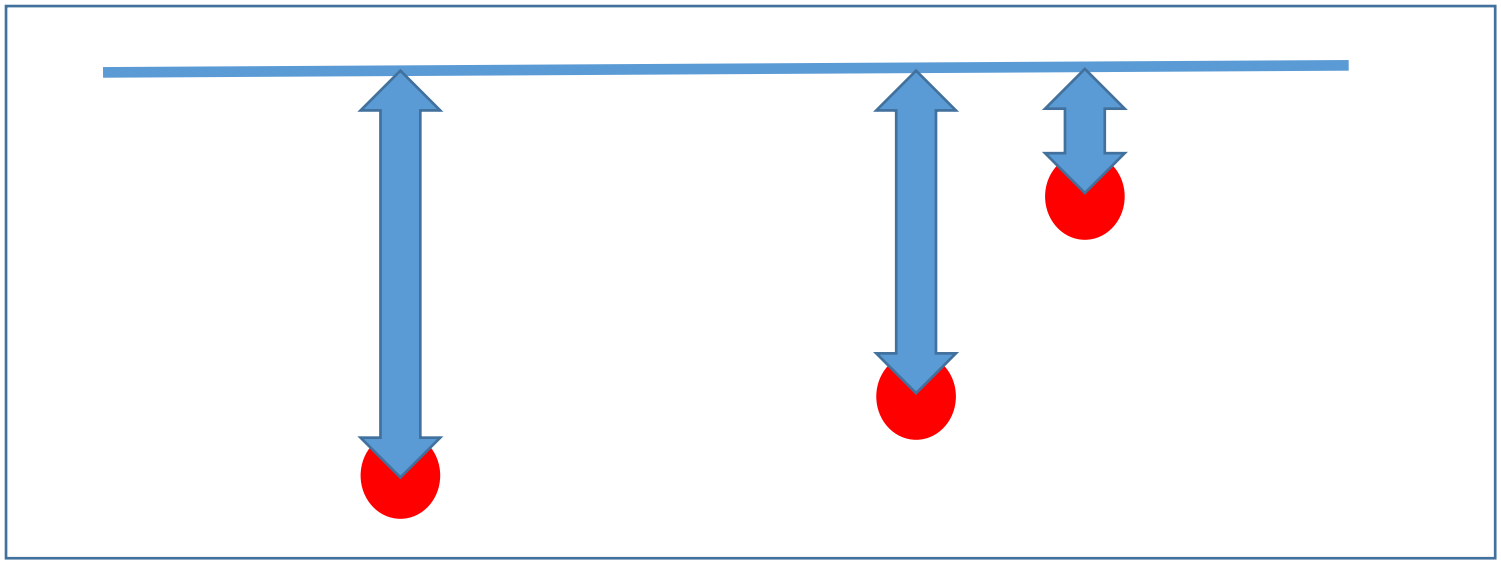
誤差大



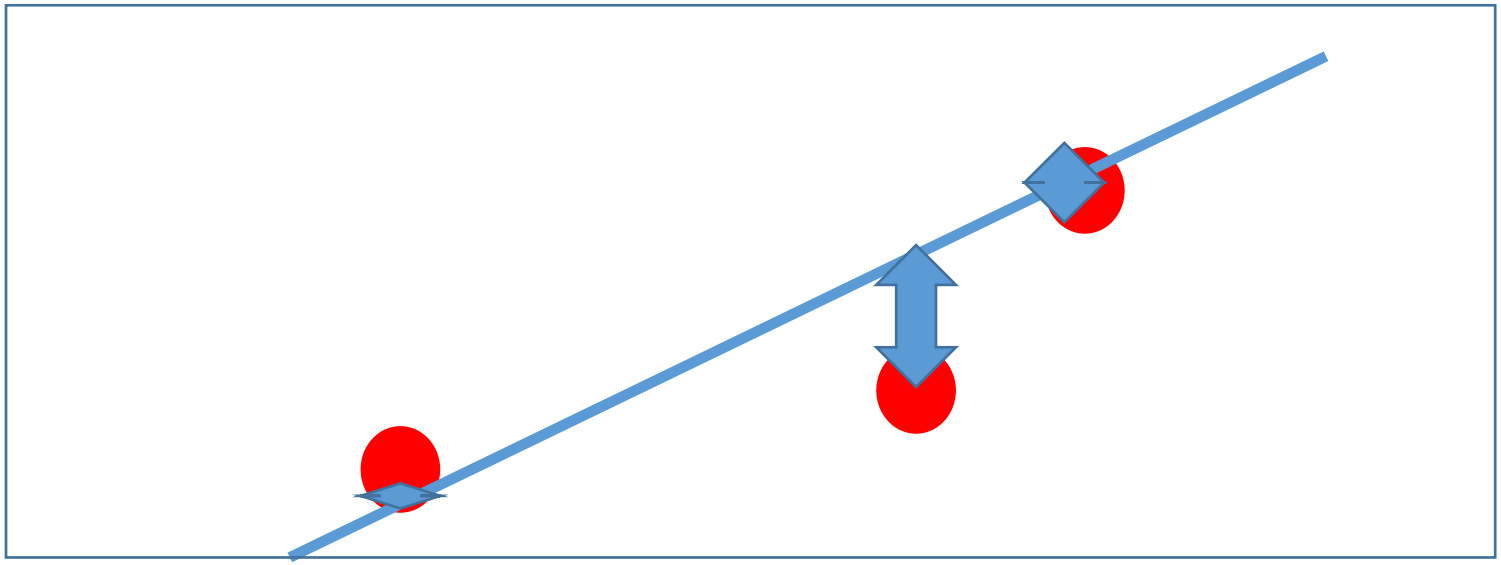
誤差小

赤点：元データ

直線の傾きの変化による誤算の変化



誤差大



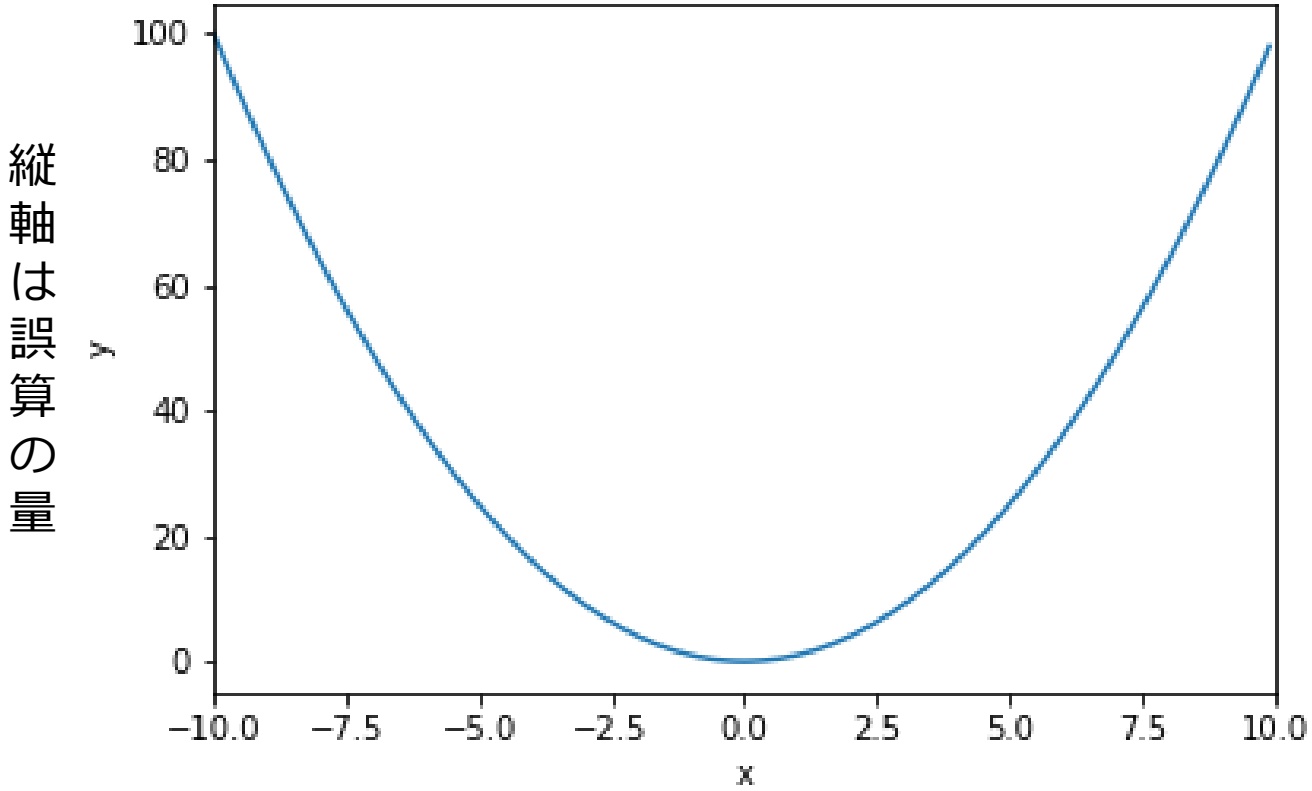
誤差小

赤点：元データ

誤差の変化

- 直線の**パラメータの変化**

  直線の上下移動や，傾きの変化により，**誤差**が変化



最適化

- ある**ゴール**を最小にするように，
  パラメータを調整すること

ゴール：　誤差

パラメータ：　直線の上下の位置と，
直線の傾き

→　教師データにフィットする
最適な線分が求まる

ノートページ

最適化の例

- 次の式が最小になるように，x の値を定めなさい．
  但し，$N = 5$ とする．

$$f(\mathbf{x}) = \sum_{i=2}^{N} 100\left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2.$$

$N = 5$ なので，x は，サイズ5 の 1次元配列である

- https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html

①「11-3」のところのプログラムを，コピー，貼り付け

最適化の例

```
>>> import numpy as np
>>> from scipy.optimize import minimize
>>> def rosen(x):
...     """The Rosenbrock function"""
...     return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)
...
>>> x0 = np.array([1.3, 0.7, 0.8, 1.9, 1.2])
>>> res = minimize(rosen, x0, method='nelder-mead',
...     options={'xtol': 1e-8, 'disp': True})
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 339
         Function evaluations: 571
>>> print(res.x)
[ 1.  1.  1.  1.  1.]
>>>
```

x = [1 1 1 1 1] のとき（すべての値が 1のとき）
最適であると求まった.

11-4 種々の例

訓練（学習）データに応じた機械学習プログラムの自動生成も


```
from tpot import TPOTClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load iris dataset
iris = load_iris()

# Split the data
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
                                                    train_size=0.75, test_size=0.25)
# Fit the TPOT classifier
tpot = TPOTClassifier(verbosity=2, max_time_mins=2)
tpot.fit(X_train, y_train)

# Export the pipeline
tpot.export('tpot_iris_pipeline.py')
```

