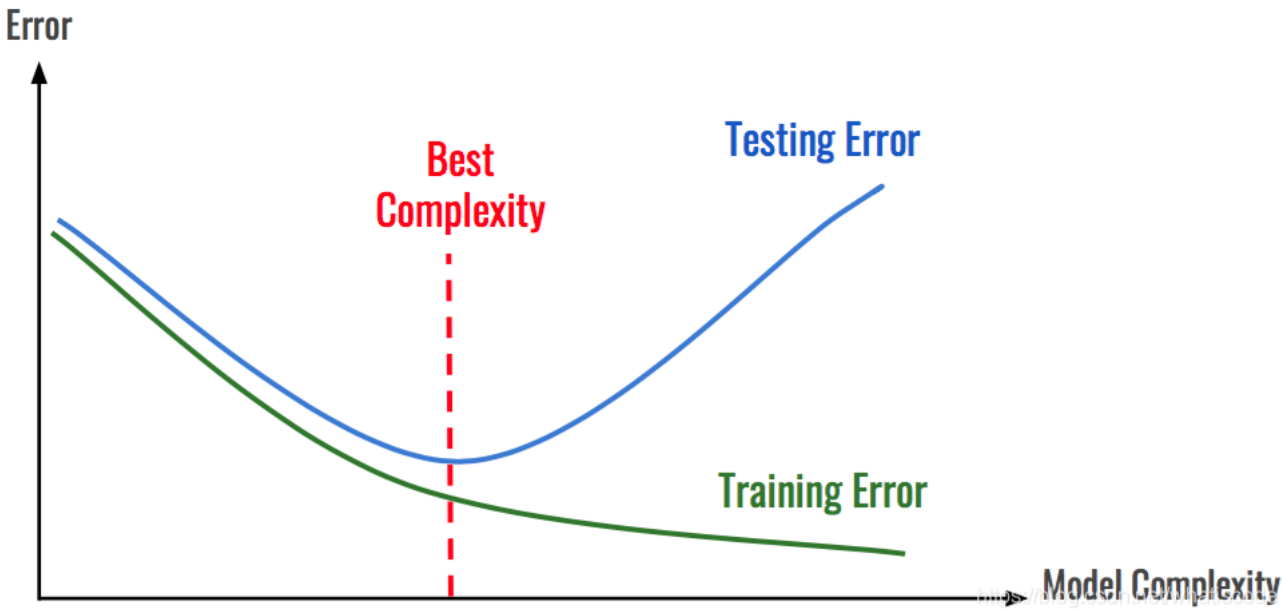
学习目标：理解验证集的作用，学会怎样构建验证集、学会使用训练集和验证集完成模型训练和验证，学会PyTorch下的模型保存与加载以及模型调参

1 构造验证集

在机器学习模型（特别是深度学习模型）的训练过程中，模型是非常容易过拟合的。深度学习模型在不断的训练过程中训练误差会逐渐降低，但测试误差的走势则不一定。

在模型的训练过程中，模型只能利用训练数据来进行训练，模型并不能接触到测试集上的样本。因此模型如果将训练集学的过好，模型就会记住训练样本的细节，导致模型在测试集的泛化效果较差，这种现象称为过拟合（Overfitting）。与过拟合相对应的是欠拟合（Underfitting），即模型在训练集上的拟合效果较差。



如图所示：随着模型复杂度和模型训练轮数的增加，CNN模型在训练集上的误差会降低，但在测试集上的误差会逐渐降低，然后逐渐升高，而我们为了追求的是模型在测试集上的精度越高越好。这种情况就属于过拟合。导致模型过拟合的情况有很多种原因，其中最为常见的情况是模型复杂度（Model Complexity）太高，导致模型学习到了训练数据的方方面面，学习到了一些细枝末节的规律。解决上述问题最好的解决方法：构建一个与测试集尽可能分布一致的样本集（可称为**验证集**），在训练过程中不断验证模型在验证集上的精度，并以此控制模型的训练。

