关于 Python 的一些说明

NumPy 简介

Python 已经成为实现深度神经网络的首选程序语言。根据我个人的使用体验，从代码的简洁和程序的高效性上来说，MATLAB 应该都优于 Python。而且，MATLAB 语言基本上匹配数学语言，尤其是线性代数方面。制约 MATLAB 普及的原因是，它是一款商业软件，而 Python 则是完全开源的。相比 MATLAB，Python 在循环上做了更多处理，以方便数据的循环迭代，而 MATLAB 通常只能通过指标方式进行数据的循环提取。

我总是假设读者熟悉 MATLAB 语言，并了解 Python 的基本常识。后面将把 MATLAB 面向对象的 (手写数字分类) 程序逐字逐句翻译成 Python 代码。为此，这里仍然介绍 Python 中 MATLAB 矩阵的替代品——NumPy 的多维数组 Ndarray。

注意，Python 的索引从 0 开始，而 MATLAB 从 1 开始。

一维数组和二维数组

NumPy 中的数组与 MATLAB 中的矩阵类似，事实上，NumPy 中也自带矩阵类型 (严格二维)，但很少使用 (后续可能删除矩阵类)。

一维数组

Python 中最频繁使用的数据类型就是列表，如 `[1,2,3]`，它可如下转化为 NumPy 中的一维数组

```
import numpy as np
a = np.array( [1,2,3] )
```

- 它的维数为 (3,)，这里括号中有一个逗号，表示该数组是一维的，且元素个数为 3；
- 一维数组不具有行列 (或轴) 的概念，因而没有转置的概念。

二维数组

- 行向量如下生成

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

  用 v.shape 可以发现其维数为 (1,3)，即 1 行 3 列。

- 列向量如下给出

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

  注意，1 和 [1] 是不同的，后者是一维数组。上面的列表中放入了三个一维数组 (元素个数为 1)，从而产生列向量。对应的维数为 (3,1).

可以看到，NumPy 中的数组与 C 或 C++ 中的数组概念是一致的，只不过后者用大括号作为容器的标志。

- 一维数组可通过如下方式转化为列向量

  ```
  a1 = a.reshape(-1,1) # 或 a1 = a.reshape([-1,1])
  ```

这里的 -1 表示根据第二个位置推测出维数，相当于 MATLAB 中的 `reshape(a,[],1)`。

类似如下转化为行向量

```
a1 = a.reshape(1,-1) # 或 a1 = a.reshape([1,-1])
```

- 二维数组可进行转置操作，例如

```
a = np.array( [1,2,3] )  # 一维数组
v = a.reshape(-1,1)      # 列向量，为二维数组
v1 = v.T                 # 或 v.transpose()：转置，结果为行向量，为二维数组
```

- 在提取二维数组的一行或一列时，默认返回的都是一维数组，这一点要特别注意。例如，

```
A = np.array( [ [1,2,3], [4,5,6],[7,8,9] ] )
a = A[:,0]    # 或 a = A[...,0], 结果为一维数组 array([1, 4, 7])
```

NumPy 中的向量
以下统称 NumPy 中的一维数组、行向量和列向量为向量。

NumPy 中的乘法

MATLAB 中的点乘

这里的点乘不是数学上的点乘，而是分量对应相乘。

- 点乘用 `A*B` 或 `np.multiply(A,B)` 实现。
- 两个相同构型的数组相乘，其乘法按 MATLAB 中矩阵的点乘理解，如

```
# 两个一维数组相乘结果为一维数组
a = np.array( [1,2,3] )
b = np.array( [-1,-2,-3] )
c = a*b     # 结果为一维数组 array([-1, -4, -9])
```

```
# 两个二维数组相乘结果为二维数组
A = np.array( [ [1,2], [3,4] ] )
B = np.array( [ [-1,-2], [-3,-4] ] )
C = A*B
# 结果为二维数组 array([[ -1,  -4],
#                       [ -9, -16]])
```

