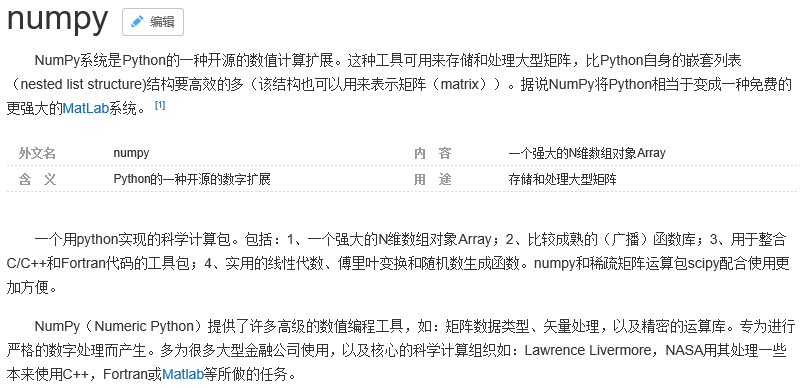
**2017.08.28--2017.09.10 周报**

**——王雪婷**

1. **摘要**
2. 自学 莫烦python 视频中，神经网络的介绍
3. 自学 numpy库**（网址：https://docs.scipy.org/doc/numpy-dev/user/quickstart.html）**
4. **主要内容**
5. **神经网络的自学**
6. **Numpy库的自学**

这个库是全英文的，为了方便自己日后复习，我也做了部分文字上的翻译（可能有部分专业名词不准确），并且查了一些中文的教程，总结如下：



**NumPy数组**

NumPy数组是一个多维数组对象，称为ndarray。其由两部分组成：

* 实际的数据
* 描述这些数据的元数据

大部分操作仅针对于元数据，而不改变底层实际的数据。

关于NumPy数组有几点必需了解的：

* NumPy数组的下标从0开始。
* 同一个NumPy数组中所有元素的 类型 必须是相同的。

**NumPy数组属性**

NumPy数组的维数称为秩（rank），一维数组的秩为1，二维数组的秩为2，以此类推。在NumPy中，每一个线性的数组称为是一个轴（axes），秩其实是描述轴的数量。比如说，二维数组相当于是两个一维数组，其中第一个一维数组中每个元素又是一个一维数组。所以一维数组就是NumPy中的轴（axes），第一个轴相当于是底层数组，第二个轴是底层数组里的数组。而轴的数量——秩，就是数组的维数。

NumPy的数组中比较重要ndarray对象属性有：

**1.ndarray.ndim：**数组的维数（即数组轴的个数），等于秩。最常见的为二维数组（矩阵）。

**2.ndarray.shape：**数组的维度。为一个表示数组在每个维度上大小的整数元组。例如二维数组中，表示数组的“行数”和“列数”。ndarray.shape返回一个元组，这个元组的长度就是维度的数目，即ndim属性。

**3.ndarray.size：**数组元素的总个数，等于shape属性中元组元素的乘积。

**4.ndarray.dtype：**表示数组中元素类型的对象，可使用标准的Python类型创建或指定dtype。

**5.ndarray.itemsize：**数组中每个元素的字节大小。例如，一个元素类型为float64的数组itemsiz属性值为8(float64占用64个bits，每个字节长度为8，所以64/8，占用8个字节），又如，一个元素类型为complex32的数组item属性为4（32/8）。

**6.ndarray.data：**包含实际数组元素的缓冲区，由于一般通过数组的索引获取元素，所以通常不需要使用这个属性。

例如：

>>> a = np.zeros((2,2,2))  
>>> print a.ndim   #数组的维数  
3  
>>> print a.shape  #数组每一维的大小  
(2, 2, 2)  
>>> print a.size   #数组的元素数  
8  
>>> print a.dtype  #元素类型  
float64  
>>> print a.itemsize  #每个元素所占的字节数  
8

