paddle学习笔记之numpy

逻辑：“前向计算输出、根据输出和真实值计算Loss、基于Loss和输入计算梯度、根据梯度更新参数值”四个部分反复执行，直到到损失函数最小。

由于参数每次只沿着梯度反方向更新一点点，因此方向并不需要那么精确。一个合理的解决方案是每次从总的数据集中随机抽取出小部分数据来代表整体，基于这部分数据计算梯度和损失来更新参数，这种方法被称作随机梯度下降法（Stochastic Gradient Descent，SGD），核心概念如下：

- minibatch：每次迭代时抽取出来的一批数据被称为一个minibatch。
- batch size：每个minibatch所包含的样本数目称为batch size。
- Epoch：当程序迭代的时候，按minibatch逐渐抽取出样本，当把整个数据集都遍历到了的时候，则完成了一轮训练，也叫一个Epoch（轮次）。启动训练时，可以将训练的轮数`num_epochs`和`batch_size`作为参数传入。

涉及到数据处理和训练过程两部分代码的修改。

1）数据处理需要实现拆分数据批次和样本乱序（为了实现随机抽样的效果）两个功能。

即拆分batch和shuffle乱序

2）训练过程代码修改。将每个随机抽取的minibatch数据输入到模型中用于参数训练。训练过程的核心是两层循环：

- 第一层循环，代表样本集合要被训练遍历几次，称为“epoch”，代码如下：

```
for epoch_id in range(num_epochs):
```

- 第二层循环，代表每次遍历时，样本集合被拆分成的多个批次，需要全部执行训练，称为“iter (iteration)”，代码如下：

```
for iter_id,mini_batch in emumerate(mini_batches):
```

在两层循环的内部是经典的四步训练流程：前向计算->计算损失->计算梯度->更新参数，这与大家之前所学是一致的

NumPy具有如下功能：

- ndarray数组：一个具有矢量算术运算和复杂广播能力的多维数组，具有快速且节省空间的特点。
- 对整组数据进行快速运算的标准数学函数（无需编写循环）。
- 线性代数、随机数生成以及傅里叶变换功能。
- 读写磁盘数据、操作内存映射文件。

本质上，NumPy期望用户在执行“向量”操作时，像使用“标量”一样轻松。

广播是指 NumPy 在算术运算期间处理不同形状的数组的能力。对数组的算术运算通常在 相应的元素上进行。如果两个阵列具有完全相同的形状，则这些操作被无缝执行。

如果两个数组的维数不相同，则元素到元素的操作是不可能的。然而，在 NumPy 中仍然可以对形状不相似的数组进行操作，因为它拥有广播功能。较小的数组会广播到较大数组的大小，以便使它们的形状可兼容。

如果满足以下条件之一，那么数组被称为可广播的。

数组拥有相同形状。数组拥有相同的维数，且某一个或多个维度长度为 1 。数组拥有极少的维度，可以在其前面追加长度为 1 的维度，使上述条件成立
广播的规则:

规则 1：如果两个数组的维度数不相同，那么小维度数组的形状将会在最左边补 1。规则 2：如果两个数组的形状在任何一个维度上都不匹配，那么数组的形状会沿着维度 为 1 的维度扩展以匹配另外一个数组的形状。规则 3：如果两个数组的形状在任何一个维度上都不匹配并且没有任何一个维度等于 1， 那么会引发异常。

1 ndarray数组

Python中的List列表也可以非常灵活的处理多个元素的操作，但效率却非常低。与之比较，ndarray数组具有如下特点：

- ndarray数组中所有元素的数据类型相同、数据地址连续，批量操作数组元素时速度更快。而list列表中元素的数据类型可能不同，需要通过寻址方式找到下一个元素。
- ndarray数组支持广播机制，矩阵运算时不需要写for循环。
- NumPy底层使用C语言编写，内置并行计算功能，运行速度高于Python代码。

