第三周

1. 模型创建与nn.Module

1.1 网络模型创建步骤

- 模型创建
  - 构建网络层：卷积层/池化层/激活函数层，`__init__()`
  - 拼接网络层：LeNet/AlexNet/ResNet，`forward()`
- 权值初始化：Xavier/Kaiming/均匀分布/正态分布

1.2 nn.Module属性

- `torch.nn`
  - `nn.Parameter`：张量子类，表示可学习参数，如权重/偏置
  - `nn.Module`：所有网络层基类，管理网络属性
  - `nn.functional`：函数具体实现，如卷积/池化/激活函数等
  - `nn.init`：参数初始化方法
- `nn.Module`
  - parameters：存储管理 `nn.Parameter`类
  - modules：存储管理 `nn.Module`类
  - buffers：存储管理缓冲属性，如BN层中的 `running_mean`
  - ***_hooks：存储管理钩子函数
- 总结
  - 一个module可以包含多个子module
  - 一个module相当于一个运算，必须实现 `forward()`函数
  - 每个module都有8个字典管理它的属性

2. 模型容器与AlexNet构建

2.1 模型容器

- Containers
  - `nn.Sequetial`：顺序性，按**顺序**包装多个网络层
  - `nn.ModuleList`：迭代性，像**list**一样包装多个网络层
  - `nn.ModuleDict`：索引性，像**dict**一样包装多个网络层
- `nn.Sequetial`
  - 按顺序包装一组网络层
  - 顺序型：各网络层之间严格按照顺序构建
  - 自带forward()：通过for循环依次执行前向传播运算
  - **LeNet模块 = features子模块 + classifier子模块**

```
# Conv1-pool1-Conv2-pool2-fc1-fc2-fc3
self.features = nn.Sequential(
    nn.Conv2d(3,6,5),
    nn.ReLU(),
```

```
    nn.MaxPool2d(kernel_size=2,stride=2),
    nn.Con2d(6,16,5),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2,stride=2),)
self.classifier = nn.Sequential(
    nn.Linear(16*5*5,120),
    nn.ReLU(),
    nn.Linear(120,84),
    nn.ReLU(),
    nn.Linear(84,classes),)
# 增加命名
self.features = nn.Sequential(OrderedDict({
    'conv1': nn.Conv2d(3, 6, 5),
    'relu1': nn.ReLU(inplace=True),
    'pool1': nn.MaxPool2d(kernel_size=2, stride=2),
    'relu2': nn.ReLU(inplace=True),
    'pool2': nn.MaxPool2d(kernel_size=2, stride=2),}))
self.classifier = nn.Sequential(OrderedDict({
    'fc1': nn.Linear(16*5*5, 120),
    'relu3': nn.ReLU(),
    'fc2': nn.Linear(120, 84),
    'relu4': nn.ReLU(inplace=True),
    'fc3': nn.Linear(84, classes),}))
# 前向传播
def forward(self, x):
    x = self.features(x)
    x = x.view(x.size()[0], -1)
    x = self.classifier(x)
    return x
```

- `nn.ModuleList`
  - 以迭代方式调用网络层
  - `append()`：在ModuleList后面添加网络层
  - `extend()`：拼接两个ModuleList
  - `insert()`：指定在ModuleList中位置插入网络层

```
# 20个fc
self.linears = nn.ModuleList(
    [nn.Linear(10, 10) for i in range(20)])
# 前向传播
def forward(self, x):
    for i, linear in enumerate(self.linears):
        x = linear(x)
    return x
```

- `nn.ModuleDict`
  - 以索引方式调用网络层
  - `clear()`：清空ModuleDict
  - `items()`：返回可迭代的键值对
  - `keys()`：返回字典的键
  - `values()`：返回字典的值
  - `pop()`：返回一对键值，并从字典中删除

```
self.choices = nn.ModuleDict({
    'conv': nn.Conv2d(10, 10, 3),
    'pool': nn.MaxPool2d(3)})
self.activations = nn.ModuleDict({
    'relu': nn.ReLU(),
    'prelu': nn.PReLU()})
# 前向传播
def forward(self, x, choice, act):
    x = self.choices[choice](x)
    x = self.activations[act](x)
    return x
```

