

確率と最適化

Google Colaboratoryを
使ってみよう

Google Colaboratoryとは

- ✓ 機械学習の教育と研究を促進するために Google が開発したツール
- ✓ 無償でクラウド上のマシンを利用可能
- ✓ ブラウザベース
- ✓ 環境構築不要でフレームワークが使える
- ✓ GPUが利用できる



Sample.ipynb ☆

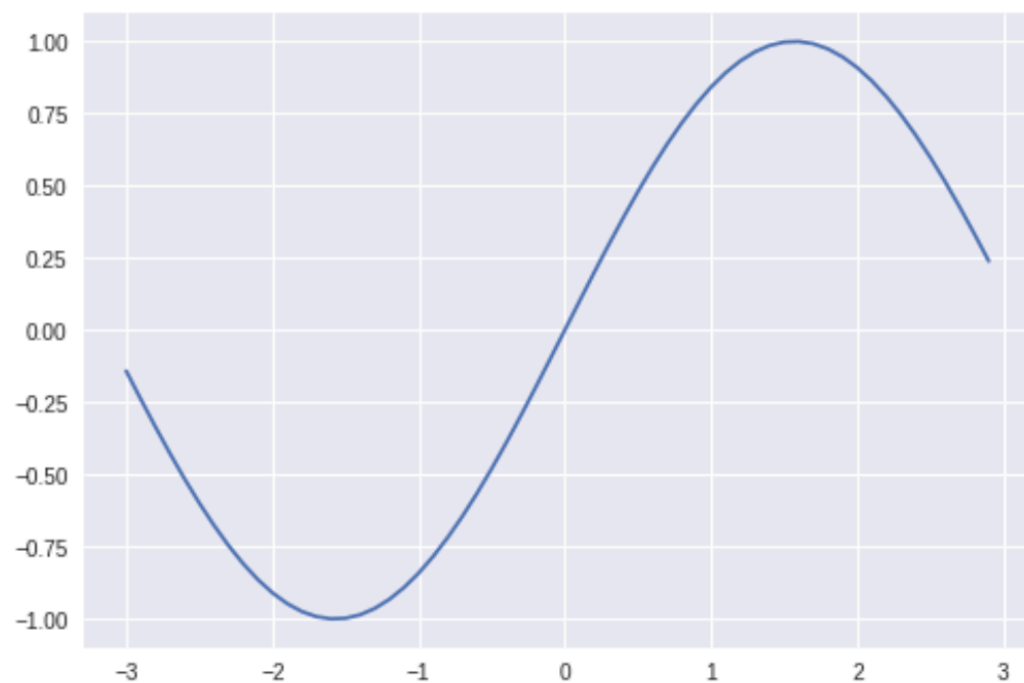
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ

+ コード + テキスト ↑ セル ↓ セル

```
[3] import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-3, 3, 0.1)
y = np.sin(x)
plt.plot(x, y)
```

↳ [<matplotlib.lines.Line2D at 0x7fdea61a9470>]



利用できる環境

- ✓ 必要なものはブラウザとgoogleアカウントのみ
- ✓ Python 2.6/3.6 が利用可能
- ✓ 機械学習に必要なライブラリはだいたい
プリインストール済み
- ✓ pipなどを利用して自由にライブラリを導入,
更新可能

使ってみよう

ブラウザで下記のURLにアクセスする

<https://colab.research.google.com/>

または講義ホームページの同リンクをクリック
windowsユーザはchromeを使ってください。
macユーザはsafariでOK

PYTHON 3の新しいノートブックをクリック

Colaboratory へようこそ

Colaboratory は、機械学習の教育や研究の促進を目的とした Google 研究プロジェクトです。完全にクラウドで実行される Jupyter ノートブック環境なしにご利用いただけます。

Colaboratory ノートブックは [Google ドライブ](#) に保存され、Google ドキュメントや Google スプレッドシートと同じように共有できます。Colaboratory は無料です。