创建ndarray数组最简单的方式就是使用`array`函数，它接受一切序列型的对象（包括其他数组），然后产生一个新的含有传入数据的NumPy数组。下面通过示例展示`array`、`arange`、`zeros`和`ones`四个主要函数的实现方法。

- **`array`：创建嵌套序列（比如由一组等长列表组成的列表），并转换为一个多维数组。**

```
# 导入numpy
import numpy as np

# 从list创建array
a = [1,2,3,4,5,6]  # 创建简单的列表
b = np.array(a)    # 将列表转换为数组
b
array([1, 2, 3, 4, 5, 6])
```

- **`arange`：创建元素从0到10依次递增2的数组。**

```
# 通过np.arange创建
# 通过指定start, stop (不包括stop)，interval来产生一个1维的ndarray
a = np.arange(0, 10, 2)
a
array([0, 2, 4, 6, 8])
```

- `zeros`：创建指定长度或者形状的全0数组。

```
# 创建全0的ndarray
a = np.zeros([3,3])
a
array([[0., 0., 0.], [0., 0., 0.], [0., 0., 0.]])
```

- `ones`：创建指定长度或者形状的全1数组。

```
# 创建全1的ndarray
a = np.ones([3,3])
a
array([[1., 1., 1.], [1., 1., 1.], [1., 1., 1.]])
```

变形

- `shape`：数组的形状 ndarray.shape，1维数组（N,）（*N*,），2维数组（M,N）（*M*,*N*），3维数组（M,N,K）3维数组（*M*,*N*,*K*）。
- `dtype`：数组的数据类型。
- `size`：数组中包含的元素个数`ndarray.size`，其大小等于各个维度的长度的乘积。
- `ndim`：数组的维度大小`ndarray.ndim`，其大小等于`ndarray.shape`所包含元素的个数。

例如

```
a = np.ones([3, 3])
print('a, dtype: {}, shape: {}, size: {}, ndim: {}'.format(a.dtype, a.shape,
a.size, a.ndim))
a, dtype: float64, shape: (3, 3), size: 9, ndim: 2
```

创建ndarray之后，可以对其数据类型或形状进行修改，代码实现如下：

```
# 转化数据类型
b = a.astype(np.int64)
print('b, dtype: {}, shape: {}'.format(b.dtype, b.shape))

# 改变形状
c = a.reshape([1, 9])
print('c, dtype: {}, shape: {}'.format(c.dtype, c.shape))
b, dtype: int64, shape: (3, 3)
c, dtype: float64, shape: (1, 9)
```

运算

ndarray数组可以像普通的数值型变量一样进行加减乘除操作，主要包含如下两种运算：标量和ndarray数组之间的运算、两个ndarray数组之间的运算。

（1）标量和ndarray数组之间的运算

标量和ndarray数组之间的运算主要包括除法、乘法、加法和减法运算，代码实现如下：

```
# 标量除以数组，用标量除以数组的每一个元素
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
1. / arr
array([[1. , 0.5 , 0.33333333], [0.25 , 0.2 , 0.16666667]])


# 标量乘以数组，用标量乘以数组的每一个元素
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
2.0 * arr
array([[ 2., 4., 6.], [ 8., 10., 12.]])


# 标量加上数组，用标量加上数组的每一个元素
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
2.0 + arr
array([[3., 4., 5.], [6., 7., 8.]])


# 标量减去数组，用标量减去数组的每一个元素
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
2.0 - arr
array([[ 1., 0., -1.], [-2., -3., -4.]])
```

