2.PyTorch训练&测试神经网络基础流程

笔记本： 【课】原理-李宏毅 deep leaning
创建时间： 2023/4/18 17:23 **更新时间：** 2023/4/21 11:58
作者： 1256876216@qq.com

使用名词

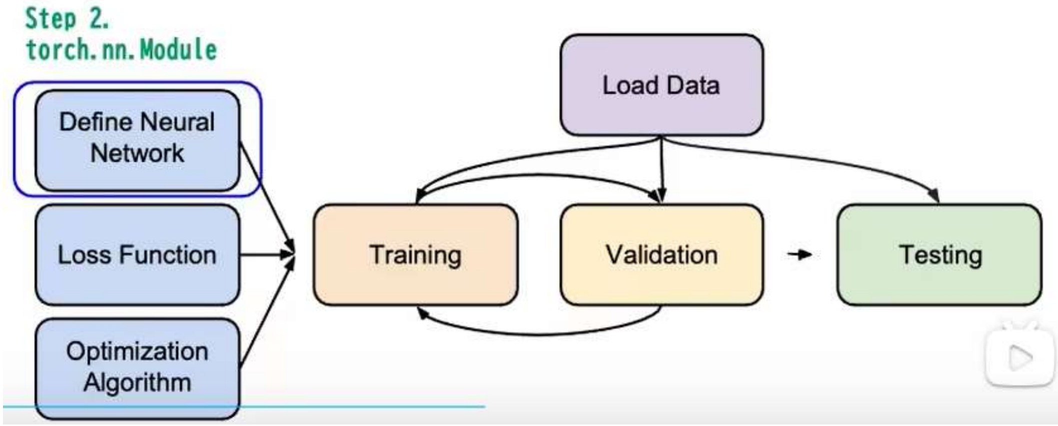
1. 名词&形容词

   shuffle: 洗牌，打乱次序, 值true（training），false（testing）
   tensor：张量
   convex： 凸面的
   optimal: 最优解
   accuracy：精确
   goodness： 吻合度
   Stochastic： 随机的

2. 公式处理

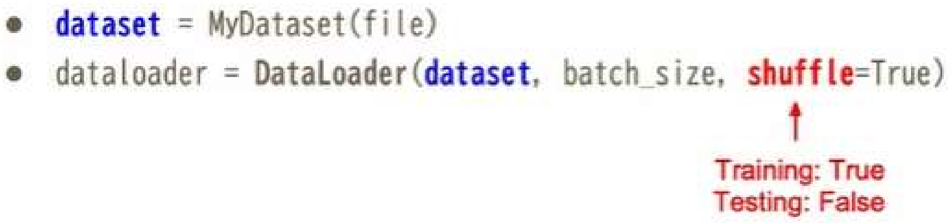
   transpose： 转换两个维度坐标进行翻转，行列互换。torch.zero([2,3])；x=x.transpose(0,1)

训练&测试神经网络基础流程



一、处理数据

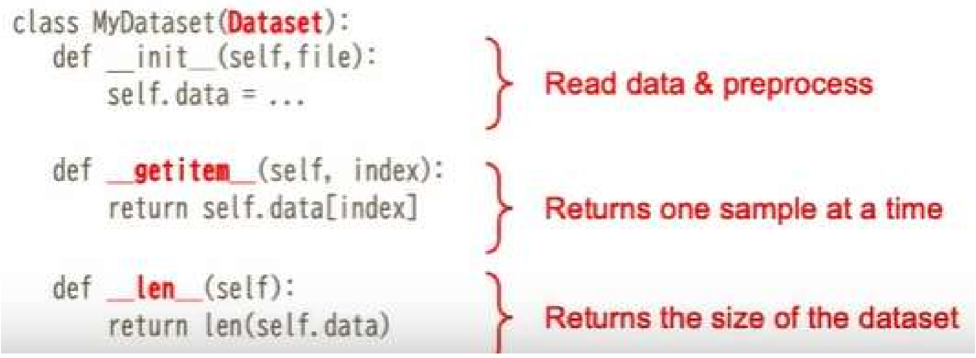
1. Dataset & Dataloader 数据集及数据处理loader



```
• dataset = MyDataset(file)
• dataloader = DataLoader(dataset, batch_size, shuffle=True)
```