例

最近のノートブック

GOOGLE ドライブ

GITHUB

ノートブックを絞り込む



タイトル

最初に開いた日
時

最終閲覧



Hello, Colaboratory

2018年2月6日

0 分前



Untitled7.ipynb

15 分前

15 分前



backprop.ipynb

2018年5月9日

2018年5月11日



Untitled6.ipynb

2018年5月9日

2018年5月9日



TISTA_sample.ipynb

2018年3月7日

2018年5月9日



PYTHON 3 の新しいノートブック



キャンセル

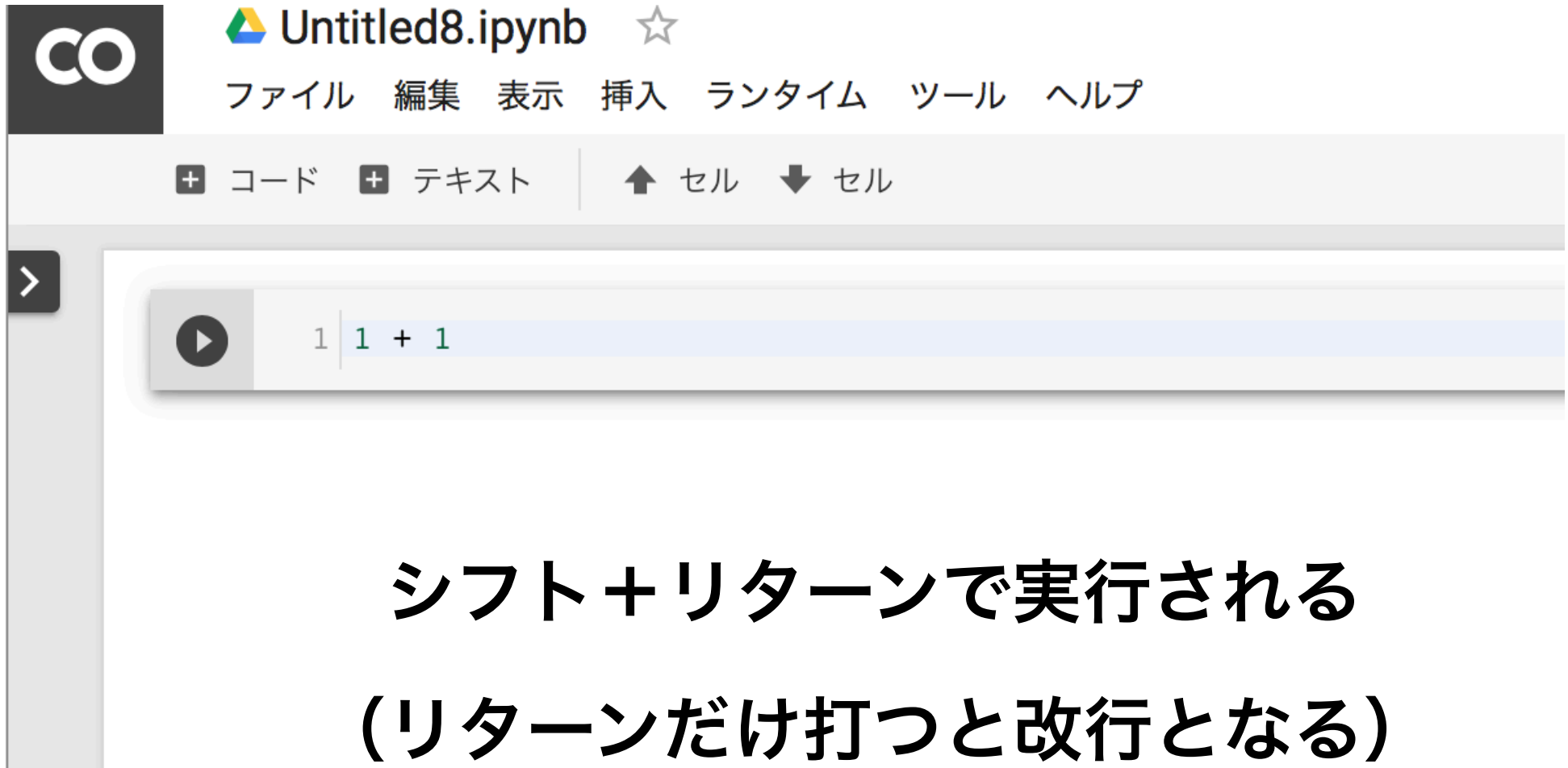
```
2 import numpy as np
3
4 with tf.Session():
5     input1 = tf.constant(1.0, shape=[2, 3])
```

新しいノートブックが開かれる

プルダウンメニュー



セルに入力してみる





Untitled8.ipynb ☆

ファイル 編集 表示 挿入 ランタイム ツール ヘルプ

+ コード + テキスト ↑ セル ↓ セル

[1] 1 | 1 + 1

↳ 2

[2] 1 | 10 * 3.14159

↳ 31.4159



1 |

for 文を使ってみる



Untitled8.ipynb ☆

ファイル 編集 表示 挿入 ランタイム ツール ヘルプ

+ コード + テキスト ↑ セル ↓ セル

```
[1] 1 1 + 1
```

```
↳ 2
```

```
[2] 1 10 * 3.14159
```

```
↳ 31.4159
```



```
1 for i in range(5):  
2     print(i)
```

```
↳ 0  
1  
2  
3  
4
```

演習問題



1から100までの数の総和を計算するプログラムを作成し、実行せよ

演習問題の解答

```
[5] 1 sum = 0
      2 for i in range(101):
      3     sum = sum + i
      4 print(sum)
```

☞ 5050

ファイル名の変更



Untitled8.ipynb ☆

ファイル 編集 表示 挿入 ランタイム ツール ヘルプ

ドライブで探す

Playground モードで開く

Python 2 の新しいノートブック

Python 3 の新しいノートブック

ノートブックを開く...

