November 7, 2019

物理研究者向けのニューラルネットワーク入門 (実践・ハンズオン)

富谷 昭夫 [1,*]

[1] *RIKEN/BNL Research center, Brookhaven National Laboratory, Upton, NY, 11973, USA*

Abstract

この講義ノートは、物理学を勉強/研究している者に向けたニューラルネットワークの解説である。こちらのノートは、ハンズオンセッションで必要となる Python の入門である。

*akio.tomiya@riken.jp

Contents

Hands on session

実行テスト環境

1. macOS Mojave 10.14.5
2. Python 3.7.1
3. Numpy 1.15.1
4. Matplotlib 3.0.0
5. Jupyter 4.4.0
6. Chainer 4.5.0
7. Chrome

I. PYTHON 入門

Perl などと同様のスクリプト言語。

1. 歴史

1. 1991 年に G. ヴァンロッサムが Python 0.90
2. 2019/10 月末時点で 3.8.0。
3. ver 2.x と 3 は破壊的な改変。2.7 のサポートは 2020 年 1 月 1 日

A. Python の基礎事項

v2 系と v3 系の大きな違い:

ソースコード 1: version 2

```
1    print "Hello world"
```

ソースコード 2: version 3

```
1    print ("Hello world")
```

なお print は、改行が最後に自動的に入る。改行を抑制するには、print("hoge", end="") の様に末端文字を「なし」にしておく。さらに、print 関数は引数を並べれば自動的に並べて表示する。

ソースコード 3: code

```
st = "hoge"
print (st, "fuga", "piyo")
# This is a comment!
```

またコメントは、# でできる[1]。

1. インデントでブロック

次で例を見る。Python 3 では、インデントにタブとスペースを混ぜることを禁止している。インデントは 4 つのスペースが主流。

2. *for* 文

Python での for 文は、他の言語の foreach に対応する。たとえば C 言語でのループ文は、

ソースコード 4: C 言語の for

```
for(int ii=0; ii<10; ii+=1){
    printf("%d clang\n",ii);
    }
```

とかけるが一方で Python では、

ソースコード 5: python の for

```
for ii in range(10):
    print(ii, "python")
```

となる。

range(N) は、0 から N 未満の数の range 型のオブジェクトを返す。for はそれを受け取って逐次的に処理する。

ソースコード 6: code

```
ar = range(10)
print(ar)
```

[1] 複数行コメントを行う方法もあるがブロックが絡むと複雑であるのでここでは紹介しない。

さらに、ループ変数が 2 ずつ進む for 文は下記で書ける。

ソースコード 7: code

```
for ii in range(1,10,2):
    print(i,"a")
```

これは、ii=1 からスタートして、2 個ずつ、10 未満までループする。

3. リスト

配列に対応するもの (オブジェクト) で四角括弧で囲んで定義する。

ソースコード 8: code

```
a = [1, 2, 3, 4]
```

リストは「append」というメソッドで要素を追加することができる。

ソースコード 9: code

```
my_list = []
print(my_list)
my_list.append(1)
print(my_list)
my_list.append(10)
print(my_list)
```

append は、for 文と合わせると便利である。

ソースコード 10: code

```
ar = [0, 43, 22, 33, -1, 0]
for ii in ar:
        print(ii)
```

またタプルという似たものもあるがこちらはイミュータブル (改変不可) なので要素を改変しようとするとエラーが出る。こちらは丸カッコで囲んで定義する。

ソースコード 11: code

```
tup = (1,2,3)
print(tup[2])
tup[2] = 1 # ここでエラー
```

4. 内包表記

下記の 2 つは同じ配列をつくる。

ソースコード 12: code

```
a = []
for ii in range(0,100,10):
        a.append(ii)
print(a)
# results -> [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

ソースコード 13: code

```
a = [x for x in range(100) if x%10 == 0]
print(a)
# results -> [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

下の例は、数学での集合の内包的記法とおなじく、条件式を並べてリストの要素を定義する[2]。

5. *if* 文

ここでは、プログラムの実行順序を変える if 文を紹介する。for 文と同様にインデントでブロックが書かれる。

ソースコード 14: code

```
if 条件 1:
    条件 1が真のときの実行文
elif 条件 2:
    条件 2が真のときの実行文
else:
    上の 2つの条件が満たされなかったときの実行文
```