两个ndarray数组之间的运算

两个ndarray数组之间的运算主要包括减法、加法、乘法、除法和开根号运算，代码实现如下：

```
# 数组 减去 数组， 用对应位置的元素相减
arr1 = np.array([[1., 2., 3.], [4., 5., 6.]])
arr2 = np.array([[11., 12., 13.], [21., 22., 23.]])
arr1 - arr2
array([[-10., -10., -10.], [-17., -17., -17.]])


# 数组 加上 数组， 用对应位置的元素相加
arr1 = np.array([[1., 2., 3.], [4., 5., 6.]])
arr2 = np.array([[11., 12., 13.], [21., 22., 23.]])
arr1 + arr2
array([[12., 14., 16.], [25., 27., 29.]])


# 数组 乘以 数组，用对应位置的元素相乘
```

```
arr1 * arr2
array([[ 11., 24., 39.], [ 84., 110., 138.]])


# 数组 除以 数组，用对应位置的元素相除
arr1 / arr2
array([[0.09090909, 0.16666667, 0.23076923], [0.19047619, 0.22727273,
0.26086957]])


# 数组开根号，将每个位置的元素都开根号
arr ** 0.5
array([[1. , 1.41421356, 1.73205081], [2. , 2.23606798, 2.44948974]])
```

索引和切片

在编写模型过程中，通常需要访问或者修改ndarray数组某个位置的元素，则需要使用ndarray数组的索引。有些情况下可能需要访问或者修改一些区域的元素，则需要使用ndarray数组的切片。

ndarray数组的索引和切片的使用方式与Python中的list类似。通过[-n , n-1]的下标进行索引，通过内置的`slice`函数，设置其`start`,`stop`和`step`参数进行切片，从原数组中切割出一个新数组。ndarray数组的索引是一个内容丰富的主题，因为选取数据子集或单个元素的方式有很多。下面从一维数组和多维数组两个维度介绍索引和切片的方法。

1维ndarray数组的索引和切片

从表面上看，一维数组跟Python列表的功能类似，它们重要区别在于：数组切片产生的新数组，还是指向原来的内存区域，数据不会被复制，视图上的任何修改都会直接反映到源数组上。将一个标量值赋值给一个切片时，该值会自动传播到整个选区。

```
# 1维数组索引和切片
a = np.arange(30)
a[10]
10
a = np.arange(30)
b = a[4:7]
b
array([4, 5, 6])


# 将一个标量值赋值给一个切片时，该值会自动传播到整个选区。
a = np.arange(30)
a[4:7] = 10
a
array([ 0, 1, 2, 3, 10, 10, 10, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29])


# 数组切片产生的新数组，还是指向原来的内存区域，数据不会被复制。
# 视图上的任何修改都会直接反映到源数组上。
a = np.arange(30)
arr_slice = a[4:7]
arr_slice[0] = 100
```

```
a, arr_slice
(array([ 0,  1,  2,  3, 100,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29]), array([100,  5,  6]))


# 通过copy给新数组创建不同的内存空间
a = np.arange(30)
arr_slice = a[4:7]
arr_slice = np.copy(arr_slice)
arr_slice[0] = 100
a, arr_slice
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29]), array([100,  5,  6]))
```

多维ndarray数组的索引和切片

多维ndarray数组的索引和切片具有如下特点：

- 在多维数组中，各索引位置上的元素不再是标量而是多维数组。
- 以逗号隔开的索引列表来选取单个元素。
- 在多维数组中，如果省略了后面的索引，则返回对象会是一个维度低一点的ndarray。

多维ndarray数组的索引代码实现如下所示。

```
# 创建一个多维数组
a = np.arange(30)
arr3d = a.reshape(5, 3, 2)
arr3d
array(
[
[[ 0,     1], [ 2, 3],[ 4, 5]],
[[ 6,   7], [ 8, 9], [10, 11]],
[[12, 13], [14, 15], [16, 17]],
[[18, 19], [20, 21], [22, 23]],
[[24, 25], [26, 27], [28, 29]]
])


# 只有一个索引指标时，会在第0维上索引，后面的维度保持不变
arr3d[0]
array([[0, 1], [2, 3], [4, 5]])


# 两个索引指标
arr3d[0][1]
array([2, 3])
```

