

mxnet 笔记

徐世桐

1 NDArray

```
from mxnet import nd
```

```
x = nd.arange(12) // 创建一个长度为 12 的行向量，类型为 NDArray 12
```

```
x.shape // 返回 (m, n)，代表 x 为 m 行 n 列矩阵。对于向量，行数或列数不存在
```

```
x.size // 返回矩阵中元素个数
```

```
x.reshape((m', n'))
```

更改 x 的 shape，元素按行填写进新张量。如果 $n' * m' < \text{原元素数}$ ，多余元素被舍弃。如果 $n' * m' > \text{原元素数}$ ，报错

当二次 resize，使用与开始定义 x 的 size，而非上一次 resize 后舍弃部分值的 x.size

```
x.reshape((-1, n')) x.reshape((m', -1))
```

当 $m', n' = -1$ ， $m' = \lfloor \frac{x.size}{n'} \rfloor$ ， n' 同理

当 m', n' 为空，reshape 成向量

```
nd.zeros((v1, v2, v3, ..., vn))
```

创建一个张量，类型仍为 NDArray。有 v_1 个子张量，每个子张量分别有 v_2 个子张量。最后一层张量有 v_n 元素，每一元素都为 0。

张量同样可以 reshape，reshape 结果只能是矩阵或向量

```
nd.ones((v1, v2, v3, ..., vn)) // 所有元素为 1 的张量
```

```
nd.array([...])
```

得到 python list 类型的矩阵，返回 NDArray 类型的矩阵

输入可以是 python 常数，但随后计算会报错

```
nd.random.normal( $\mu, \sigma$ , shape=(v1, v2, v3, ..., vn))
```

随机生成张量，元素值 $\sim N(\mu, \sigma)$

```
X + - * / Y
```

张量 element-wise 操作

当 X Y 维数不同时，广播 broadcast 机制先将 X, Y 按行或列复制成维数一样的张量，随后 element-wise 操作

```
X.exp() // 张量 element-wise 取指数
```

```
nd.dot(X, Y) // 矩阵乘法
```

```
nd.concat(X, Y, dim=n)
```

在第 n 纬度将矩阵 concat，除此纬度其余所有纬度必须完全一样

```
X == Y
```

elementwise 比较张量元素，维度必须相同

`X.sum()` // 所有元素和

`X.sum(axis=n, keepdims=True/False)`

对张量第 n 维的数据求和

keepdims 时 shape 转换: $v_1, v_2, \dots, v_{n-1}, v_n, v_{n+1}, \dots \rightarrow v_1, v_2, \dots, v_{n-1}, 1, v_{n+1}, \dots$

否则, shape 转换: $v_1, v_2, \dots, v_{n-1}, v_n, v_{n+1}, \dots \rightarrow v_1, v_2, \dots, v_{n-1}, v_{n+1}, \dots$

例: $n=0$, 对列求和 $n=1$, 对行求和

`X.argmax(axis=n, keepdims=True/False)`

找到给定维度中最大元素的 index, shape 转换同上

算法: 对张量 $v_1, v_2, \dots, v_{n-1}, v_n, v_{n+1}, \dots$

结果张量包括一个 v_1, v_2, \dots, v_{n-1} 张量

每一元素张量为原张量 v_n 维中所有张量的 element wise 比较结果。shape 为 v_{n+1}, \dots , 其中每一元素为一 index, 标记 v_n 维中此位置最大元素值所在的张量 index

`X.norm()`

得到仅包含一元素的矩阵, 元素值为 2-norm

可以对张量取 2-norm

`X.asscalar()` // 如果 X 仅包含一元素, 输出此元素值

`X[v1, v2, v3, ..., vn]`

index 取值操作, 同 `X[v1][v2][v3]...[vn]`

当 v_i 为 $n:m$ 时, 代表范围 $[n, m)$

`X.asnumpy()` // 转换成 python list

`from d2lzh import plt`

`plt.scatter(array_x, array_y, 1)`

描点, `array_x, array_y` 为 python list

2 训练

`from mxnet import autograd`

`x.attach_grad()` // 为自变量 x 的 $\frac{d}{dx}$ 项分配内存

`with autograd.record():`

因变量 = 关于 x 的表达式

- 关于 x 的表达式可以为一个自定义 function, 不需要是一个连续的数学函数

- 自定义函数必须将所有使用的变量包括在 `def` 输入变量中, 不可使用全局变量。const 仍可使用全局变量

- 当变量为另一函数的结果, 计算另一函数的步骤需放进 `with` 中, 不能在 `with scope` 外计算完 `with` 内调取

因变量.`backward()`

- 定义 x 的表达式, 并计算表达式在 x 内每一元素值上的斜率, 对应斜率矩阵存在 `x.attach_grad()` 分配的内存中

- 当使用多组 sample, 因变量为向量。此时 backward 等同于因变量.sum().backward()
 x.grad // 调取斜率矩阵
 autograd.is_training() // 在 autograd.record() 内返回 true, 否则返回 false

3 使用 neural network 模型训练

```
from mxnet.gluon import nn
from mxnet import init
from mxnet.gluon import loss as gloss
net = nn.Sequential() // 创建一个神经网络模型, 不包含任何 layer
net.add(nn.Dense(2)) // 在模型中加入一个全连接层, 包含一个节点
net.initialize(init.Normal(sigma=0.3)) // 初始化整个神经网络, 所有权重  $\sim N(0, 0.3)$ , 所有偏差值 = 0
loss = gloss.L2Loss()
    定义损失函数为平方损失函数, loss 为一函数
trainer = gluon.Trainer(net.collect_params(NAME), 'sgd', 'learning_rate': 0.03, WD)
    定义每一步优化函数, 使用 sgd 梯度下降
    NAME 无定义, 则同时训练权重
        NAME = '.*weight', 只训练权重
        NAME = '.*bias', 只训练偏差
    WD 无定义, 则不使用权重衰减
        WD = wd:'wd', 使用权重衰减
dataset = gdata.ArrayDataset(features, labels)
data_iter = gdata.DataLoader(dataset, batch_size, shuffle=True, num_workers)
    按批量读取数据, num_workers 代表使用的额外处理器数, 0 代表没有额外处理器
trainer.step(batch_size)
    调用优化函数, 取 batch_size 个 sample 做一步训练
l = loss(net(features), labels)
    调用 loss 函数
```

4 图像分类数据集 fashion-MINST

```
from mxnet.gluon import data as gdata
mnist_train = gdata.version.FashionMINST(train=True)
    得到数据集, train=True 为训练数据集, 否则为测试训练集
features, label = mnist_train[i:j]
    得到一个或多个 sample
    features 为 (j-i, 28, 28, 1) 的张量, 类型为 NDArray
from d2lzh import show_fashion_mnist
show_fashion_mnist(features, ['label1', 'label2'])
    打印图片, 显示图片仍使用 plt.show()
```

features 必须包含多余一个图片矩阵，否则报错