何もしないという実行文は、"pass"で、ブロックに仮置することがある。

ソースコード 15: code

```
if 条件 1:
    pass
else:
    実行文
```

またこの様に、elif はなくてもいい他、else も置かなくても良い。

[2] 外延的記法とは、集合を $A = \{2, 4, 6, \cdots\}$ の様に要素をならべて定義する方法で、内包的記法とは、集合を $A = \{x | x = 2k, k \in \mathbb{Z}_{++}\}$ の様に条件から定義する方法である。

6. 整数の割り算

返り値が整数になる割算は、//で行なうことが出来る。

ソースコード 16: code

```
a = 10
a = a//2
print(a)
```

7. 関数の定義

ソースコード 17: code

```
def my_func1():
     print("hoge")

def my_func2():
     pass
     print("hoge")
     return

def my_func3():
     a = 1
     b = 2
     return a, b

def my_func4():
     a = [1,2,3]
     b = 2
     return a, b
```

my_func1 と my_func2 は、同じ機能をもつ。つまりブロック終了時に呼び出しもとの制御に戻るので return はあってもなくても良い。

次に、my_func 3 は、タプルをかえす関数で my_func 4 は、リストと変数のタプルをかえす関数である。この様に関数は、色々なオブジェクトを返すことが出来る。

次に引数がある関数を見ていこう。

ソースコード 18: code

```
def my_func5(x):
        print(x)
```

これは、第一引数を表示する関数である。

また Python の関数は変数にデフォルトの値を置くことが出来る。

ソースコード 19: code

```
def my_func6(x, y=10):
        print(x,y)

my_func6(1)
my_func6(2, 20)
```

この関数は第一引数は必須ではあるが第 2 引数は入力しなくても 10 が代入される。

8. フォーマット付きリラテル

Python3.6 以降でのフォーマット付きリラテル。中括弧で囲む。

ソースコード 20: code

```
idx=10
print(f"{idx}")
```

ソースコード 21: code

```
i = 1234
print(f'zero padding: {i:08}')
```

ソースコード 22: code

```
f = 12.3456
print(f'digit(decimal): {f:.3f}')
print(f'digit(all) : {f:.3g}')
#results -> digit(decimal): 12.346
#results -> digit(all) : 12.3
```

同様のものは、

ソースコード 23: python3.5 以前

```
print("result = {0}".format(c) )
```

9. ライブラリの呼び方

import で外部機能を呼べる。

ソースコード 24: code

```
import モジュール名
```

使うときは、

ソースコード 25: code

```
1 モジュール名.メンバ名
```

というふうにピリオドで区切る。

また、別名をつけることができる。

ソースコード 26: code

```
1 import numpy as np
```

こう呼ぶと、

ソースコード 27: code

```
1 np.zeros(10)
```

といった感じでライブラリで定義されている関数を呼べる [3]。大体デフォルトの略称が決まっているので自前で略称をつけないほうがいい。

機能のうち、1 つのみも呼べる

ソースコード 28: code

```
1 from math import pi
```

この場合、pi には、円周率が代入されている。

名前空間の衝突を避けるために、本講義では以下をつかわない。

ソースコード 29: code

```
1 from モジュール import *
```

10. その他

実数型は、倍精度が標準。また「PEP: 8」に推奨の書き方がのってる。

11. 注意事項

ループが遅いので、後述の numpy を使う。ただし要素ごとのアクセスはリストのほうが早いので場合によって使い分けるほうがいいこともある。

[3] 因みに、この命令は 10 個の要素を持つ、0 で埋められた配列 (正確には numpy array) を生成する。。

B. Python の実行環境

1. terminal から python を起動する (対話型環境)
2. terminal から ipython を起動する (対話型環境、高機能)
3. terminal から Jupyter を起動する
4. Web 上で google の colab をつかう
5. vscode をつかう (2019 October release 以降, 今日は触れない)

