知识蒸馏

目录

1.摘要

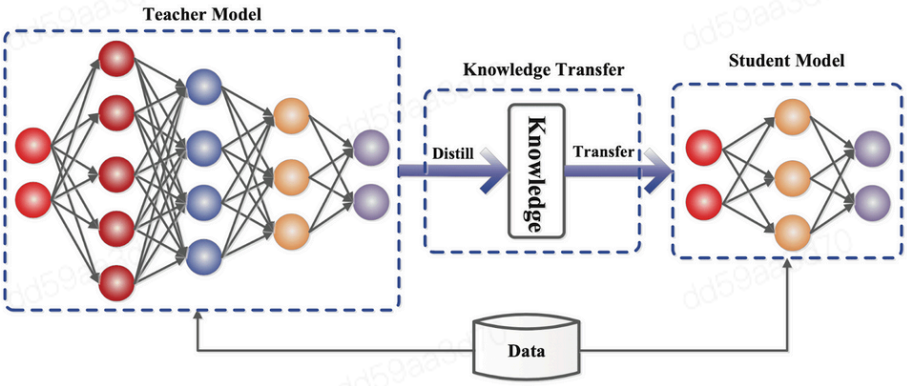
知识蒸馏（Knowledge Distillation）是深度学习领域的一种技术，旨在将一个复杂模型（通常称为“教师模型”）的知识转移到一个结构更简单、计算成本更低的模型（称为“学生模型”）中。这一概念最早由Hinton等人在2015年提出，目的是解决部署在资源受限环境下的深度学习模型的问题。

提出背景

1. **模型规模的增大**：随着深度学习技术的发展，模型的复杂度和规模不断增大。这些大模型虽然在训练数据上表现出色，但在实际部署时，尤其是在计算资源受限的设备上，如移动设备或嵌入式系统上，其庞大的计算需求成为了一个问题。
2. **计算资源的限制**：在很多实际应用场景中，尤其是在边缘计算设备上，由于硬件性能的限制，无法直接运行大型复杂的深度学习模型。
3. **能源消耗的考虑**：大模型通常需要更多的能源来支持其计算，这在能源敏感或成本敏感的应用中是不可接受的。
4. **实时性要求**：在需要快速响应的应用中，大型模型因计算延迟较大可能无法满足需求。