⌘/Ctrl+O

ノートブックをアップロード...

名前を変更...

ゴミ箱に移動

ドライブにコピーを保存...

コピーを GitHub Gist として保存...

GitHub にコピーを保存...

保存

⌘/Ctrl+S

変更内容を保存して固定

⌘/Ctrl+M S

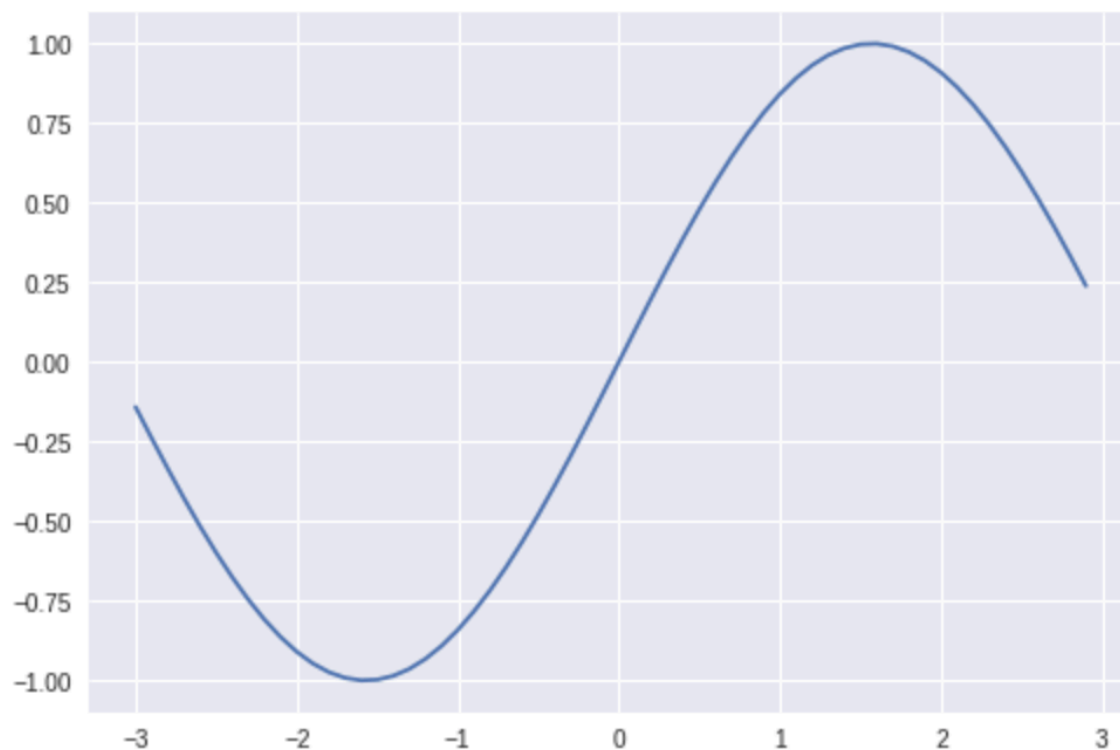
変更履歴

図をプロットしてみる

```
[8] 1 import numpy as np
     2 import matplotlib.pyplot as plt
     3
     4 x = np.arange(-3, 3, 0.1)
     5 y = np.sin(x)
     6 plt.plot(x,y)
```

numpy をimport
matplotlibを
import

☞ [<matplotlib.lines.Line2D at 0x7fa8c42af3c8>]



演習問題



下記のページをみてpythonの基本文法に関して例を試してみよ。

Python3基礎文法

<https://qiita.com/Fendo181/items/a934e4f94021115efb2e>

講義ホームページにリンクがあります

if, while, 関数の定義あたりまでで十分
(余裕のある人は当然もう少し先まで
どんどん試してみるべし)

最小化の手法: 勾配法 (gradient descent method)

制約無し最適化問題

$$\text{minimize}_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \quad (11)$$

