智能计算系统第一次作业

学号：2112193　姓名：王可一 专业：物联网工程

1.2 简述人工智能研究的三个学派

人工智能按研究学派主要分为三类，包括行为主义(Behaviorism)，符号主义(Symbolism)，连接主义(Connectionism)。

行为主义：以控制论为基础，构建感知-动作型控制系统。这种方法通过模拟反馈机制，实现机器人的行走、抓取、平衡等任务，具有实用性。但并非通向强人工智能的终极道路。

符号主义：基于符号逻辑，将知识和问题表示为逻辑表达式，并用逻辑推理解决问题。然而，存在着难以找到简洁的符号逻辑体系、无法完全表达所有知识以及逻辑求解器不可判定等问题。

连接主义：借鉴大脑中神经元连接的计算模型，使用人工神经网络来拟合智能行为。从最初的神经元模型到深度学习，连接主义逐渐成为人工智能的主流方向。然而，深度学习存在泛化能力有限、缺乏推理能力、可解释性差等局限性。

这三种方法各有优劣，并没有一种方法是通向强人工智能的终极道路。行为主义适用于反应性控制系统，符号主义强调逻辑推理，而连接主义则着重于神经网络和深度学习，但它们都存在各自的限制和挑战

2.3 反向传播中，神经元的梯度是如何计算的？权重是如何更新的?

基于梯度下降法的神经网络反向传播过程首先需要根据应用场景定义合适损失函数（loss function），常用的损失函数有均方差损失函数和交叉熵损失函数。确定了损失函数之后，把网络的实际输出与期盼输出结合损失函数计算loss，如果loss不符合预期，则对loss分别求权重和偏置的偏导数，然后沿梯度下降方向更新权重及偏置参数。

不引入惩罚项的权重更新公式如下：

$$w \leftarrow w - \eta(\nabla_w L(w; x, y))$$

2.5 请简述三种避免过拟合问题的方法

正则化是通过向损失函数添加惩罚项来降低模型的复杂度，从而减少过拟合的风险。常见的正则化方法包括L1正则化和L2正则化。L1正则化倾向于产生稀疏权重，而L2正则化倾向于平滑权重。正则化的选择取决于具体问题和模型的需求。

将**数据集划分**为训练集、验证集和测试集。模型在训练集上进行参数更新，利用验证集进行模型调优，最终在测试集上评估模型的性能。这样可以确保模型在未见过的数据上的泛化能力。

Bagging（Bootstrap Aggregating）集成学习是一种集成学习方法，通过构建多个分类器（或回归器）并将它们的预测结果进行组合来改善整体的预测性能。它的基本思想是通过随机采样训练数据的子集来构建多个模型，然后将这些模型的预测结果进行平均或投票来得到最终的预测结果。

2.7 假设激活函数的表达式如下，请给出它的导数形式并求出其在原点的取值

$$\phi(v) = \frac{v}{\sqrt{1+v^2}}$$

$$\frac{\mathrm{d}\phi(v)}{\mathrm{d}v} = \frac{\mathrm{d}v * (1+v^2)^{-1/2}}{\mathrm{d}v} = \frac{\mathrm{d}(1+v^2)(1+v^2)^{-3/2} + \mathrm{d} - v^2(1+v^2)^{-3/2}}{\mathrm{d}v} = \frac{1}{(1+v^2)^{3/2}}$$

当在原点处时，该激活函数的导数取值为1。

3.2 计算习题3.1中三个网络（即AlexNet、VGG19、ResNet152）完成一次正向过程所需要的乘法数量和加法数量。

计算三个网络（AlexNet、VGG19、ResNet152）完成一次正向过程所需要的乘法数量和加法数量，需要考虑每个网络中每一层的运算量，然后将它们相加以得到总的乘法数量和加法数量。即对于每个网络，遍历每一层，计算每一层的乘法数量和加法数量。这通常包括卷积层、全连接层等需要大量计算的层，将每一层的乘法数量和加法数量相加，得到总的乘法数量和加法数量。因计算全部过于繁琐，故只计算卷积层和全连接层的数目情况。

