🔦

Pytorch

> torch.Tensor若在cuda中则不能和np混用

> toch.Tensor作索引需要是long, byte, bool类型 (torch.Tensor.long())

🔬 Pytorch Experience

📝 Pytroch RNN

🚨 Pytorch Exception

▼ Content

- Basic Operations
  - Type Conversion
  - Operations
- Gradient
  - Loss
  - Optimizer
- Layers
  - Conv
    - nn.Conv1d()
  - BatchNorm
    - nn.BatchNorm1d()
    - nn.BatchNorm2d()
  - Pooling
    - nn.AdaptiveAvgPool2d()
    - nn.AdaptiveMaxPool2d()
- nn.Module
- Criterion

Basic Operations

Type Conversion

`torch.Tensor.device` : check device

`torch.Tensor.dtype` : check type of Tensor

to()

不仅可以用来转换类型，也可以转换设备

```
# device
torch.Tensor.to("cpu")
torch.Tensor.to("cuda:0")

# type
torch.Tensor.to(torch.float32)
torch.Tensor.to(torch.float64)

# 根据其他变量转换类型，注意这种方法会同时转换类型和设备
a: torch.Tensor
b: torch.Tensor
a = a.to(b)
```

int()

```
torch.Tensor.int()
torch.Tensor.long()
torch.Tensor.float()
```

type()

```
torch.Tensor.type(torch.FloatType)
```

> 💡 以上方法不会改变原值，需要将结果赋值回去

Operations

`torch.matmul()`

类似矩阵乘法，但可以广播

```
m1 = torch.ones((5,1,2))
m2 = torch.ones((1,2,5))
m3 = torch.matmul(m1,m2)
# => (5,1,5)

m1 = torch.ones((2,1,3,2))
m2 = torch.ones((5,2,4))
m3 = torch.matmul(m1,m2)
# => (2,5,3,4)
```

`torch.arange()`

torch.arange(10) = torch.Tensor([0,1,2,3,4,5,6,7,8,9])

`torch.repeat()`

当参数只有两个时：（列的重复倍数，行的重复倍数）。1表示不重复

当参数有三个时：（通道数的重复倍数，列的重复倍数，行的重复倍数）。

```
import torch
a= torch.arange(30).reshape(5,6)
print(a)
print('b:',a.repeat(2,2))
print('c:',a.repeat(2,1,1))
```

```
/usr/bin/python3 /home/thu/test_python/repeat.py
tensor([[ 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]])
b: tensor([[ 0,  1,  2,  3,  4,  5,  0,  1,  2,  3,  4,  5],
        [ 6,  7,  8,  9, 10, 11,  6,  7,  8,  9, 10, 11],
        [12, 13, 14, 15, 16, 17, 12, 13, 14, 15, 16, 17],
        [18, 19, 20, 21, 22, 23, 18, 19, 20, 21, 22, 23],
        [24, 25, 26, 27, 28, 29, 24, 25, 26, 27, 28, 29],
        [ 0,  1,  2,  3,  4,  5,  0,  1,  2,  3,  4,  5],
        [ 6,  7,  8,  9, 10, 11,  6,  7,  8,  9, 10, 11],
        [12, 13, 14, 15, 16, 17, 12, 13, 14, 15, 16, 17],
        [18, 19, 20, 21, 22, 23, 18, 19, 20, 21, 22, 23],
        [24, 25, 26, 27, 28, 29, 24, 25, 26, 27, 28, 29]])
c: tensor([[[ 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]],
        [[ 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]]]])

Process finished with exit code 0
```

`tensor.permute` & `tensor.transpose`

permute可以同时换多个维度，transpose只能换两个

Gradient

Loss

`loss.backforward()`后会将计算图销毁，如果要累积loss的`backforward`，需要

```
loss1.backward(retain_graph=True)
loss2.backward()
# 注意最后一个反向传播要销毁
```

Optimizer

```
import torch.optim as optim
optimizers = optim.Adam([
        {'params': net.decoders[0].parameters(), "lr": lr2},
        {"params": net.decoders[1].parameters(), "lr": lr2},
        {"params": net.decoders[2].parameters(), "lr": lr3},
        {"params": net.decoders[3].parameters(), "lr": lr2},
        {"params": net.decoders[4].parameters(), "lr": lr3}],
        lr=lr1)
# 给不同layer设置不同的初始学习率
```

Layers

Conv

nn.Conv1d()

> class torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)

一般来说，一维卷积`nn.Conv1d`用于文本数据，只对宽度进行卷积，对高度不卷积。通常，输入大小为`word_embedding_dim * max_length`，其中，`word_embedding_dim`为词向量的维度，`max_length`为句子的最大长度。卷积核窗口在句子长度的方向上滑动，进行卷积操作。

> 需要(B, C, N)，卷积核在最后一维移动

BatchNorm

nn.BatchNorm1d()

`torch.nn.BatchNorm1d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)`

- num_features – 特征维度
- eps – 为数值稳定性而加到分母上的值。
- momentum – 移动平均的动量值。
- affine – 一个布尔值，当设置为真时，此模块具有可学习的仿射参数。

> input可以是二维或者三维。当input的维度为（N, C）时，BN将对C维归一化；当input的维度为(N, C, L) 时，归一化的维度同样为C维。

nn.BatchNorm2d()

Input: (B, C, H, W)

通道为C

Pooling

nn.AdaptiveAvgPool2d()

`torch.nn.AdaptiveAvgPool2d(out_H, out_W)`

全局平均池化层，参数是输出的长宽，任何输入都能输出成想要的长宽

nn.AdaptiveMaxPool2d()

`torch.nn.AdaptiveMaxPool2d(out_H, out_W)`

nn.Module

load_state_dict()

```
# 
net.load_state_dict(state_dict)
```

Criterion

`nn.CrossEntropyLoss()`

```
criterion = nn.CrossEntropyLoss()
pred = torch.Tensor([[0.1,0.3,0.6],[0.5,0.4,0.1]])
label = torch.Tensor([2, 1]).long() # should be long
loss = criterion(pred, label)
```

`label` 表示哪些索引位置是标签，如上表示第一行的第3个和第二个的第2个是标签