2.2 AlexNet构建

- AlexNext特点
  - 采用ReLU，替换饱和激活函数Sigmoid，减轻梯度消失
  - 采用LRU，对数据归一化，减轻梯度消失
  - 采用Dropout，提高全连接层鲁棒性，增加模型泛化能力
  - 采用数据增强：TenCrop，色彩修改

```
alexnet = torchvision.models.AlexNet()
```

2.3 作业

第三周作业1

3. nn网络层-卷积层

3.1 1d/2d/3d卷积

- 卷积过程
  - 卷积核：又称为滤波器，可认为是某种特征
  - 卷积运算：卷积核在输入信号上滑动，相应位置上进行乘加
  - 卷积过程类似于一个模板去图像上寻找与它相似的区域，与卷积核越相似，激活值越高，从而实现特征提取
- 卷积维度
  - 一般情况下，卷积核在几个维度上滑动，就是几维卷积

3.2 卷积-nn.Conv2d()

```
# 对多个二维信号进行二维卷积
nn.Conv2d(in_channels,     # 输入通道数
        out_channels,     # 输出通道数，等价于卷积核个数
        kernel_size,     # 卷积核尺寸
        stride=1,     # 步长
        padding=0,     # 填充个数，保持输入输出的尺寸一致
        dilation=1,     # 空洞卷积大小
        groups=1,     # 分组卷积设置
        bias=True,     # 偏置
        padding_mode='zeros')
```

简化版

$$Out_{size} = (In_{size} - kernel_{size})/stride + 1$$

完整版

$$Out_{size} = \frac{In_{size} + 2 \times padding - dilation \times (kernel_{size} - 1) - 1}{stride} + 1$$

其中，out_{size} 为输出尺寸；In_{size} 为输入尺寸；$kernel_{size}$ 为内核尺寸；$stride$ 为步长；$padding$ 为填充个数；$dialtion$ 为空洞卷积大小

3.3 转置卷积-nn.ConvTranspose

- 转置卷积又称为反卷积和部分跨越卷积，用于对图像上采样
- 使用矩阵乘法实现卷积操作：
  - 常规卷积
    - 假设图像尺寸44，*卷积核*3，padding=0，stride=1
    - 图像：I_{16*1}；卷积核 K_{4*16}；
    - 输出：$O_{4*1} = K_{4*16} * I_{16*1}$；再reshape为 $(2 * 2)$
  - 转置卷积
    - 假设图像尺寸2*2，卷积核3*3，padding=0，stride=1
    - 图像：I_{4*1}；卷积核 K_{16*4}；
    - 输出：$O_{16*1} = K_{16*4} * I_{4*1}$；再reshape为 $(4 * 4)$

```
# 转置卷积实现上采样
nn.ConvTranspose2d(in_channels,     # 输入通道数
        out_channels,     # 输出通道数，等价于卷积核个数
        kernel_size,     # 卷积核尺寸
        stride=1,     # 步长
        padding=0,     # 填充个数，保持输入输出的尺寸一致
        output_padding=0,
        groups=1,     # 分组卷积设置
        bias=True,     # 偏置
        dilation=1,     # 空洞卷积大小
        padding_mode='zeros')
```

简化版

$$Out_{size} = (In_{size} - 1) * stride + kernel_{size}$$

完整版

$$Out_{size} = (In_{size} - 1) \times stride - 2 * padding + dilation \times (kernel_{size} - 1) + out\_padding + 1$$

4. nn网络层-池化层、线性层、激活函数层

4.1 池化层

- 池化运算：对信号进行**收集**并**总结**，类似水池收集水资源，用于减少冗余信息，降低后续的计算量
  - **收集**：由多变少
  - **总结**：最大值/平均值

```
# 1. 最大池化
nn.MaxPool2d(kernel_size,     # 池化核尺寸
        stride=None,     # 步长，通常与kernel_size相同
```

```
        padding=0,     # 填充个数
        dilation=1,     # 池化核间隔大小
        return_indices=False,     # 记录池化像素索引
        ceil_mode=False)     # 尺寸向上取整
# 2. 平均池化
nn.AvgPool2d(kernel_size,     # 池化核尺寸
        stride=None,     # 步长，通常与kernel_size相同
        padding=0,     # 填充个数
        ceil_mode=False,     # 尺寸向上取整
        count_include_pad=True,     # 填充值用于计算
        divisor_override=None)     # 除法因子
# 3. 最大值反池化
nn.MaxUnpool2d(kernel_size,     # 池化核尺寸
        stride=None,     # 步长，通常与kernel_size相同
        padding=0)     # 填充个数
# 在前向传播的过程中要添加反池化的索引值indices
forward(self, input, indices, output_size=None)
```