多维ndarray数组的切片代码如下所示。

```
# 创建一个数组
```

```
a = np.arange(24)
a
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23])
# reshape成一个二维数组
a = a.reshape([6, 4])
a
array([[ 0, 1, 2, 3], [ 4, 5, 6, 7], [ 8, 9, 10, 11], [12, 13, 14, 15], [16, 17,
18, 19], [20, 21, 22, 23]])


# 使用for语句生成list
[k for k in range(0, 6, 2)]
[0, 2, 4]


# 结合上面列出的for语句的用法
# 使用for语句对数组进行切片
# 下面的代码会生成多个切片构成的list
# k in range(0, 6, 2) 决定了k的取值可以是0, 2, 4
# 产生的list的包含三个切片
# 第一个元素是a[0 : 0+2]，
# 第二个元素是a[2 : 2+2]，
# 第三个元素是a[4 : 4+2]
slices = [a[k:k+2] for k in range(0, 6, 2)]
slices
[array([[0, 1, 2, 3], [4, 5, 6, 7]]),
array([[ 8, 9, 10, 11], [12, 13, 14, 15]]),
array([[16, 17, 18, 19], [20, 21, 22, 23]])]
slices[0]
array([[0, 1, 2, 3], [4, 5, 6, 7]])
```

数组的统计方法

可以通过数组上的一组数学函数对整个数组或某个轴向的数据进行统计计算。主要包括如下统计方法：

- `mean`：计算算术平均数，零长度数组的mean为NaN。
- `std`和`var`：计算标准差和方差，自由度可调（默认为n）。
- `sum`：对数组中全部或某轴向的元素求和，零长度数组的sum为0。
- `max`和`min`：计算最大值和最小值。
- `argmin`和`argmax`：分别为最大和最小元素的索引。
- `cumsum`：计算所有元素的累加。
- `cumprod`：计算所有元素的累积。

`sum`、`mean`以及标准差`std`等聚合计算既可以当做数组的实例方法调用，也可以当做NumPy函数使用。

```
# 计算均值，使用arr.mean() 或 np.mean(arr)，二者是等价的
arr = np.array(
[[1,2,3],
```

```
[4,5,6],
[7,8,9]])
arr.mean(), np.mean(arr)
(5.0, 5.0)


# 求和
arr.sum(), np.sum(arr)
(45, 45)


# 求最大值
arr.max(), np.max(arr)
(9, 9)


# 求最小值
arr.min(), np.min(arr)
(1, 1)


# 指定计算的维度
# 沿着第1维求平均，也就是将[1, 2, 3]取平均等于2，[4, 5, 6]取平均等于5，[7, 8, 9]取平均等于8
arr.mean(axis = 1)
array([2., 5., 8.])


# 沿着第0维求和，也就是将[1, 4, 7]求和等于12，[2, 5, 8]求和等于15，[3, 6, 9]求和等于18
arr.sum(axis=0)
array([12, 15, 18])


# 沿着第0维求最大值，也就是将[1, 4, 7]求最大值等于7，[2, 5, 8]求最大值等于8，[3, 6, 9]求最大值等于9
arr.max(axis=0)
array([7, 8, 9])


# 沿着第1维求最小值，也就是将[1, 2, 3]求最小值等于1，[4, 5, 6]求最小值等于4，[7, 8, 9]求最小值等于7
arr.min(axis=1)
array([1, 4, 7])


# 计算标准差
arr.std()
2.581988897471611


# 计算方差
arr.var()
6.666666666666667


# 找出最大元素的索引
arr.argmax(), arr.argmax(axis=0), arr.argmax(axis=1)
```

```
(8, array([2, 2, 2]), array([2, 2, 2]))


# 找出最小元素的索引
arr.argmin(), arr.argmin(axis=0), arr.argmin(axis=1)
(0, array([0, 0, 0]), array([0, 0, 0]))
```

