

PyTorch API

冯哲*

西安电子科技大学计算机学院

2019年2月

目录

1	Pytorch 张量操作	4
1.1	创建 Tensor	4
1.1.1	从 numpy 导入	4
1.1.2	从 list 导入	4
1.1.3	未初始化	4
1.1.4	设置 Tensor 默认类型	4
1.1.5	随机初始化	4
1.1.6	初始化为同一个数	4
1.1.7	生成递增递减序列	4
1.1.8	ones/zeros/eyes/*_like	5
1.1.9	生成随机种子	5
1.2	索引和切片	5
1.2.1	简单索引	5
1.2.2	利用切片选取连续索引	5

*电子邮件: 1194585271@qq.com

1.2.3	利用切片选取间隔索引	5
1.2.4	选取不规则索引	6
1.2.5	使用符号 ... 推测维度	6
1.2.6	依据掩码的位置信息索引	6
1.2.7	打平索引	6
1.3	Tensor 维度变换	6
1.3.1	重塑维度-reshape/view	6
1.3.2	删减增加维度-squeeze/unsqueeze	7
1.3.3	单次多次交换维度-transpose/permute	7
1.3.4	维度扩展-expand/repeat	7
2	Pytorch 张量高阶操作	8
2.1	Broadcasting	8
2.2	Tensor 分割与合并	8
2.2.1	合并不增加维度-cat	8
2.2.2	合并增加维度-stack	8
2.2.3	根据长度分割-split	9
2.2.4	根据数量分割-chunk	9
2.3	Tensor 运算	9
2.3.1	加减乘除	9
2.3.2	矩阵乘	10
2.3.3	幂次方	10
2.3.4	平方根/平方根的倒数	10
2.3.5	自然常数幂/自然常数底	10
2.3.6	近似运算	10

2.3.7	数字范围裁剪	10
2.4	Tensor 统计	11
2.4.1	范数	11
2.4.2	最大最小平均累和累积	11
2.4.3	最大最小参数位置	12
2.4.4	第k大与topK	12
2.4.5	比较操作	12
2.5	Tensor 高阶操作	12
2.5.1	GPU离散复制-where	12
2.5.2	GPU收集查表操作-gather	13
3	随机梯度下降	14
3.1	激活函数	14
3.1.1	Sigmoid/Logistic	14
3.1.2	tanh	14
3.1.3	ReLU	14
3.2	损失函数	14
3.2.1	MSE	14
3.2.2	Cross Entropy Loss	14
3.2.3	Softmax	14
3.3	求导方法	15
3.3.1	自动求导	15
3.3.2	反向回传求导数	15

1 Pytorch 张量操作

1.1 创建 Tensor

1.1.1 从 numpy 导入

```
torch.from_numpy(numpy.array)
```

1.1.2 从 list 导入

```
torch.tensor(list)
```

1.1.3 未初始化

```
torch.empty(shape_no_list)
torch.Tensor(shape_no_list)
torch.IntTensor(shape_no_list)
torch.FloatTensor(shape_no_list)
```

1.1.4 设置 Tensor 默认类型

```
torch.set_default_tensor_type(torch.FloatTensor/DoubleTensor)
```

1.1.5 随机初始化

```
rand(shape)
randint(min,max,[shape]) #[min,max]
rand_like(tensor)
randn(shape) #data ~ N(0,1)
```

1.1.6 初始化为同一个数

```
torch.full([shape],number)
```

1.1.7 生成递增递减序列

```
torch.arange(min, max, step)    #[min, max)    int
torch.linspace(min, max, steps)  #[min, max]    steps=numbers float
torch.logspace(min, max, steps) #[min, max]    step^n
```

1.1.8 ones/zeros/eyes/*_like

```
torch.ones(shape)
torch.zeros(shape)
torch.eye(shape)
torch.*_like(tensor)
```

1.1.9 生成随机种子

```
torch.randperm(int)
```

1.2 索引和切片

1.2.1 简单索引

```
a = torch.rand(10,3,28,28)
a[0]    #第0张照片
a[0,0]   #第0张照片的第0个通道
a[0,0,0]  #第0张照片的第0个通道的第0行像素 dim为1
a[0,0,0,0] #第0张照片的第0个通道的第0行的第0个像素 dim为0
```

1.2.2 利用切片选取连续索引

```
a = torch.rand(10,3,28,28)
a[:2]    #取前两张图片
a[-2:]   #取后两张图片
a[:2,:1]  #取前两张图片的第一个通道
a[-2:,-2:] #取后两张图片的后两个通道
```

