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' < 原元素数，多余元素被舍弃。如果 n' * m' > 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(`$(v_1, v_2, v_3, ..., v_n)$`)`

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(`$(v_1, v_2, v_3, ..., v_n)$`)` // 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(`μ, σ`, shape=(`$v_1, v_2, v_3, ..., v_n$`))`

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 boardcast 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

`X.relu()` // 对每一 X 元素使用 relu 激发函数

`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，纬度必须相同

`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：$v_1, v_2, ..., v_{n-1}, v_n, v_{n+1}, ...$ -> $v_1, v_2, ..., v_{n-1}, 1, v_{n+1}, ...$

否则，shape 转换：$v_1, v_2, ..., v_{n-1}, v_n, v_{n+1}, ...$ -> $v_1, v_2, ..., v_{n-1}, v_{n+1}, ...$

例：n=0，对列求和 n=1，对行求和

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

找到给定维度中最大元素的 index，shape 转换同上

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

结果张量包括一个 $v_1, v_2, ..., v_{n-1}$ 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 $v_{n+1}, ...$，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[`$v_1, v_2, v_3, ..., v_n$`]`

index 取值操作，同 X[v_1][v_2][v_3]...[v_n]

当 v_i 为 n:m 时，代表范围 $[n, m)$

`X.asnumpy()` // 转换成 python list

`nd.stack(`x_1`,` x_2`, ...)` // 根据提供的子张量，构造高维的张量 `nd.add_n(`x_1`,` x_2`, ...)` // 多个 NDArray 相加，等同 $x_1 + x_2 + ...$

`func(*[elem, elem, ...])`

在函数输入中，当函数得到多个元素，调取方法为 func(elem, elem, ...)。使用 * 将 list 转换成分隔的参数输入

```
from d2lzh import plt
plt.scatter(array_x, array_y, 1)
```

描点，`array_x`, `array_y` 为 python list

```
nd.save('FILE_NAME', X)
```

存储 NDArray 数据，存入 FILE_NAME 文件中

X 可为 NDArray, [NDArray] 数组, KEY: NDArray, ... 字典

读取：`X = load('FILE_NAME')`。读取 NDArray 和 [NDArray] 时返回类型都为 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.Dense(2, IN_UNITS, ACTIVATION)
```

创建一个全连接层，包含 2 个节点

IN_UNITS = `in_units=N` 避免延后初始化。定义输入变量数，使得 net 在调用 initialize() 后即有权重矩阵，否则需要一次 forward 后才能访问权重矩阵

ACTIVATION = `activation='relu'` 定义激发函数

`net = nn.Sequential()` // 创建一个神经网络模型，不包含任何 layer

`net.add(nn.Dense(...), nn.Dropout(PROB), ...))`

add 中可同时包含多个层

nn.Dropout() 为丢弃法使用的丢弃层。定义前一层的权重有 PROB 几率被清零，1-p 几率被拉伸

`net.initialize(INIT, FORCE_REINIT)`

初始化层内的参数，随后调用 `net(X)` 得到全连接层对输入矩阵 X 的输出

INIT = `init.Normal(sigma=0.3)` 初始化整个神经网络，对每一层调用 initialize。所有权重 $\sim N(0, 0.3)$，所有偏差值 =0

INIT = `init.Xavier()` 使用 Xavier 随机初始化

FORCE_INIT = `force_init = True` 强制初始化参数，不论参数有没有被初始化过

对 nn.Dense() 和 nn.Sequential() 创建的 net 都可调用，用法一样

`net[i].params`

访问第 i hidden layer 的权重偏差值。

返回类型为 ParameterDict，可通过 ['KEY'] 分别得到权重和偏差

`net[i].weight`

直接访问权重

net[i].weight.data() 得到 NDArray 类型的权重矩阵

`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 自定义 class 形式的 neural network

```
class CLASS_NAME(nn.Block):
    def __init__(self, **kwargs):
      super(NySequential, self).__init__(**kwargs)
    // 初始化 class

      self.weight = self.params.get('weight', shape=(., .))
    // 通过系统定义的 params 生成类型为 parameterDict 的参数矩阵

    def add(self, block):
      self._children[block.KEY] = block
    // 加入一个 layer，使用 layer 的一个 key 做 index 标注此 layer 存在哪一层
    // 例：layer 为 nn.Dense(2, activation='relu')

    def forward(self, x):
      for block in self._children.values():
        x = block(x)
        return x
```

// forward 写法，调用 net(X) 等同于调用 net.forward(X)

```
class INIT_NAME(init.Initializer):
    def _init_weight(self, name, data):
      (对 data 更改值的函数)
net.initialize(INIT_NAME)
```

使用定义的函数初始化权重偏差

5 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

填充 `padding`:

超参数，

6 convolutional network 卷积神经网络

```
CONV_2D = nn.Conv2D(CHANNEL_NUM, KERNEL_SIZE, PADDING, STRIDE)
```

- 定义一卷积层
- CHANNEL_NUM = 整数，定义输出通道数
- KERNEL_SIZE = `kernel_size = (., .)` 定义 kernel 的 shape
  `kernel=N` 等同 `kernel=(N, N)`
- PADDING = `padding=(n, m)` 输入矩阵上下分别添加 n 行，左右分别添加 m 列全零元素
  `padding=N` 等同 `padding=(N, N)`
- STRIDE = `stride=(n, m)` 横向移动步幅为 n，纵向步幅为 m
  `stride=N` 等同 `stride=(N, N)`

```
CONV_2D.weight.data()[:] = ... * CONV_2D.weight.grad()
```

优化函数写法。若输出矩阵计算由对每一元素赋值得到，无法使用自动求斜率

使用的输入输出矩阵 shape: (批量大小，通道数，行数，列数)

```
nn.MaxPool2D(SHAPE, PADDING, STRIDE)
```

SHAPE = `(n, m)` 或 `N` 定义池化窗口 shape

PADDING, STRIDE 同 kernel