（1）AlexNet

```
import torch
import torchvision.models as models

def count_operations(model, input_size=(3, 224, 224)):
    # 将模型设置为评估模式
    model.eval()

    # 生成随机输入数据
    inputs = torch.randn(1, *input_size)

    # 将模型移到GPU（如果可用）
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.to(device)
    inputs = inputs.to(device)

    # 定义计数器
    total_mults = 0
    total_adds = 0

    # 遍历模型的每一层
    for module in model.modules():
        if isinstance(module, torch.nn.Conv2d):
            # 计算卷积层的乘法和加法数量
            in_channels = module.in_channels
```

```
            out_channels = module.out_channels
            kernel_size = module.kernel_size[0] * module.kernel_size[1]
            out_size = inputs.size(2) * inputs.size(3)
            total_mults += in_channels * out_channels * kernel_size * out_size
            total_adds += (in_channels * out_channels * kernel_size - 1) *
out_size
        elif isinstance(module, torch.nn.Linear):
            # 计算全连接层的乘法和加法数量
            in_features = module.in_features
            out_features = module.out_features
            total_mults += in_features * out_features
            total_adds += (in_features * out_features - 1)

    return total_mults, total_adds

# 加载AlexNet模型
alexnet = models.alexnet()

# 计算操作数量
mults, adds = count_operations(alexnet)
print("乘法数量:", mults)
print("加法数量:", adds)
```

```
乘法数量: 123920285696
加法数量: 123920034813
```

故由此可得，AlexNet的乘法数目为123920285696，加法数目为123920034813。

(2) VGG19

```
import torch
import torchvision.models as models

def count_operations(model, input_size=(3, 224, 224)):
    # 将模型设置为评估模式
    model.eval()

    # 生成随机输入数据
    inputs = torch.randn(1, *input_size)

    # 将模型移到GPU（如果可用）
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.to(device)
    inputs = inputs.to(device)

    # 定义计数器
    total_mults = 0
    total_adds = 0

    # 遍历模型的每一层
    for module in model.modules():
        if isinstance(module, torch.nn.Conv2d):
            # 计算卷积层的乘法和加法数量
            in_channels = module.in_channels
```

```
            out_channels = module.out_channels
            kernel_size = module.kernel_size[0] * module.kernel_size[1]
            out_size = inputs.size(2) * inputs.size(3)
            total_mults += in_channels * out_channels * kernel_size * out_size
            total_adds += (in_channels * out_channels * kernel_size - 1) *
out_size
        elif isinstance(module, torch.nn.Linear):
            # 计算全连接层的乘法和加法数量
            in_features = module.in_features
            out_features = module.out_features
            total_mults += in_features * out_features
            total_adds += (in_features * out_features - 1)

    return total_mults, total_adds

# 加载VGG19模型
vgg19 = models.vgg19()

# 计算操作数量
mults, adds = count_operations(vgg19)
print("乘法数量:", mults)
print("加法数量:", adds)
```

```
乘法数量: 1004590956544
加法数量: 1004590153725
```

故由此可得，VGG19的乘法数目为 1004590956544，加法数目为1004590153725。

(3) ResNet152

```
import torch
import torchvision.models as models

def count_operations(model, input_size=(3, 224, 224)):
    # 将模型设置为评估模式
    model.eval()

    # 生成随机输入数据
    inputs = torch.randn(1, *input_size)

    # 将模型移到GPU（如果可用）
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.to(device)
    inputs = inputs.to(device)

    # 定义计数器
    total_mults = 0
    total_adds = 0

    # 遍历模型的每一层
    for module in model.modules():
        if isinstance(module, torch.nn.Conv2d):
            # 计算卷积层的乘法和加法数量
            in_channels = module.in_channels
```

```
            out_channels = module.out_channels
            kernel_size = module.kernel_size[0] * module.kernel_size[1]
            out_size = inputs.size(2) * inputs.size(3)
            total_mults += in_channels * out_channels * kernel_size * out_size
            total_adds += (in_channels * out_channels * kernel_size - 1) *
out_size
        elif isinstance(module, torch.nn.Linear):
            # 计算全连接层的乘法和加法数量
            in_features = module.in_features
            out_features = module.out_features
            total_mults += in_features * out_features
            total_adds += (in_features * out_features - 1)

    return total_mults, total_adds

# 加载ResNet-152模型
resnet152 = models.resnet152()

# 计算操作数量
mults, adds = count_operations(resnet152)
print("乘法数量:", mults)
print("加法数量:", adds)
```

```
乘法数量: 2909827907584
加法数量: 2909820130303
```

故由此可得，ResNet152的乘法数目为 2909827907584，加法数目为2909820130303。