勾配法のステップ

Step 1 (初期点設定) $\mathbf{x} := \mathbf{x}_0$

Step 2 (勾配の計算) $\mathbf{g} := \nabla f(\mathbf{x})$

Step 3 (探索点更新) $\mathbf{x} := \mathbf{x} - \alpha \mathbf{g}$ (α は学習係数)

Step 4 (反復) Step 2 に戻る

(注)

- ▶ $\nabla f(\mathbf{x})$ は勾配ベクトル (gradient vector) である。例えば、 $f(x_0, x_1)$ の場合、

$$\nabla f(x_0, x_1) = \left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1} \right)^T$$

である。勾配ベクトルは f の等高線に直交する。

勾配法のコード

$x^2 + y^2$ の最小化

```
1 def grad_x(x):  
2     return 2.0 * x  
3 def grad_y(y):  
4     return 2.0 * y  
5 xt = 5  
6 yt = 5  
7 alpha = 0.1  
8 for i in range(50):  
9     xt = xt - alpha * grad_x(xt)  
10    yt = yt - alpha * grad_y(yt)  
11    print (i, xt, yt)
```

講義HPの[gradient_descent.ipynb](#)をクリック
「このプログラムgoogleが作ったものではありません。。」
などと言われるが構わずOKして進める

演習問題

- ✓ ページのコードを実行してみよ。
- ✓ 学習率の値を変えて探索点の振る舞いを観察せよ
- ✓ 初期点を変えて実行してみよ。
- ✓ $(x - 1)^2 + (y - 1)^2$ 最小化を行え

PyTorchの概要

ライティングpytorch入門

<https://qiita.com/sh-tatsuno/items/42fccff90c98103dffc9>

(この部分の説明は完全に理解する必要はありません。だいたいの雰囲気を感じ取ってください。)

```
[17] 1 | x = torch.zeros(5, 3).
```

```
[18] 1 | print(x).
```



```
0  0  0
0  0  0
0  0  0
0  0  0
0  0  0
[torch.FloatTensor of size 5x3]
```

```
[19] 1 | y = torch.rand(5, 3).
```

```
[20] 1 | print(y).
```



```
0.5924  0.1891  0.3995
0.5054  0.4481  0.6360
0.5679  0.2776  0.2153
0.9880  0.2935  0.1306
0.1766  0.8389  0.3627
[torch.FloatTensor of size 5x3]
```

```
[21] 1 | z = 2 * y
```

```
[22] 1 | print(z).
```



```
1.1847  0.3782  0.7989
1.0109  0.8962  1.2720
1.1358  0.5552  0.4305
1.9760  0.5871  0.2613
0.3533  1.6777  0.7255
[torch.FloatTensor of size 5x3]
```

ニューラルネットワーク 学習プログラム



AND関数を学習するプログラム

講義HPのbackprop.ipynbをクリック

PyTorchのインストール



```
1 # http://pytorch.org/
2 from os import path
3 from wheel.pep425tags import get_abbr_impl, get_impl_ver, get_abi_tag
4 platform = '{}{}-{}'.format(get_abbr_impl(), get_impl_ver(), get_abi_tag())
5
6 accelerator = 'cu80' if path.exists('/opt/bin/nvidia-smi') else 'cpu'
7
8 !pip install -q http://download.pytorch.org/whl/{accelerator}/torch-0.3.0.post1
9 import torch
```

▼ 必要なパッケージのインポート

```
[ ] 1 import torch
    2 from torch.autograd import Variable
    3 import torch.nn as nn # ネットワーク構築用
    4 import torch.optim as optim # 最適化関数
    5 import torch.nn.functional as F # ネットワーク用の様々な関数
    6
```

▼ グローバル変数の定義

```
[ ] 1 mb_size = 10
```

▼ ニューラルネットワークの定義

```
[ ] 1
2 class Net(nn.Module):
3     def __init__(self):
4         super(Net, self).__init__()
5         self.fc1 = nn.Linear(2, 2) # 名前はlinear だが  $Ax + b$  のアフィン変換の形
6         self.fc2 = nn.Linear(2, 2)
7
8     def forward(self, x):
9         x = F.sigmoid(self.fc1(x))
10        x = F.sigmoid(self.fc2(x))
11        return x
12
```

▼ インスタンス作成

```
[ ] 1 model = Net() # モデルインスタンス作成
2
3 # Loss関数の指定
4 loss_func = nn.MSELoss()
5
6 # Optimizerの指定
7 optimizer = optim.Adam(model.parameters(), lr=0.1)
8
9 # データ全てのトータルロス
10 running_loss = 0.0
```