1.2.3 利用切片选取间隔索引

```
a[:, :, 0:28:2, 0:28:2]    #取全部图片的全部通道的长宽均间隔采样
```

1.2.4 选取不规则索引

```
a[2][1][18][26] #取第2张图片的第1通道的第18行26列的像素值，标量
a.index_select(dim,tensor) #第一个参数表示维度，第二个是tensor值
a.index_select(0,torch.tensor([0, 3, 3])) #选择第0第3第3张图片
a.index_select(1,torch.tensor([0,2])) #选择四张图片的第0和第2的通道
a.index_select(2,torch.arange(0,8)) #选择四张图片每个通道的前8所有列的像素
```

1.2.5 使用符号 ... 推测维度

```
a[...].shape
a[:3,...].shape
a[:,1,...].shape
a[...,:10].shape
a[0,...,:2].shape #间隔采样 ::
```

1.2.6 依据掩码的位置信息索引

```
x = torch.randn(3,3)
mask = x.ge(0.5) #>=0.5的位置信息
torch.masked_select(x,mask) #得到所有>=0.5的tensor值
```

1.2.7 打平索引

```
src = torch.tensor([[1,2,3], [4,5,6]])
torch.take(src, torch.tensor([0,2,5])).shape #torch.Size([3])
```

1.3 Tensor 维度变换

1.3.1 重塑维度-reshape/view

```
tensor.view(a,b,c,...)
tensor.reshape(a,b,c,...)
```

1.3.2 删减增加维度-squeeze/unsqueeze

Pos. Idx	0	1	2	3
	4	3	28	28
Neg. Idx	-4	-3	-2	-1

```
a = torch.rand(4, 3, 28, 28)
a.unsqueeze(0).shape  #[1, 4, 1, 28, 28] 0维度前面插入一个维度
a.unsqueeze(-1).shape  #[4, 1, 28, 28, 1] 在最后一个维度后面插入一个维度
a = torch.rand(1, 32, 1, 1)
a.squeeze().shape  #torch.Size([32]) 尽可能多的删减维度
a.squeeze(0).shape  #torch.Size([32, 1, 1])
a.squeeze(-2).shape  #torch.Size([1, 32, 1])
```

1.3.3 单次多次交换维度-transpose/permute

```
a = torch.randn(3,4)
a.t() 只能适用于二维转置
```

```
a = torch.randn(4, 3, 28, 28) 记录维度信息否则污染数据
b = a.transpose(1,3).reshape(4, 3*28*28).reshape(4, 3, 28, 28) 数据污染
c = a.transpose(1,3).reshape(4, 3*28*28).reshape(4, 28, 28, 3)
    .transpose(1,3)
torch.all(torch.eq(a, b))  #tensor(0, dtype=torch.uint8)
torch.all(torch.eq(a, c))  #tensor(1, dtype=torch.uint8)
```

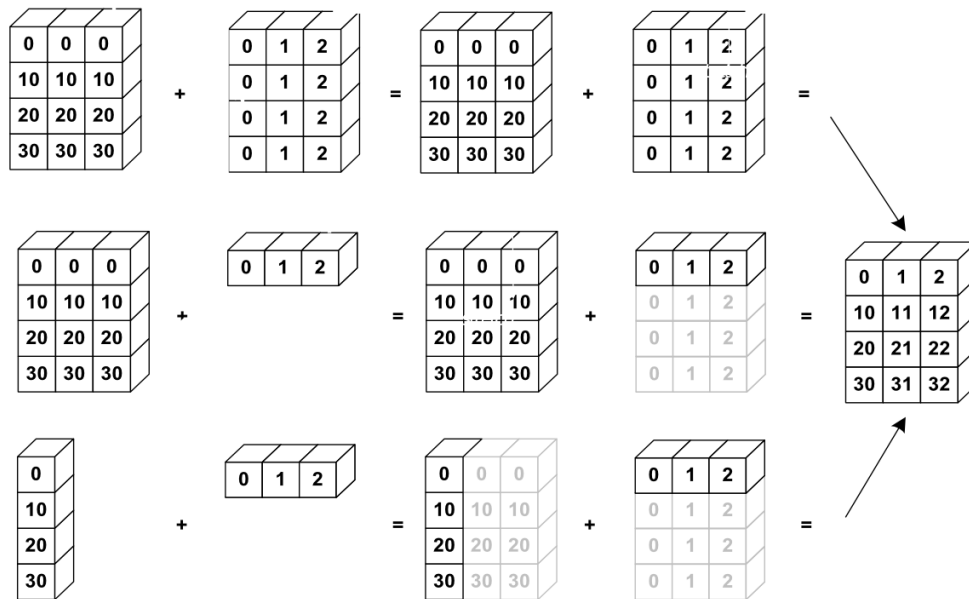
```
transpose 只能做单次交换但 permute 可以做多次交换
a = torch.randn(4, 3, 28, 32) 目标 (4, 28, 32, 3)
a.transpose(1, 3).transpose(1, 2).shape #torch.Size([4, 28, 32, 3])
a.permute(0, 2, 3, 1).shape #torch.Size([4, 28, 32, 3])
```