C. Jupyter とは

Jupyter とはもともと ipython notebook と呼ばれた、mathematica like な環境。ブラウザで起動する。Julia, Python R をサポート。発起人である Fernando Perez は、Anna Hasenfratz の学生で格子 QCD で博士号を取得したひと。

詳しくは、Wikipedia の Project Jupyter や Jupyter の公式ページ参照のこと。

D. Numpy hands-on

1. Numpy とは

Python はループが遅いので、Numpy というライブラリを使ってベクトル演算をする。ベクトルを成分でなく、ベクトルとして扱いたい。内部で BLAS が走る。

BLAS = Basic Linear Algebra Subprograms で行列やベクトルの基本的な計算をやってくれる関数群のこと。

2. Numpy のつかいかた

ソースコード 30: import

```
1 import numpy as np
```

どの BLAS が呼ばれているかは、

ソースコード 31: import

```
1 print( np.show_config() )
```

で確認可能。

3. *Numpy array* の作り方

np.array() の引数として、リスト型オブジェクトを入れるとでてくる。

ソースコード 32: code

```
numpy_arrray = np.array( [1,2,3] )
print(numpy_arrray)
```

4. *Numpy* の形

5. *Numpy* と *python loop* の比較

ソースコード 33: Python による内積

```
a = [1., 2., 3.]
b = [4., 5., 6.]
c = 0.0
for jj in range(3):
        c+=a[jj]*b[jj]
print(f"result = {c}" )
```

ソースコード 34: Numpy での内積

```
import numpy as np

a = np.array([1., 2., 3.])
b = np.array([4., 5., 6.])
c = np.dot(a,b)
print(f"result = {c}" )
```

更に多自由度、行列行列積などでは更に速度差がでる。

今回はやらないが、GPU を使って高速化も可能である。そのためには、CUDA を内部でつかう cuPy をつかう。Chainer と合わせて使う場合は殆ど隠蔽されているので気にしなくて良い[4]。

6. 配列操作

[4] Chainer のマニュアルが詳しい。また、Google colab では GPU が無料で使用可能である。

ソースコード 35: code

```
ar = np.array([0,1,2,3,4,5,6,7,8,9])
print(ar)
idx = [2,4,6,8]
ar2 = ar[idx]
print(ar2)
```

numpy 配列に index を持つリスト型オブジェクトを入れるとその番号の要素をもつ配列がピックアップされて帰ってくる。

7. *Shape* と *reshape*

numpy array は多次元配列を持てるが、その形状は shape と呼ばれる。また reshape で形をかえることが出来る。

ソースコード 36: code

```
ar = np.array([0,1,2,3,4,5,6,7,8,9])
print(ar.shape )
ar = ar.reshape(2,5)
print(ar.shape )
```

これを実行すると 10×1 の 1 次元配列が 2×5 の多次元配列に形が変わっていることがわかる。

8. スライス

numpy array は配列として特殊な切り出し方が可能でスライスと呼ばれている。

ソースコード 37: code

```
ar = np.array([0,1,2,3,4,5,6,7,8,9])
ar2 = ar[:] # すべてをとりだす(参照がとられる)
ar3 = ar[1:2] # 2番目だけの要素
ar4 = ar[0:3] # 0, 1, 2番目の要素
print(ar2)
print(ar3)
print(ar4)
```

E. matplotlib hands-on

1. *matlab* 風な書き方

ソースコード 38: code

```
import matplotlib.pyplot as plt
%matplotlib inline
# Magic command for jupyter notebook
# = = = =
x = [1, 2, 3, 4]
y = [1**2, 2**2, 3**2, 4**2]
plt.plot(x,y)
plt.show()
```

よく、list append と併用する。plot は、2 つの引数をとる。

2. オブジェクト指向的な書き方。

本講義では matlab 風な書き方を使うが、下記のような書き方もできる。

ソースコード 39: code

```
import matplotlib.pyplot as plt
%matplotlib inline
# Magic command for jupyter notebook
# = = = =
x = [1, 2, 3, 4]
y = [1**2, 2**2, 3**2, 4**2]
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x,y)
plt.show()
```