- 最大池化后的图片亮度**大于**平均池化的图片亮度

4.2 线性层

- 线性层又称为全连接层，其每个神经元与上一层所有神经元相连，实现对前一层的线性组合

```
# 对一维信号进行线性组合
nn.Linear(in_features,     # 输入结点数
    out_features,     # 输出结点数
    bias=True)     # 是否需要偏置
```

计算公式

$$y = xW^T + bias$$

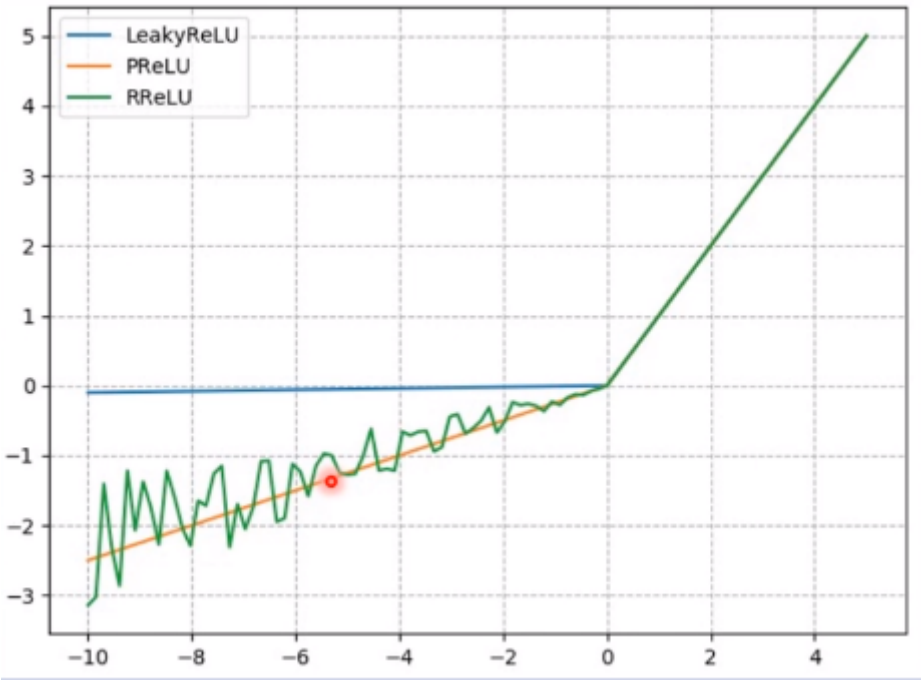
4.3 激活函数层

- 激活函数对特征进行非线性变换，赋予多层神经网络具有深度的意义
- **若无激活函数，多个线性层叠加等价于一个线性层**

激活函数

1. `nn.Sigmoid`
    - 计算公式：$y = \frac{1}{1+e^{-x}}$
    - 梯度公式：$y' = y * (1 - y)$
    - 特性：
        - 输出值在 $(0, 1)$，符合概率的取值范围
        - 导数取值范围 $[0, 0.25]$，容易出现梯度消失的现象
        - 输出值大于0，会破坏数据0均值分布
2. `nn.tanh`
    - 计算公式：$y = \frac{\sin x}{\cos x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1+e^{-2x}} + 1$
    - 梯度公式：$y' = 1 - y^2$
    - 特性：
        - 输出值在 $(-1, 1)$，符合数据0均值分布

        - 导数取值范围 $(0, 1)$，容易出现梯度消失的现象
3. `nn.ReLU`
    - 计算公式：$y = max(0, x)$
    - 梯度公式：$y' = \begin{cases} 1, & x > 0 \\ \text{undefined}, & x = 0 \\ 0, & x < 0 \end{cases}$
    - 特性：
        - 输出值均为正数，由于负半轴值为 0 会导致死神经元
        - 导数值为 1，缓解了梯度消失，但易产生梯度爆炸的现象
4. 改进的ReLU激活函数
    - `nn.LeakyReLU`
        - `negative_slope`：负半轴斜率，值很小
    - `nn.PReLU`
        - `init`：可学习斜率
    - `nn.RReLU`
        - `lower`：均匀分布下限
        - `upper`：均匀分布上限



4.4 作业

第三周作业2