1.3.4 维度扩展-expand/repeat

```
rexpand 表示扩展到的维度内存无关
tensor.expand(a,b,c,d)
repeat 表示复制次数内存相关
tensor.repeat(a,b,c,d)
```

2 Pytorch 张量高阶操作

2.1 Broadcasting



2.2 Tensor 分割与合并

2.2.1 合并不增加维度-cat

```
a = torch.rand(4, 32, 8) #[classes, students, scores]
b = torch.rand(5, 32, 8) cat维度可以不同
torch.cat([a, b],dim=0).shape #torch.Size([9, 32, 8])
```

2.2.2 合并增加维度-stack

```
a = torch.rand(32, 8) #[students, scores]
b = torch.rand(32, 8) 创建新的维度, 旧维度必须一致
c = torch.rand(32, 8)
torch.stack([a, b, c],dim=0).shape #torch.Size([3, 32, 8])
```


2.2.3 根据长度分割-split

```
a = torch.rand(4, 32, 8) #[classes, students, scores]
aa, bb, cc = a.split([1,2,1], dim=0)
aaa, bbb = a.split(2, dim=0)
aa.shape #torch.Size([1, 32, 8])
bb.shape #torch.Size([2, 32, 8])
cc.shape #torch.Size([1, 32, 8])
aaa.shape #torch.Size([2, 32, 8])
bbb.shape #torch.Size([2, 32, 8])
```

2.2.4 根据数量分割-chunk

```
a = torch.rand(6, 32, 8) #[classes, students, scores]
aa, bb= a.chunk(2, dim=0)
cc, dd, ee =a.split(2, dim=0)
aa.shape #torch.Size([3, 32, 8])
bb.shape #torch.Size([3, 32, 8])
cc.shape #torch.Size([2, 32, 8])
dd.shape #torch.Size([2, 32, 8])
ee.shape #torch.Size([2, 32, 8])
```

2.3 Tensor 运算

2.3.1 加减乘除

```
a = torch.rand(4,3)
b = torch.rand(3)
torch.all(torch.eq(a+b, torch.add(a,b))) #tensor(1, dtype=torch.uint8)
a-b      #torch.sub
a*b      #torch.mul
a/b      #torch.div
a//b     地板除
```

2.3.2 矩阵乘

```
a = torch.rand(4,3)    最后两维做矩阵乘运算，其他符合broadcast机制
b = torch.rand(3,8)
torch.mm(a, b)         #only for 2d
(a @ b).shape          #torch.matmul torch.Size([4, 8])
```

2.3.3 幂次方

```
a**2      #torch.pow
```

2.3.4 平方根/平方根的倒数

```
a.sqrt()   #a**0.5
a.rsqrt()
```

2.3.5 自然常数幂/自然常数底

```
torch.exp(a)    #e**a
torch.log(a)     #lna
```

2.3.6 近似运算

```
a = torch.tensor(3.14)    #tensor(3.14)
a.floor(), a.ceil(), a.round()  #tensor(3.)  tensor(4.)  tensor(3.)
a.trunc()                 #tensor(3.)
a.frac()                   #tensor(0.1400)
```

2.3.7 数字范围裁剪

```
grad = torch.rand(3,4)*15
grad.min()      #min number
grad.max()      #max number
grad.median()   #median number
grad.clamp(10)  #min number is 10
grad.clamp(0, 10)  #all numbers is [0,10]
```