3. グラフの装飾

plt.show() の前に以下を挿入するとグラフにラベルをつけることが出来る。

ソースコード 40: code

```
plt.title("Plot")
plt.xlabel("x") # x label
plt.ylabel("y") # y label
plt.xlim([0, 5]) # x range
```

4. エラーバー付きのグラフ

ソースコード 41: code

```
import matplotlib.pyplot as plt
```

```
%matplotlib inline
# Magic command for jupyter notebook
# = = = =
x = [1, 2, 3, 4]
y = [1**2, 2**2, 3**2, 4**2]
er = [0.7, 0.7, 0.7, 0.7]
plt.errorbar(x, y, yerr = er)
plt.show()
```

F. pandas hands-on

pandas は、データを扱う高機能なフレームワークである。今回は、フィッシャーのアヤメのデータを numpy array に変換するためだけに使う。ロードは下記のように行われる。

ソースコード 42: code

```
import pandas as pd
```

アヤメのデータが下記のような csv(comma-separated values) ファイルに格納されていたとする。

ソースコード 43: iris_data.csv

```
SepalLength,SepalWidth,PetalLength,PetalWidth,Name
5.1, 3.5, 1.4, 0.2, Iris-setosa
5.0, 3.3, 1.4, 0.2, Iris-setosa
7.0, 3.2, 4.7, 1.4, Iris-versicolor
6.9, 3.1, 4.9, 1.5, Iris-versicolor
```

下記のプログラムで、x2 に SepalLength の列 (0 番目の行から 5 行目まで) が抜き出され、代入される。

ソースコード 44: code

```
iris_fl = pd.read_csv("iris_data.csv", encoding="utf-8")
x2 = iris_fl.loc[0:5, "SepalLength"]
```

引数は対象行と抽出したい列のラベルである。

下記のプログラムで、x_raw に SepalLength と SepalWidth の列 (全ての行) が抜き出され、代入される。

ソースコード 45: code

```
iris_fl = pd.read_csv("iris_data.csv", encoding="utf-8")
x_raw = iris_fl.loc[:, ["SepalLength", "SepalWidth"]]
```

1 つ目の引数はスライスと同じで全ての行を対象としていることを意味する。

G. Python でのクラス

1. クラスとは？

オブジェクトを生成するための雛形、設計図。オブジェクトとは、配列変数 (構造体) に関数を内蔵させたもの。オブジェクトはデータとふるまいが定義されている。

2. *Python* でのクラスとは？

次のように定義されて使われる。

ソースコード 46: code

```
class animal:
    def __init__(self, name):
        self.name = name
        print("animal constructor is called")
    def say_name(self):
        print(self.name, "in animal class")

pochi_obj = animal("pochi")
pochi_obj.say_name()
```

python には private 変数は無いので注意が必要。(ネームマングリングで似たものが実装できるが割愛)

3. 継承

python での継承は、以下。

ソースコード 47: code

```
class human(animal):
    def __init__(self, name):
        super().__init__(name)
        print("human constructor is called")

taro_obj = human("taro")
taro_obj.say_name()
```

class 子クラス名 (親クラス名)

継承した時に親のコンストラクタもよべる。

H. Chainer とは

機械学習ライブラリの一種で define by run (定義と実行) を導入した。define by run は、定義と計算を同時に行なうことが出来るという枠組みである。他のライブラリは基本的に define and run (定義そして実行) 形式で、計算のためのグラフを作成した後に計算を実行する。define by run だと動的にグラフを変えたりも出来るため柔軟な計算ができる。

Chainer には活性化関数、線形変換、誤差関数、逆誤差伝播、optiminzer などが実装してある。つまりそれらのパーツを呼び出すだけで簡単にニューラルネットとその学習が実装できる。

I. Chainer の機能

1. Variables ユニットに対応する自由度。
2. Links 線形変換を扱う機能で学習の対象となる。
3. Functions 学習の対象とならない活性化関数などを扱う。
4. Optimizers 学習のプロセス、つまり勾配法を実装してある。SGD、Adam などがつかえる。

Serializer と *Trainer*

本講義では扱わないが、Chainer の機能であるので紹介しておく。Serializer は、学習済みモデルの保存と呼び出しを行なう機能である。長時間・大規模の学習の後保存しておき、学習済みモデルをもちいて計算を行なう場合に使う。