▼ 訓練ループ

```
[ ] 1 # 訓練ループ
2 for i in range(1000):
3     # フィードするデータの作成
4     inputs = torch.bernoulli(0.5 * torch.ones(mb_size, 2)) # 確率0.5で1となるベルヌーイ分布
5     labels = torch.Tensor(mb_size, 2)
6     for j in range(mb_size):
7         if (inputs[j, 0] == 1.0) and (inputs[j, 1] == 1.0):
8             labels[j, 0] = 1.0
9             labels[j, 1] = 0.0
10        else:
11            labels[j, 0] = 0.0
12            labels[j, 1] = 1.0
13
14    # Variableに変形(モデルに入力するときはVariableにする)
15    inputs, labels = Variable(inputs), Variable(labels)
16
17    optimizer.zero_grad() # optimizerの初期化
18
19    outputs = model(inputs) # 推論計算
20    loss = loss_func(outputs, labels) # 損失関数の定義
21
22    loss.backward() # バックプロパゲーション
23    optimizer.step() # パラメータ更新
24
25    # ロスの表示
26    # print statistics
27    running_loss += loss.data[0]
28    if i % 10 == 9: # print every 10 mini-batches
29        print('[%5d] loss: %.3f' %
30              (i + 1, running_loss / 10))
31        running_loss = 0.0
```


演習問題

 コードを実行してみよう

損失関数値がどんどん小さくなっていく！



```
↳ [ 10] loss: 0.203
    [ 20] loss: 0.192
    [ 30] loss: 0.134
    [ 40] loss: 0.147
    [ 50] loss: 0.103
    [ 60] loss: 0.078
    [ 70] loss: 0.041
    [ 80] loss: 0.030
    [ 90] loss: 0.026
    [100] loss: 0.017
    [110] loss: 0.014
    [120] loss: 0.010
    [130] loss: 0.010
    [140] loss: 0.008
    [150] loss: 0.006
    [160] loss: 0.006
    [170] loss: 0.005
    [180] loss: 0.005
    [190] loss: 0.004
    [200] loss: 0.004
    [210] loss: 0.003
    [220] loss: 0.003
    [230] loss: 0.003
    [240] loss: 0.003
    [250] loss: 0.002
    [260] loss: 0.002
    [270] loss: 0.002
    [280] loss: 0.002
    [290] loss: 0.002
    [300] loss: 0.002
    [310] loss: 0.002
    [320] loss: 0.002
    [330] loss: 0.002
    [340] loss: 0.002
    [350] loss: 0.002
    [360] loss: 0.002
    [370] loss: 0.002
    [380] loss: 0.002
    [390] loss: 0.002
    [400] loss: 0.002
    [410] loss: 0.002
    [420] loss: 0.002
    [430] loss: 0.002
    [440] loss: 0.002
    [450] loss: 0.002
    [460] loss: 0.002
    [470] loss: 0.002
    [480] loss: 0.002
    [490] loss: 0.002
    [500] loss: 0.002
```

学習結果の評価

```
[7] 1 # 性能の評価
    2 inputs = Variable(torch.Tensor(1, 2))
    3 inputs.data[0, 0] = 0.0
    4 inputs.data[0, 1] = 0.0
    5
    6 outputs = model(inputs)
    7 print('0 & 0 = %.4f' % (outputs.data[0, 0]))
    8
    9 inputs.data[0, 0] = 0.0
   10 inputs.data[0, 1] = 1.0
   11 outputs = model(inputs)
   12 print('0 & 1 = %.4f' % (outputs.data[0, 0]))
   13
   14 inputs.data[0, 0] = 1.0
   15 inputs.data[0, 1] = 0.0
   16 outputs = model(inputs)
   17 print('1 & 0 = %.4f' % (outputs.data[0, 0]))
   18
   19 inputs.data[0, 0] = 1.0
   20 inputs.data[0, 1] = 1.0
   21 outputs = model(inputs)
   22 print('1 & 1 = %.4f' % (outputs.data[0, 0]))
   23
```

```
↳ 0 & 0 = 0.0001
    0 & 1 = 0.0067
    1 & 0 = 0.0051
    1 & 1 = 0.9809
```

演習問題

-  学習回数が50回の場合はどうなるだろうか
(注：「インスタンス作成」のセルから再実行する必要がある)
-  OR関数を学習するように改造してみよ

MNIST文字認識

講義HPのmnist.ipynbをクリック

- ✓ パラメータをさわってみよう
- ✓ 改造してみよう

興味を持った人は 自分でやってみよう

- ✓ フルスクラッチで書くのは結構大変だが、人の書いたソースコード(PyTorch, TensorFlow, Keras)がいっぱいネットにはあるので、最初はそれを有り難く使わせていただく
- ✓ 人のコードを読んだり、改造したりしているうちに使えるようになる