2.4 Tensor 统计

2.4.1 范数

vector norm

$$\|x\|_1 = \sum_{i=1}^n |a_i|$$

$$\|x\|_e = \sqrt{\sum_{i=1}^n x_i^2}$$

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}}$$

matrix norm

$$\|A\|_1 = \max_{i \leq j \leq n} \sum_{i=1}^n |a_{ij}|$$

$$\|A\|_e = \sqrt{\sum_{i=1}^n \sum_{j=1}^n a_{ij}^2}$$

$$\|A\|_p = \left(\sum_{i=1}^n \sum_{j=1}^n a_{ij}^2 \right)^{\frac{1}{p}}$$

```
a = torch.full([8], 1)
b = a.reshape(2, 4)
c = b.reshape(2, 2, 2)
a #tensor([1., 1., 1., 1., 1., 1., 1., 1.])
b #tensor([[1., 1., 1., 1.], [1., 1., 1., 1.]])
a.norm(1), b.norm(1), c.norm(1) #tensor(8.)
a.norm(2), b.norm(2), c.norm(2) #tensor(2.8284)
#two parameters norm, dimension
a.norm(1, dim=0) #tensor(8.)
b.norm(1, dim=1) #tensor([4., 4.])
c.norm(2, dim=2) #tensor([[1.4142, 1.4142], [1.4142, 1.4142]])
```

2.4.2 最大最小平均累和累积

```
a = torch.arange(8).reshape(2,4).float()
a.min() #tensor(0.)
a.max() #tensor(7.)
a.mean() #tensor(3.5)
a.mean(1) #tensor([1.5000, 5.5000])
a.sum() #tensor(28.)
a.prod() #tensor(0.)
```

2.4.3 最大最小参数位置

```
a = torch.randn(4, 10) 4张照片 0-9 10个概率值
a.argmax() a.argmin() 无参数默认打平
a.argmax(1) 返回每张照片概率最大的数字
a.argmax(1, keepdim=True) 返回每张照片概率最大的数字并保持维度信息
a.max(1) 返回每张照片最大的概率及数字
```

2.4.4 第k大与topK

```
a = torch.randn(4, 10) 4张照片 0-9 10个概率值
a.topk(2, dim=1, largest=True) largest = False 表示最小的 k 个
a.kthvalue(10, dim=1) 返回第10小的概率及位置
```

2.4.5 比较操作

```
>, <, >=, <=, ! =, ==
torch.eq() 可 broadcast, 返回 0/1 同型
torch.equal() 比较每一值, 都相等返回 True
```

2.5 Tensor 高阶操作

2.5.1 GPU离散复制-where

```
torch.where(condition, x, y) --> Tensor 满足条件取 x, 否则取 y
其功能可由for 逻辑功能实现, 但运行在CPU, 难以高度并行
condition 必须是与 x, y 同型的1/0型 x, y可 broadcast
a = torch.rand(2, 2)
b = torch.ones(2, 2)
c = torch.zeros(2, 2)
torch.where(a>0.5, b, c)
```

2.5.2 GPU收集查表操作-gather

```
torch.gather(input, dim, index, out=None) --> Tensor 查表操作
out[i][j][k] = input[index[i][j][k]][j][k] dim=0
out[i][j][k] = input[i][index[i][j][k]][k] dim=1
out[i][j][k] = input[i][j][index[i][j][k]] dim=2
Gather 查表用来索引全局标签
prob = torch.rand(4, 10) 四张图片十个概率值
idx = prob.topk(3, dim=1)[1]
label = torch.arange(10)+100
torch.gather(label.expand(4, 10), dim=1, index=idx)
共四张图片每张查概率最大的三个标
```

3 随机梯度下降

3.1 激活函数

3.1.1 Sigmoid/Logistic

```
torch.sigmoid()
```

3.1.2 tanh

```
torch.tanh()
```

3.1.3 ReLu

```
torch.relu()
```

3.2 损失函数

3.2.1 MSE

```
from torch.nn import functional as F
mse = F.mse_loss(y, w*x+b)
```

3.2.2 Cross Entropy Loss

```
torch.nn.CrossEntropyLoss
criterion = nn.CrossEntropyLoss()
loss = criterion(sample, target)
```

3.2.3 Softmax

```
from torch.nn import functional as F
a = torch.rand(3, requires_grad=True)
p = F.softmax(a, dim=0)
a      #tensor([0.4588, 0.0768, 0.0897], requires_grad=True)
```

```
p    #tensor([0.4212, 0.2875, 0.2912], grad_fn=<SoftmaxBackward>)  
torch.autograd.grad(p[0], a, retain_graph=True) #output scalar  
#(tensor([ 0.2438, -0.1211, -0.1227]),)
```

3.3 求导方法

3.3.1 自动求导

```
torch.autograd.grad(loss, [w1, w2, ...])  
[w1 grad, w2 grad, ...]
```

3.3.2 反向回传求导数

```
loss.backward()  
w1.grad  
w2.grad  
...
```