Trainer は、学習の自動化を行なう機能である。ニューラルネットワークの学習を行う場合、学習のループ処理やミニバッチの処理は多くの場合共通している。これらを統一的に扱うのが Trainer である。

くわしくは、Chainer のチュートリアルページを参照して欲しい。

ここからは、ニューラルネットワークを実際に用いた計算を見ていく。

II. あやめの分類 HANDS-ON

A. 実験用のデータセット

機械学習では、下記のような代表的なデータセットが存在し、手法の確認に使われている[5]。

1. iris: 統計学者フィッシャーの論文から抽出したアヤメの分類のデータ。データは 150 個。下で詳しく述べる。
2. titanic: タイタニック号の乗客者データ。乗っていた階級や性別、生存したかの情報が入っている。データは 1300 件程度。
3. MNIST: 28×28 のグレースケールの $0, 1, \cdots, 9$ の手書き文字。ラベルもついている。学習用に 6×10^4 個、検証用に 6×10^4 個のデータをもつ。
4. Fashion MNIST: MNIST の衣服版で、衣服の 10 カテゴリーの 7×10^4 個のグレースケール画像を含む。
5. Cifer10: 10 種類のカラー画像のデータセット。

がある。

まずは Fisher のあやめのデータセットをもちいてみる。つまり、4 つの特徴量をもちいて 3 種にわける。

1. ニューラルネットワークのためのデータの扱い

Fisher のあやめとは、統計学者の Fisher の集めたアヤメのデータで花びらの幅とながさ、がくの幅と長さの 4 次元のデータと 3 種のアヤメのラベルを持つ。全データ数は 150 個。

[5] 余談だが、Kaggle という機械学習コンペがあり、賞金は 250 万円から 1000 万円程度とのこと。

a. 入力　入力は

$$\vec{x} = \begin{pmatrix} \text{花びらの幅} \\ \text{花びらの長さ} \\ \text{がくの幅} \\ \text{がくの長さ} \end{pmatrix} \in \mathbb{R}^4. \tag{1}$$

といった 4 次元ベクトルとなる。

2. one-hot vector

前述の通り、ラベルをワンホットベクトルにしておく。今の場合、分類がセトナ (setosa)、バーシクル (versicolor)、 バージニカ (virginica) の 3 種なので、3 次元の“単位ベクトル” を割り当てることにする。

$$\text{セトナ} \equiv (1,0,0) \tag{2}$$

$$\text{バーシクル} \equiv (0,1,0) \tag{3}$$

$$\text{バージニカ} \equiv (0,0,1) \tag{4}$$

としておく。

III. ISING MODEL の相転移検出

ここでは、[Juan Carrasquilla, Roger G. Melko 1605.01735] に基づき、相転移の検出にチャレンジする。

A. 2 次元イジング模型とは

まず、2 次元イジング模型は下記のハミルトニアンで与えられる統計模型である。

$$H[s] = -\sum_{<\vec{j},\vec{k}>} s_{\vec{j}} s_{\vec{k}} \tag{5}$$

$<\vec{j},\vec{k}>$ は、最近接の和を表す。更に分配関数は、

$$Z = \sum_{\{s\}} \mathrm{e}^{-\beta H[s]} \tag{6}$$

となる。この系は、臨界温度

$$\beta_{\rm cr} = \frac{1}{2}\log(\sqrt{2}+1) = 0.4406\cdots \tag{7}$$

で相転移を示すことが厳密解から知られている。

系が単純であり、また厳密解があるので新手法のテスト環境として最適である。

B. 何をどうみつけたのか

彼らは、イジング模型の配位を生成し、高温相と低温相を分類するニューラルネットワークを作製した。ニューラルネットワークの出力は 2 次元で、高温相に属しているか低温相に属しているかを判別する。一方で中間ぐらいの温度では、正解率は下がってくる。そこで彼らは、2 次元の出力を逆温度の関数としてプロットし、交差点として転移温度をみつけた。

C. 実演

ここではニューラルネットワークをもちいて、$L_x \times L_y = 32^2$ のイジング模型の相の分類を行い、相転移温度を決定する。