2随机数np.random

主要介绍创建ndarray随机数组以及随机打乱顺序、随机选取元素等相关操作的方法。

创建随机ndarray数组

创建随机ndarray数组主要包含设置随机种子、均匀分布和正态分布三部分内容，代码如下所示。

（1）设置随机数种子

```
# 可以多次运行，观察程序输出结果是否一致
# 如果不设置随机数种子，观察多次运行输出结果是否一致
np.random.seed(10)
a = np.random.rand(3, 3)
a
array([[0.77132064, 0.02075195, 0.63364823], [0.74880388, 0.49850701,
0.22479665], [0.19806286, 0.76053071, 0.16911084]])
```

（2）均匀分布

```
# 生成均匀分布随机数，随机数取值范围在[0, 1)之间
a = np.random.rand(3, 3)
a
array([[0.08833981, 0.68535982, 0.95339335], [0.00394827, 0.51219226,
0.81262096], [0.61252607, 0.72175532, 0.29187607]])


# 生成均匀分布随机数，指定随机数取值范围和数组形状
a = np.random.uniform(low = -1.0, high = 1.0, size=(2,2))
a
array([[ 0.83554825, 0.42915157], [ 0.08508874, -0.7156599 ]])
```

（3）正态分布

```
# 生成标准正态分布随机数
a = np.random.randn(3, 3)
a
array([[ 1.484537 , -1.07980489, -1.97772828], [-1.7433723 , 0.26607016,
2.38496733], [ 1.12369125, 1.67262221, 0.09914922]])


# 生成正态分布随机数，指定均值loc和方差scale
a = np.random.normal(loc = 1.0, scale = 1.0, size = (3,3))
a
array([[2.39799638, 0.72875201, 1.61320418], [0.73268281, 0.45069099, 1.1327083
], [0.52385799, 2.30847308, 1.19501328]])
```

随机打乱ndarray数组顺序

（1）随机打乱1维ndarray数组顺序

```
# 生成一维数组
a = np.arange(0, 30)
print('before random shuffle: ', a)


# 打乱一维数组顺序
np.random.shuffle(a)
print('after random shuffle: ', a)
before random shuffle:  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
19 20 21 22 23
 24 25 26 27 28 29]
after random shuffle:  [14  3 20 21  9 26 25  1 22 23  0 11 17 10 27 16  6 19 13
15  8  2 28 18
 29  7  4  5 12 24]
```

从输出结果看，所有元素位置都被打乱了。

（2）随机打乱2维ndarray数组顺序

```
# 生成一维数组
a = np.arange(0, 30)
# 将一维数组转化成2维数组
a = a.reshape(10, 3)
print('before random shuffle: \n{}'.format(a))


# 打乱一维数组顺序
np.random.shuffle(a)
print('after random shuffle: \n{}'.format(a))
before random shuffle:
[[ 0  1  2]
 [ 3  4  5]
```

```
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]
 [15 16 17]
 [18 19 20]
 [21 22 23]
 [24 25 26]
 [27 28 29]]
after random shuffle:
[[ 9 10 11]
 [24 25 26]
 [18 19 20]
 [12 13 14]
 [21 22 23]
 [27 28 29]
 [ 0  1  2]
 [15 16 17]
 [ 6  7  8]
 [ 3  4  5]]
```

从输出结果看，只有行的顺序被打乱了，列顺序不变。

随机选取元素

```
# 随机选取部分元素
a = np.arange(30)
b = np.random.choice(a, size=5)
b
array([23, 3, 29, 16, 20])
```

3线性代数函数

线性代数（如矩阵乘法、矩阵分解、行列式以及其他方阵数学等）是任何数组库的重要组成部分，NumPy中实现了线性代数中常用的各种操作，并形成了numpy.linalg线性代数相关的模块。本节主要介绍如下函数：