- 两个不同构型的数组相乘，其乘法通常没有意义。但当其中一个为向量时，NumPy 做了特殊处理。
  - 当向量为一维数组时，按行向量理解。一方面，数组的构造是行优先的；另一方面，点乘运算要满足可交换性。
  - 将向量“垂直于轴”进行扩展，若能和被乘对象相容，则进行扩展后的乘法。
  - MATLAB 在新版本中也增加了点乘的该功能 (至少在 2015 版没有)。

```
# 行向量乘以列向量
v = np.array( [ [1,2] ] )         # (1,2)
w = np.array( [ [1],[2],[3] ] )   # (3,1)
u = v*w                           # (2,3)
# 计算过程：v 和 w 可扩展为 (2,3) 的数组
# 结果为
# array([[1, 2],
#        [2, 4],
#        [3, 6]])
```

```
# 行向量乘以列向量
v = np.array( [ [1,2] ] )               # (1,2)
w = np.array( [ [1,2],[3,4],[5,6] ] )   # (3,2)
u = v*w                                 # (3,2)
# 计算过程：v 可根据 w
# 结果为
# array([[ 1,  4],
#        [ 3,  8],
#        [ 5, 12]])
```

矩阵乘法 Ax 使用 np.dot 或 @ 实现，在具体使用时要考虑几种情形，我们用具体例子说明。

矩阵乘法

矩阵乘法使用 np.dot 或 @ 实现，在具体使用时要考虑几种情形，我们用具体例子说明。以下设 A 为 (2,2) 的 ndarray （只要不是一维数组）.3, 4]])

矩阵乘法

- 矩阵乘法用 `A@B` 或 `np.dot(A,B)` 实现。
- 当 `A` 是一维数组，视其为行向量；当 `B` 是一维数组，视其为列向量。这是因为矩阵乘法不可交换。
- 当 `A` 和 `B` 均为一维数组时，此时恰好为数学中向量的点乘或点积。

矩阵乘法 Ax 使用 np.dot 或 @ 实现，在具体使用时要考虑几种情形，我们用具体例子说明。

```
# A @ 向量
A = np.array([[1, 2], \
              [3, 4], \
              [5, 6]])
u = np.array( [ [1],[2] ] )   # (2,1)：矩阵乘法，结果为列向量
v = np.array( [ [1,2] ] )     # (1,2)：无意义
w = np.array( [1,2] )         # (2,)：  w 视为列向量，结果为一维数组，而非列向量
```

Python 中的循环

Python 的 for 循环比 MATLAB 的功能要强大，后者一般只能通过索引进行循环。

- 通过索引循环

```
fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
    print('当前水果 :', fruits[index]) # 注意使用 Tab 键产生空格
```

- 通过序列循环

```
fruits = ['banana', 'apple',  'mango']
for fruit in fruits:
    print('当前水果 :', fruit)
```

- `enumerate`为循环的列表加上索引

```
val = [1,2,3,5,8]
for id,val in enumerate(val):
    print(id,val)
```

- `zip`方法
  - 将多个 list 组合成 tuple

```
s = [1,2,3]
t = ['a','b','c']
# zip 方法将对应位置的分量打包在元组中，但它仅返回地址 <zip at 0x22303f7a748>
zip(s,t) # 返回地址 <zip at 0x22303f7a748>
# 可把打包的数据放在列表或元组中查看
list(zip(s,t))  # 结果为 [(1, 'a'), (2, 'b'), (3, 'c')]
tuple(zip(s,t)) # 结果为 ((1, 'a'), (2, 'b'), (3, 'c'))
```

  - 将 tuple 组成的 list 拆分成多个 tuple

```
st = [(1, 'a'), (2, 'b'), (3, 'c')]
s,t = zip(*st)
# 结果为 s = (1, 2, 3),  t = ('a', 'b', 'c')
```

- 多个 list 打包循环

```
s = [1,2,3]
t = ['a','b','c']
for si, ti in zip(s,t):
    print("{}: {}".format(si, ti))
# 结果为
#   1: a
#   2: b
#   3: c
```