Training: True
Testing: False

可重写Dataset和Dataloader



```
class MyDataset(Dataset):
    def __init__(self,file):
        self.data = ...

    def __getitem__(self, index):
        return self.data[index]

    def __len__(self):
        return len(self.data)
```

Read data & preprocess

Returns one sample at a time

Returns the size of the dataset

2. tensors 向量

- 创建tensors

```
直接由数组转换
x=torch.tensor([[-1,1],[-1,1]])
x=torch.from_numpy(np.array[[-1,1],[-1,1]])
从常量转换shape变形
x=torch.zeros([2,2])
x=torch.ones([1,2,3])
```

- tensors常用操作:

```
y=x.sum() // 求和
y=x.mean() // 平均值
y=x.pow(2) //幂
x.shape  // torchh.Size([2,3)
x=x.transpose(0,1) // 将0,1位置坐标进行互换
x=x.squeeze(0)  // 拿掉位置的维度，降维
x=x.squeeze(1)  // 位置上插入长度为1的维度，升维
w=torch.cat([x,y,z], dim=1) // 沿着dim方向拼接，保证其他维一致
```

- 数据类型： float 32位 long 64 位
- 指定运行设备: 默认CPU， x=x.to('cuda') 转到GPU上运行 torch.cuda.iis_available()判断是否有GPU
- 梯度下降计算

```
x= torch.tensor([[1., 0.],[-1., 1.]], require_grad=true) // 二维向量
z=x.pow(2).sum() // 每项平方求和
z.backward() // z对每项做微分
x.grad  // 查看结果 tensor([[2., 0.],[-2., 2.]])
```

二、 定义神经网络

1. 使用模型包nn

eg. nn.Linear(32, 64) 输入32维，输出64维，全连接模型 W(64*32) * x + b = y

```
layer= torch.nn.Linear(32, 64)
layer.weight.shape  // torch.Size([64, 32])
layer.bias.shape  // torch.Size([64])
```

2. 选择激活函数

nn.Sigmoid()
nn.ReLU()

```
import torch.nn as nn
方式一:
class MyModel(nn.model):
    def __init__(self):   # 初始化模型&定义层
```

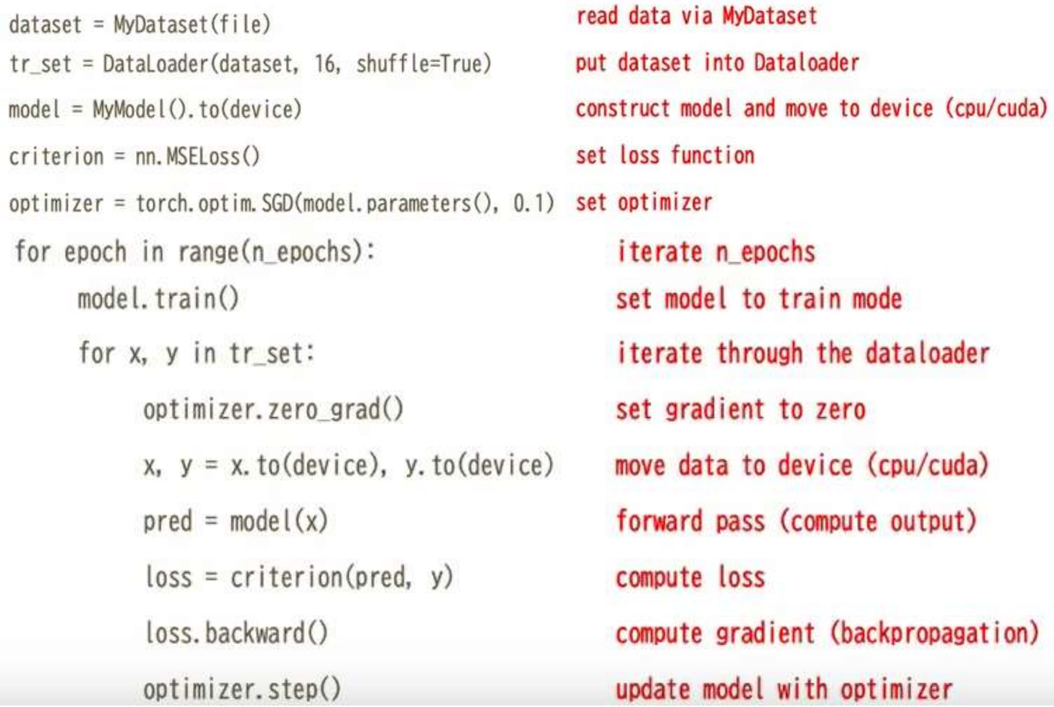
```
        super(Mymodel, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(10, 32),
            nn.Sigmoid(),
            nn.Lineat(32, 1)
        )
    def forward(self, x):
        return self.net(x)
方式二:
class MyModel(nn.model):
    def __init__(self):
        super(Mymodel, self).__init__()
        self.layer1 = nn.Linear(10,32)
        self.layer2 = nn.Sigmoid()
        self.layer3 = nn.Lineat(32, 1)
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer(out)
        return out
```