- `diag`：以一维数组的形式返回方阵的对角线（或非对角线）元素，或将一维数组转换为方阵（非对角线元素为0）。
- `dot`：矩阵乘法。
- `trace`：计算对角线元素的和。
- `det`：计算矩阵行列式。
- `eig`：计算方阵的特征值和特征向量。
- `inv`：计算方阵的逆。

```
# 矩阵相乘
a = np.arange(12)
b = a.reshape([3, 4])
c = a.reshape([4, 3])
```

```
# 矩阵b的第二维大小，必须等于矩阵c的第一维大小
d = b.dot(c) # 等价于 np.dot(b, c)
print('a: \n{}'.format(a))
print('b: \n{}'.format(b))
print('c: \n{}'.format(c))
print('d: \n{}'.format(d))
a:
[ 0  1  2  3  4  5  6  7  8  9 10 11]
b:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
c:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
d:
[[ 42  48  54]
 [114 136 158]
 [186 224 262]]


# numpy.linalg  中有一组标准的矩阵分解运算以及诸如求逆和行列式之类的东西
# np.linalg.diag 以一维数组的形式返回方阵的对角线（或非对角线）元素，
# 或将一维数组转换为方阵（非对角线元素为0）
e = np.diag(d)
f = np.diag(e)
print('d: \n{}'.format(d))
print('e: \n{}'.format(e))
print('f: \n{}'.format(f))
d:
[[ 42  48  54]
 [114 136 158]
 [186 224 262]]
e:
[ 42 136 262]
f:
[[ 42   0   0]
 [  0 136   0]
 [  0   0 262]]


# trace，计算对角线元素的和
g = np.trace(d)
g
440


# det，计算行列式
h = np.linalg.det(d)
h
1.3642420526593978e-11


# eig，计算特征值和特征向量
```

```
i = np.linalg.eig(d)
i
(array([ 4.36702561e+02, 3.29743887e+00, -2.00728619e-14]), array([[ 0.17716392,
0.77712552, 0.40824829], [ 0.5095763 , 0.07620532, -0.81649658], [ 0.84198868,
-0.62471488, 0.40824829]]))


# inv，计算方阵的逆
tmp = np.random.rand(3, 3)
j = np.linalg.inv(tmp)
j
array([[ -5.81993172, 15.83159278, -11.18756558], [ 3.67058369, -6.46065502,
4.50665723], [ 2.11102657, -5.91510414, 5.31932688]])
```

4NumPy保存和导入文件

文件读写

NumPy可以方便的进行文件读写，如下面这种格式的文本文件：

```
0.00632  18.00   2.310  0  0.5380  6.5750  65.20  4.0900   1  296.0  15.30 396.90   4.98  24.00
0.02731   0.00   7.070  0  0.4690  6.4210  78.90  4.9671   2  242.0  17.80 396.90   9.14  21.60
0.02729   0.00   7.070  0  0.4690  7.1850  61.10  4.9671   2  242.0  17.80 392.83   4.03  34.70
0.03237   0.00   2.180  0  0.4580  6.9980  45.80  6.0622   3  222.0  18.70 394.63   2.94  33.40
0.06905   0.00   2.180  0  0.4580  7.1470  54.20  6.0622   3  222.0  18.70 396.90   5.33  36.20
0.02985   0.00   2.180  0  0.4580  6.4300  58.70  6.0622   3  222.0  18.70 394.12   5.21  28.70
0.08829  12.50   7.870  0  0.5240  6.0120  66.60  5.5605   5  311.0  15.20 395.60  12.43  22.90
0.14455  12.50   7.870  0  0.5240  6.1720  96.10  5.9505   5  311.0  15.20 396.90  19.15  27.10
0.21124  12.50   7.870  0  0.5240  5.6310 100.00  6.0821   5  311.0  15.20 386.63  29.93  16.50
0.17004  12.50   7.870  0  0.5240  6.0040  85.90  6.5921   5  311.0  15.20 386.71  17.10  18.90
```

```
# 使用np.fromfile从文本文件'housing.data'读入数据
# 这里要设置参数sep = ' '，表示使用空白字符来分隔数据
# 空格或者回车都属于空白字符，读入的数据被转化成1维数组
d = np.fromfile('./work/housing.data', sep = ' ')
d
array([6.320e-03, 1.800e+01, 2.310e+00, ..., 3.969e+02, 7.880e+00, 1.190e+01])
```