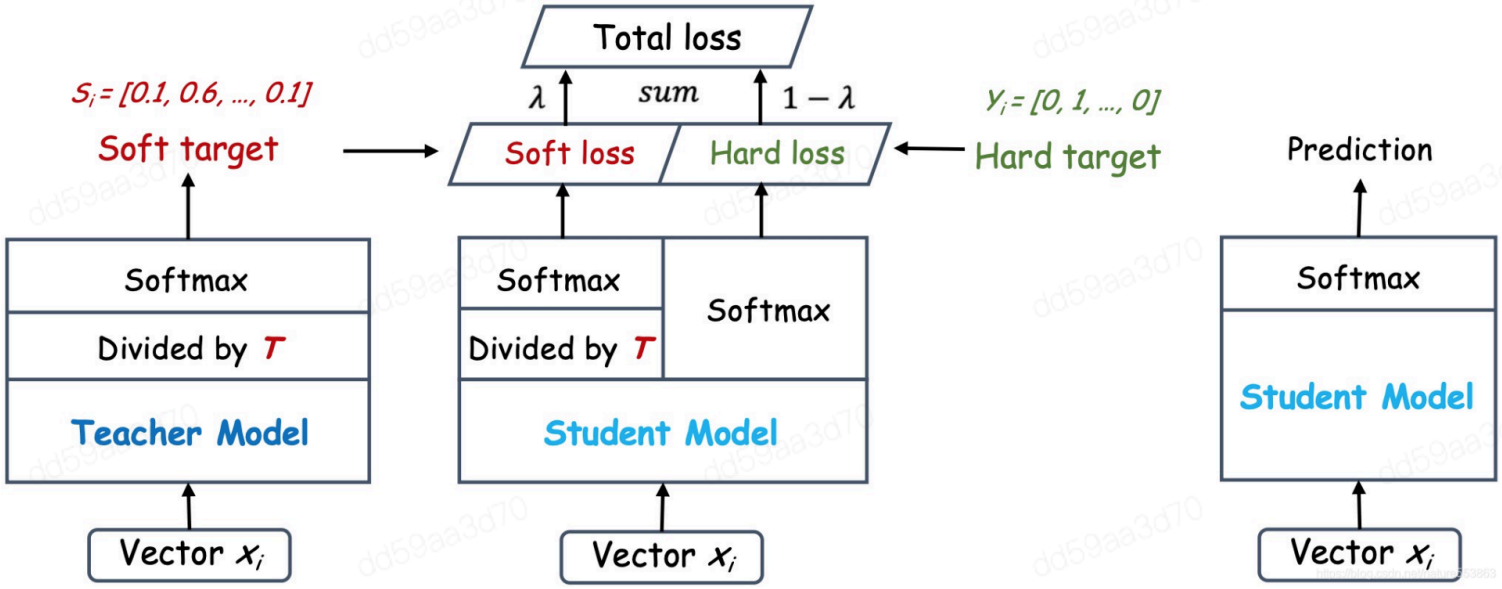
2.业务概述

知识蒸馏基于这样的需求提出：通过一种"软化"的信息传递方式（通常是通过温度调整softmax输出），使得简化的学生模型可以学习到复杂教师模型的行为特征和决策方式，而不仅仅是模仿最终输出的硬标签（即直接的分类结果）。这种方法使得学生模型在体积更小、运算速度更快的情况下，也能达到接近教师模型的表现。



`

2.1 业务视图



本节简要描述从系统外部看到的系统的特征，比如谁使用该系统解决什么业务问题，该业务问题原先是如何解决的，该系统是如何改进的。

工作过程

1. **输入数据（Input Data）**：
   - 输入向量 `xi` 表示输入数据，它被同时送到教师模型和学生模型中。
2. **教师模型（Teacher Model）**：
   - 教师模型是一个训练好的大型深度学习模型。它的输出是一个softmax概率分布，这里被称为“软目标”（Soft target）。
   - 软目标在温度 `T` 的调节下变得更加“软”，即概率分布更加平滑，包含了更多关于各个类别的信息。
3. **学生模型（Student Model）**：
   - 学生模型结构更简单，参数更少。它的目的是模仿教师模型的行为。
   - 学生模型也使用了相同的温度 `T` 来软化它的softmax输出，以便与教师模型的输出进行比较。
4. **损失函数（Loss Function）**：
   - 损失函数由两部分组成：软损失和硬损失。

- 软损失计算学生模型的软化概率分布与教师模型的软化概率分布之间的差异，目的是让学生模型学习到教师模型的行为。
- 硬损失计算学生模型的输出（在正常温度下）与实际的标签（Hard target）之间的差异。
- 这两个损失按照参数λ（lambda）的权重进行结合，形成总损失，用于学生模型的训练。

5. **输出预测（Output Prediction）**：
   - 经过训练的学生模型能够在新数据上进行预测。

在知识蒸馏的上下文中，总损失（Total Loss）是由两部分组成的：软损失（Soft Loss）和硬损失（Hard Loss）。这两个损失通常是通过一个加权和来结合的。具体的总损失计算公式如下：

总损失（L_total）= λ * L_soft + (1 - λ) * L_hard

其中：

- Lsoft$L$soft 是软损失，通常通过教师模型和学生模型的软目标之间的交叉熵来计算。
- Lhard$L$hard 是硬损失，是学生模型的预测输出和真实标签之间的交叉熵。
- λ (lambda) 是一个介于0和1之间的超参数，用于调节软损失和硬损失在总损失中的比重。