然后是一些numpy的**应用**：

注：在使用numpy之前一定要先安装，之前老师给过如何安装的word，所以我就不写了；或者直接下一个anaconda，里面自带numpy库。然后在写以下语句之前，是需要一行：**import numpy as np** 的。

**使用numpy.array方法**

以list或tuple变量为参数产生**一维数组**：

>>> print np.array([1,2,3,4])  
[1 2 3 4]  
>>> print np.array((1.2,2,3,4))  
[ 1.2  2.   3.   4. ]  
>>> print type(np.array((1.2,2,3,4)))  
<type 'numpy.ndarray'>

以list或tuple变量为元素产生**二维数组**：

>>> print np.array([[1,2],[3,4]])  
[[1 2]  
 [3 4]]

生成数组的时候，可以**指定数据类型**，例如numpy.int32, numpy.int16, and numpy.float64等：

>>> print np.array((1.2,2,3,4), dtype=np.int32)  
[1 2 3 4]

**使用numpy.arange方法**

>>> print np.arange(15)  
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]  
>>> print type(np.arange(15))  
<type 'numpy.ndarray'>  
>>> print np.arange(15).reshape(3,5)  
[[ 0  1  2  3  4]  
 [ 5  6  7  8  9]  
 [10 11 12 13 14]]  
>>> print type(np.arange(15).reshape(3,5))  
<type 'numpy.ndarray'>

**使用numpy.linspace方法**

例如，在从1到3中产生9个数：

>>> print np.linspace(1,3,9)  
[ 1.    1.25  1.5   1.75  2.    2.25  2.5   2.75  3.  ]

**使用numpy.zeros，numpy.ones，numpy.eye等方法可以构造特定的矩阵**

例如：

>>> print np.zeros((3,4))  
[[ 0.  0.  0.  0.]  
 [ 0.  0.  0.  0.]  
 [ 0.  0.  0.  0.]]  
>>> print np.ones((3,4))  
[[ 1.  1.  1.  1.]  
 [ 1.  1.  1.  1.]  
 [ 1.  1.  1.  1.]]  
>>> print np.eye(3)  
[[ 1.  0.  0.]  
 [ 0.  1.  0.]  
 [ 0.  0.  1.]]

创建一个三维数组：

>>> print np.zeros((2,2,2))  
[[[ 0.  0.]  
  [ 0.  0.]]

 [[ 0.  0.]  
  [ 0.  0.]]]

**数组索引，切片，赋值**

示例：

>>> a = np.array( [[2,3,4],[5,6,7]] )  
>>> print a  
[[2 3 4]  
 [5 6 7]]  
>>> print a[1,2]  
7  
>>> print a[1,:]  
[5 6 7]  
>>> print a[1,1:2]  
[6]  
>>> a[1,:] = [8,9,10]  
>>> print a  
[[ 2  3  4]  
 [ 8  9 10]]

**使用for操作元素**

>>> for x in np.linspace(1,3,3):  
...     print x  
...  
1.0  
2.0  
3.0

**基本的数组运算**

先构造数组a、b：

>>> a = np.ones((2,2))  
>>> b = np.eye(2)  
>>> print a  
[[ 1.  1.]  
 [ 1.  1.]]  
>>> print b  
[[ 1.  0.]  
 [ 0.  1.]]

数组的加减乘除：

>>> print a > 2  
[[False False]  
 [False False]]  
>>> print a+b  
[[ 2.  1.]  
 [ 1.  2.]]  
>>> print a-b  
[[ 0.  1.]  
 [ 1.  0.]]  
>>> print b\*2  
[[ 2.  0.]  
 [ 0.  2.]]  
>>> print (a\*2)\*(b\*2)  
[[ 4.  0.]  
 [ 0.  4.]]  
>>> print b/(a\*2)  
[[ 0.5  0. ]  
 [ 0.   0.5]]  
>>> print (a\*2)\*\*4  
[[ 16.  16.]  
 [ 16.  16.]]