3. 最佳化演算法调参

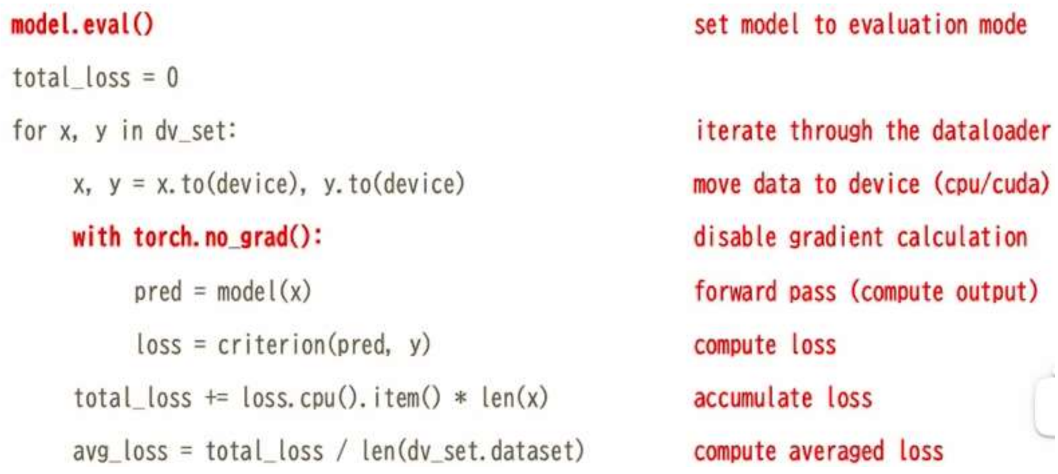
调用torch.optim库
SDG: Stochastic Gradient Decent

```
optimizer = torch.optim.SDG(model.parameters(), lr, momentum=0)
步骤:
调用 optimizer.zero_grad() 重置模型参数的梯度
调用 loss.backward() 利用gradients调整模型参数
调用 optimizer.step() 调整模型参数
```