具体的软损失计算公式通常为：

Lsoft=∑$i$$t_i$·log($s_i$)$L$soft=∑$i$$t_i$·log($s_i$)

$$L_{\text{soft}} = \sum_i t_i \cdot \log(s_i)$$

- $t_i$$t_i$ 是教师模型的输出概率（已通过温度T软化）。
- $s_i$$s_i$ 是学生模型的输出概率（同样经过温度T软化）。
- loglog 表示自然对数。

硬损失计算公式是标准的交叉熵损失：

Lhard=−∑$i$$y_i$·log($p_i$)$L$hard=−∑$i$$y_i$·log($p_i$)

$$L_{\text{hard}} = -\sum_i y_i \cdot \log(p_i)$$

- $y_i$$y_i$ 是真实标签的one-hot编码。
- $p_i$$p_i$ 是学生模型在正常softmax下的预测概率。

在实际应用中，温度T也会影响软损失的计算，需要在计算交叉熵之前应用温度调整。此外，交叉熵损失后通常会乘以温度的平方进行缩放，以保持梯度的大小不随温度调整而改变。因此，带温度调节的软损失公式可以写为：

Lsoft=∑$i$$t_i$·log($s_i$)$L$soft=∑$i$$t_i$·log($s_i$) 其中 $t_i$$t_i$ 和 $s_i$$s_i$ 都是经过温度T调整的概率分布。

$$L_{\text{soft}} = \sum_i t_i \cdot \log(s_i)$$

2.2 功能描述

Python

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

# CIFAR-10 数据集的预处理
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# 加载训练集和测试集
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=128, shuffle=False, num_workers=2)

# 定义简单的教师和学生网络
class TeacherNet(nn.Module):
    def __init__(self):
        super(TeacherNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(64 * 16 * 16, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = x.view(-1, 64 * 16 * 16)
        x = self.fc(x)
        return x

class StudentNet(nn.Module):
    def __init__(self):
        super(StudentNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
```

```
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(16 * 16 * 16, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = x.view(-1, 16 * 16 * 16)
        x = self.fc(x)
        return x

# 初始化教师和学生网络
teacher_net = TeacherNet()
student_net = StudentNet()

# 定义优化器和损失函数
optimizer_teacher = optim.SGD(teacher_net.parameters(), lr=0.001, momentum=0.9)
optimizer_student = optim.SGD(student_net.parameters(), lr=0.001, momentum=0.9)
criterion = nn.CrossEntropyLoss()

# 训练教师模型
def train_teacher(model, trainloader, optimizer, num_epochs=10):
    model.train()
    for epoch in range(num_epochs):
        for inputs, labels in trainloader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

train_teacher(teacher_net, trainloader, optimizer_teacher)

# 蒸馏过程
def distillation(y, labels, teacher_scores, T, alpha):
    return nn.KLDivLoss()(F.log_softmax(y/T, dim=1), F.softmax(teacher_scores/T, dim=1)) * (T*T * 2.0 * alpha) + \
           F.cross_entropy(y, labels) * (1. - alpha)

# 训练学生模型
def train_student(teacher_net, student_net, trainloader, optimizer_student, T=20.0, alpha=0.7):
    teacher_net.eval()
    student_net.train()
    for data in trainloader:
        inputs, labels = data
```

```
        optimizer_student.zero_grad()

        with torch.no_grad():
            teacher_outputs = teacher_net(inputs)

        student_outputs = student_net(inputs)
        loss = distillation(student_outputs, labels, teacher_outputs, T, alpha)
        loss.backward()
        optimizer_student.step()

train_student(teacher_net, student_net, trainloader, optimizer_student)

# 测试学生模型的准确性
def evaluate(model, testloader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            inputs, labels = data
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return 100 * correct / total

accuracy = evaluate(student_net, testloader)
print(f'Test Accuracy of the student model: {accuracy:.2f}%')

```