 使用数组对象自带的方法：

>>> a.sum()  
4.0  
>>> a.sum(axis=0)   #计算每一列（二维数组中类似于矩阵的列）的和  
array([ 2.,  2.])  
>>> a.min()  
1.0  
>>> a.max()  
1.0

使用numpy下的方法：

>>> np.sin(a)  
array([[ 0.84147098,  0.84147098],  
       [ 0.84147098,  0.84147098]])  
>>> np.max(a)  
1.0  
>>> np.floor(a)  
array([[ 1.,  1.],  
       [ 1.,  1.]])  
>>> np.exp(a)  
array([[ 2.71828183,  2.71828183],  
       [ 2.71828183,  2.71828183]])  
>>> np.dot(a,a)   ##矩阵乘法  
array([[ 2.,  2.],  
       [ 2.,  2.]])

**合并数组**

使用numpy下的vstack和hstack函数：

>>> a = np.ones((2,2))  
>>> b = np.eye(2)  
>>> print np.vstack((a,b))  
[[ 1.  1.]  
 [ 1.  1.]  
 [ 1.  0.]  
 [ 0.  1.]]  
>>> print np.hstack((a,b))  
[[ 1.  1.  1.  0.]  
 [ 1.  1.  0.  1.]]

看一下这两个函数有没有涉及到浅拷贝这种问题：

>>> c = np.hstack((a,b))  
>>> print c  
[[ 1.  1.  1.  0.]  
 [ 1.  1.  0.  1.]]  
>>> a[1,1] = 5  
>>> b[1,1] = 5  
>>> print c  
[[ 1.  1.  1.  0.]  
 [ 1.  1.  0.  1.]]

可以看到，a、b中元素的改变并未影响c。

**深拷贝数组**

数组对象自带了浅拷贝和深拷贝的方法，但是一般用深拷贝多一些：

>>> a = np.ones((2,2))  
>>> b = a  
>>> b is a  
True  
>>> c = a.copy()  #深拷贝  
>>> c is a  
False

**基本的矩阵运算**

转置：

>>> a = np.array([[1,0],[2,3]])  
>>> print a  
[[1 0]  
 [2 3]]  
>>> print a.transpose()  
[[1 2]  
 [0 3]]

迹**（翻译得不是很准？？？）：**

>>> print np.trace(a)  
4

numpy.linalg模块中有很多关于矩阵运算的方法：

>>> import numpy.linalg as nplg

特征值、特征向量：

>>> print nplg.eig(a)  
(array([ 3.,  1.]), array([[ 0.        ,  0.70710678],  
       [ 1.        , -0.70710678]]))

然后还找到了几篇博客，是讲numpy的，里面还有一两个知识点的科普，但看评论说由于更新问题，好像有些知识点有问题，所以就不总结到这里了，但有兴趣的同学可以看看~

<http://blog.csdn.net/sunny2038/article/details/9002531>

http://blog.csdn.net/sunny2038/article/details/9023797

<http://blog.csdn.net/sunny2038/article/details/8907736>

http://www.jianshu.com/p/6ca8729dd102

1. **小结**

由于中途电脑死机又送去修，耽误了不少时间，加上新疆的IP上不了github等开发性网站……所以我这两周的自学内容不是很多，也就合并了这两周的周报。

神经网络的自学中，感觉莫烦讲的还是通俗易懂，但还是稍微有些散乱，每个视频之间的联系度一般，所以看的过程中也会记录下来自己不懂得地方或者一些词汇，再上网搜索（已补充在上面的周报中）；numpy库时看到之前有人自学了，然后看一些代码里也挺常见，就上官网自学了他的手册，总的来讲感觉就是数学的各种矩阵运算，但同样的答案可以有不同的写法，相似的写法也可能出不同的答案，所以中间还是有很多点需要区分和记忆，还有就是使用了numpy之后相当于一种运算的简化，比普通的人为输入一些算式要省时省力很多。