三、完整训练过程



```
dataset = MyDataset(file)                                  read data via MyDataset
tr_set = DataLoader(dataset, 16, shuffle=True)             put dataset into Dataloader
model = MyModel().to(device)                               construct model and move to device (cpu/cuda)
criterion = nn.MSELoss()                                   set loss function
optimizer = torch.optim.SGD(model.parameters(), 0.1)       set optimizer
for epoch in range(n_epochs):                              iterate n_epochs
    model.train()                                          set model to train mode
    for x, y in tr_set:                                    iterate through the dataloader
        optimizer.zero_grad()                              set gradient to zero
        x, y = x.to(device), y.to(device)                  move data to device (cpu/cuda)
        pred = model(x)                                    forward pass (compute output)
        loss = criterion(pred, y)                          compute loss
        loss.backward()                                    compute gradient (backpropagation)
        optimizer.step()                                   update model with optimizer
```



```
model.eval()                                  set model to evaluation mode
total_loss = 0
for x, y in dv_set:                           iterate through the dataloader
    x, y = x.to(device), y.to(device)         move data to device (cpu/cuda)
    with torch.no_grad():                     disable gradient calculation
        pred = model(x)                       forward pass (compute output)
        loss = criterion(pred, y)             compute loss
    total_loss += loss.cpu().item() * len(x)  accumulate loss
    avg_loss = total_loss / len(dv_set.dataset)  compute averaged loss
```

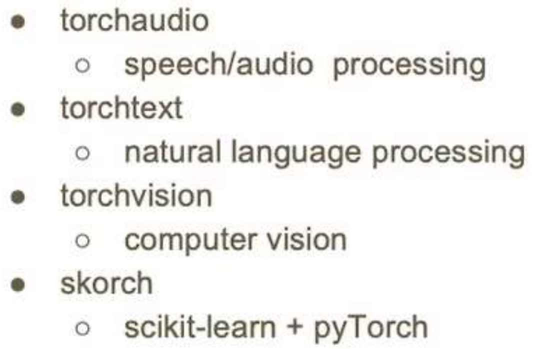
注意：

1. model.eval() 改变某些层的行为， 例如： dropout （随机让某些节点不起作用） & batch normalization (批数据 标准化)
2. with torch.no_grad() 避免在validation/testing过程中影响参数（学习数据）
3. 保存 torch.save(model.state_dic(), path)
4. 加载load ckpt= torch.load(path) model.load_state_dic(clpt)

四、PyTorch 应用&参考网址

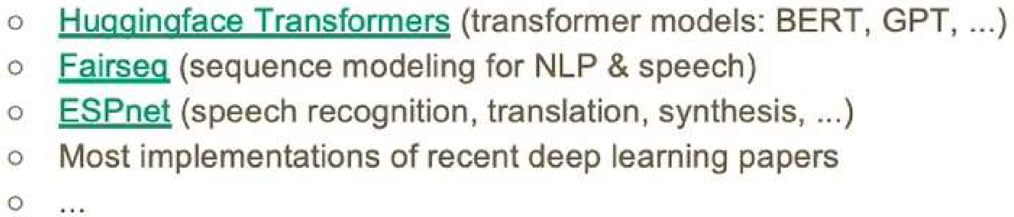
官网

https://pytorch.org/docs/stable/

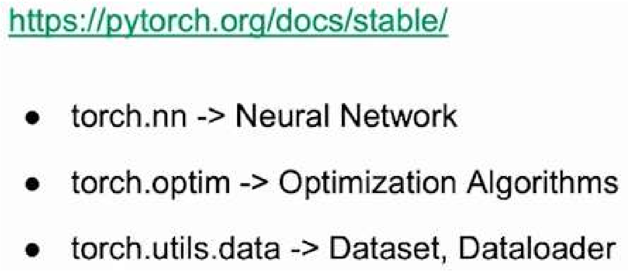


- torchaudio
  - speech/audio processing
- torchtext
  - natural language processing
- torchvision
  - computer vision
- skorch
  - scikit-learn + pyTorch

仓库：



- Huggingface Transformers (transformer models: BERT, GPT, ...)
- Fairseq (sequence modeling for NLP & speech)
- ESPnet (speech recognition, translation, synthesis, ...)
- Most implementations of recent deep learning papers
- ...



https://pytorch.org/docs/stable/

- torch.nn -> Neural Network
- torch.optim -> Optimization Algorithms
- torch.utils.data -> Dataset, Dataloader