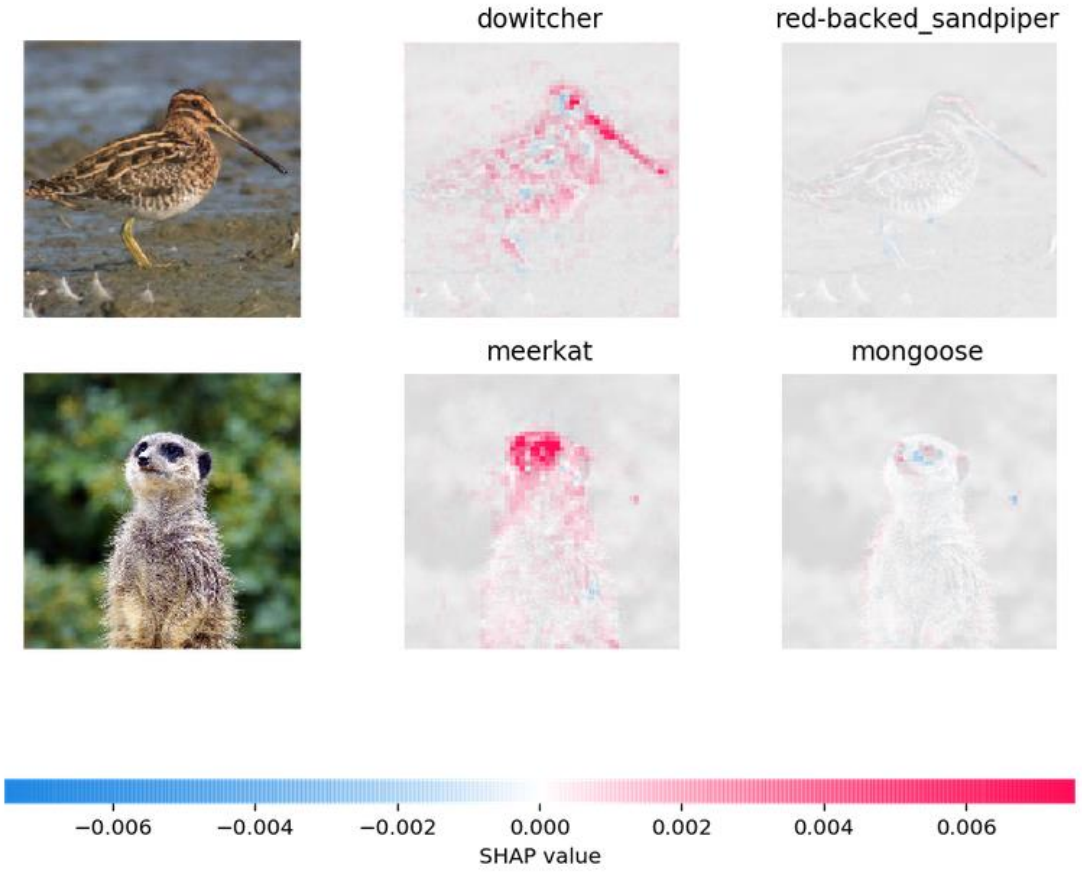
```
import numpy as np
import pandas as pd
from sklearn.kernel_approximation import RBFSampler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.tree import DecisionTreeClassifier

# NOTE: Make sure that the class is labeled 'target' in the data file
tpot_data = pd.read_csv('PATH/TO/DATA/FILE', sep='COLUMN_SEPARATOR', dtype=np.float64)
features = tpot_data.drop('target', axis=1).values
training_features, testing_features, training_target, testing_target = \
            train_test_split(features, tpot_data['target'].values, random_state=None)

# Average CV score on the training set was:0.9913043478260869
exported_pipeline = make_pipeline(
    RBFSampler(gamma=0.25),
    DecisionTreeClassifier(criterion="entropy", max_depth=10, min_samples_leaf=3, min_samples_split=14)
)

exported_pipeline.fit(training_features, training_target)
results = exported_pipeline.predict(testing_features)
```

機械学習プログラムを
生成するためのプログラム

生成された
機械学習プログラム

https://pypi.org/project/TPOT/

画像の「どの部分」を使って，予測値を高めたのかをプロットする技術



https://github.com/slundberg/shap#sample-notebooks

https://pypi.org/project/shap/

機械学習で問題となる，データの乱れを洗浄するソフトウエア

10	JaMES	M$$ax%%well	875	taco
11	Isaac	Newton	992	pasta
12	Emmy%%	Nöether$	234	pasta
13	Max!!!	Planck!!!	111	hamburguer
14	Fred	Hoy&&&le	553	pizzza

10	james	maxwell	875	taco
11	isaac	newton	992	pasta
12	emmy	noether	234	pasta
13	max	planck	111	hamburguer
14	fred	hoyle	553	pizza

- https://pypi.org/project/optimuspyspark/