文件保存

NumPy提供了`save`和`load`接口，直接将数组保存成文件(保存为.npy格式)，或者从.npy文件中读取数组。

```
# 产生随机数组a
a = np.random.rand(3,3)
np.save('a.npy', a)

# 从磁盘文件'a.npy'读入数组
b = np.load('a.npy')

# 检查a和b的数值是否一样
check = (a == b).all()
check
True
```

5NumPy应用举例

计算激活函数Sigmoid和ReLU

使用ndarray数组可以很方便的构建数学函数，并利用其底层的矢量计算能力快速实现计算。下面以神经网络中比较常用激活函数Sigmoid和ReLU为例，使用Numpy计算激活函数Sigmoid和ReLU的值，并使用matplotlib画出图形

```
# ReLU和Sigmoid激活函数示意图
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.patches as patches

#设置图片大小
plt.figure(figsize=(8, 3))

# x是1维数组，数组大小是从-10. 到10.的实数，每隔0.1取一个点
x = np.arange(-10, 10, 0.1)
# 计算 Sigmoid函数
s = 1.0 / (1 + np.exp(- x))

# 计算ReLU函数
y = np.clip(x, a_min = 0., a_max = None)

#############################################################
# 以下部分为画图程序

# 设置两个子图窗口，将Sigmoid的函数图像画在左边
f = plt.subplot(121)
# 画出函数曲线
plt.plot(x, s, color='r')
# 添加文字说明
plt.text(-5., 0.9, r'$y=\sigma(x)$', fontsize=13)
# 设置坐标轴格式
currentAxis=plt.gca()
currentAxis.xaxis.set_label_text('x', fontsize=15)
currentAxis.yaxis.set_label_text('y', fontsize=15)

# 将ReLU的函数图像画在右边
f = plt.subplot(122)
# 画出函数曲线
plt.plot(x, y, color='g')
# 添加文字说明
plt.text(-3.0, 9, r'$y=ReLU(x)$', fontsize=13)
# 设置坐标轴格式
currentAxis=plt.gca()
currentAxis.xaxis.set_label_text('x', fontsize=15)
currentAxis.yaxis.set_label_text('y', fontsize=15)

plt.show()
```

图片翻转和裁剪

图片是由像素点构成的矩阵，其数值可以用ndarray来表示。将上述介绍的操作用在图像数据对应的ndarray上，可以很轻松的实现图片的翻转、裁剪和亮度调整，代码实现如下。

```
# 导入需要的包
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# 读入图片
image = Image.open('./work/pic1.jpg')
image = np.array(image)

# 查看数据形状，其形状是[H, W, 3]，
# 其中H代表高度， W是宽度，3代表RGB三个通道
image.shape
(404, 640, 3)
# 原始图片

plt.imshow(image)
```

```
# 垂直方向翻转
# 这里使用数组切片的方式来完成，
# 相当于将图片最后一行挪到第一行，
# 倒数第二行挪到第二行，...，
# 第一行挪到倒数第一行
# 对于行指标，使用::-1来表示切片，
# 负数步长表示以最后一个元素为起点，向左走寻找下一个点
# 对于列指标和RGB通道，仅使用:表示该维度不改变
image2 = image[::-1, :, :]
plt.imshow(image2)
```

```
# 水平方向翻转
image3 = image[:, ::-1, :]
plt.imshow(image3)
```

```
# 保存图片
im3 = Image.fromarray(image3)
im3.save('im3.jpg')


#  高度方向裁剪
H, W = image.shape[0], image.shape[1]
# 注意此处用整除，H_start必须为整数
H1 = H // 2
H2 = H
image4 = image[H1:H2, :, :]
plt.imshow(image4)
```

```
#  宽度方向裁剪
W1 = W//6
W2 = W//3 * 2
image5 = image[:, W1:W2, :]
plt.imshow(image5)


# 两个方向同时裁剪
image5 = image[H1:H2, \
               W1:W2, :]
plt.imshow(image5)


# 调整亮度
image6 = image * 0.5
plt.imshow(image6.astype('uint8'))


# 调整亮度
image7 = image * 2.0


# 由于图片的RGB像素值必须在0-255之间，
# 此处使用np.clip进行数值裁剪
image7 = np.clip(image7, \
        a_min=None, a_max=255.)
plt.imshow(image7.astype('uint8'))


#高度方向每隔一行取像素点
image8 = image[::2, :, :]
plt.imshow(image8)


#宽度方向每隔一列取像素点
image9 = image[:, ::2, :]
plt.imshow(image9)


#间隔行列采样，图像尺寸会减半，清晰度变差
image10 = image[::2, ::2, :]
plt.imshow(image10)
image10.shape
```

6飞桨的张量表示

飞桨使用张量（Tensor）表示数据。Tensor可以理解为多维数组，具有任意的维度，如一维、二维、三维等。不同Tensor可以有不同的数据类型 (dtype) 和形状 (shape)，同一Tensor的中所有元素的数据类型均相同。Tensor是类似于Numpy数组（ndarray）的概念。飞桨的Tensor高度兼容Numpy数组，在基础数据结构和方法上，增加了很多适用于深度学习任务的参数和方法，如：反向计算梯度，指定运行硬件等。

如下述代码声明了两个张量类型的向量*x*和*y*，指定CPU为计算运行硬件，要自动反向求导。两个向量除了可以与Numpy类似的做相乘的操作之外，还可以直接获取到每个变量的导数值。

```
import paddle
x = paddle.to_tensor([1.0, 2.0, 3.0], dtype='float32', place=paddle.CPUPlace(),
stop_gradient=False)
y = paddle.to_tensor([4.0, 5.0, 6.0], dtype='float32', place=paddle.CPUPlace(),
stop_gradient=False)
z = x * y
z.backward()
print("tensor's grad is: {}".format(x.grad))
```

此外，飞桨的张量还可以实现与Numpy的数组转换，代码实现如下。

```
import paddle
import numpy as np

tensor_to_convert = paddle.to_tensor([1.,2.])

#通过 Tensor.numpy() 方法，将张量转化为 Numpy数组
tensor_to_convert.numpy()

#通过paddle.to_tensor() 方法，将 Numpy数组 转化为张量
tensor_temp = paddle.to_tensor(np.array([1.0, 2.0]))
```

plt.figure与plt.subplot的区别

plt.figure(figsize=(6,8))
表示figure 的大小为宽、长（单位为inch）
plt.subplot(121)
表示整个figure分成1行2列，共2个子图，这里子图在第一行第一列

np.transpose
图片读入成ndarry时，形状是[H, W, 3]，
将通道这一维度调整到最前面
x = np.transpose(x, (2,0,1))

np.transpose(x)是行列转置；
np.transpose(x, (2,0,1))是将原本的(0,1,2)转换为顺序为(2，0，1)

list[::k]

if k > 0，数组从头往尾取；
if k < 0，数组从尾往头取

```
a = [1,2,3,4,5,6,7,8,9]
b = a[::2]
# [1, 3, 5, 7, 9]
b = a[::-2]
# [9, 7, 5, 3, 1]
12345'
```

numpy.repeat

w = np.repeat(w, 3, axis=1)

axis=0,沿着y轴复制，实际上增加了行数
axis=1,沿着x轴复制，实际上增加列数