在给定赛题后，赛题方会给定训练集和测试集两部分数据。参赛者需要在训练集上面构建模型，并在测试集上面验证模型的泛化能力。因此参赛者可以通过提交模型对测试集的预测结果，来验证自己模型的泛化能力。同时参赛方也会限制一些提交的次数限制，以此避免参赛选手“刷分”。

在一般情况下，参赛选手也可以自己在本地划分出一个验证集出来，进行本地验证。训练集、验证集和测试集分别有不同的作用：

- 训练集（Train Set）：模型用于训练和调整模型参数；
- 验证集（Validation Set）：用来验证模型精度和调整模型超参数；
- 测试集（Test Set）：验证模型的泛化能力。

因为训练集和验证集是分开的，所以模型在验证集上面的精度在一定程度上可以反映模型的泛化能力。在划分验证集的时候，需要注意验证集的分布应该与测试集尽量保持一致，不然模型在验证集上的精度就失去了指导意义。

构造验证集的方法

既然验证集这么重要，那么如何划分本地验证集呢。在一些比赛中，赛题方会给定验证集；如果赛题方没有给定验证集，那么参赛选手就需要从训练集中拆分一部分得到验证集。验证集的划分有如下几种方式：

(1) 留出法（Hold-Out）

直接将训练集划分成两部分，新的训练集和验证集。这种划分方式的优点是最为直接简单；缺点是只得到了一份验证集，有可能导致模型在验证集上过拟合。留出法应用场景是数据量比较大的情况。

(2) 交叉验证法（Cross Validation，CV）

将训练集划分成K份，将其中的K-1份作为训练集，剩余的1份作为验证集，循环K训练。这种划分方式是所有的训练集都是验证集，最终模型验证精度是K份平均得到。这种方式的优点是验证集精度比较可靠，训练K次可以得到K个有多样性差异的模型；CV验证的缺点是需要训练K次，不适合数据量很大的情况。

(3) 自助采样法（BootStrap）

通过有放回的采样方式得到新的训练集和验证集，每次的训练集和验证集都是有区别的。这种划分方式一般适用于数据量较小的情况。

在本次赛题中已经划分为验证集，因此选手可以直接使用训练集进行训练，并使用验证集进行验证精度（当然你也可以合并训练集和验证集，自行划分验证集）。

当然这些划分方法是从数据划分方式的角度来讲的，在现有的数据比赛中一般采用的划分方法是留出法和交叉验证法。如果数据量比较大，留出法还是比较合适的。当然任何的验证集的划分得到的验证集都是要保证训练集-验证集-测试集的分布是一致的，所以如果不管划分何种的划分方式都是需要注意的。

这里的分布一般指的是与标签相关的统计分布，比如在分类任务中“分布”指的是标签的类别分布，训练集-验证集-测试集的类别分布情况应该大体一致；如果标签是带有时序信息，则验证集和测试集的时间间隔应该保持一致。

2 模型训练与验证

在本节我们目标使用Pytorch来完成CNN的训练和验证过程，CNN网络结构与之前的章节中保持一致。我们需要完成的逻辑结构如下：

- 构造训练集和验证集；
- 每轮进行训练和验证，并根据最优验证集精度保存模型。

```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=10,
    shuffle=True,
    num_workers=10,
)

val_loader = torch.utils.data.DataLoader(
    val_dataset,
    batch_size=10,
    shuffle=False,
    num_workers=10,
)

model = SVHN_Model1()
criterion = nn.CrossEntropyLoss (size_average=False)
optimizer = torch.optim.Adam(model.parameters(), 0.001)
best_loss = 1000.0
for epoch in range(20):
    print('Epoch: ', epoch)

    train(train_loader, model, criterion, optimizer, epoch)
    val_loss = validate(val_loader, model, criterion)

    # 记录下验证集精度
    if val_loss < best_loss:
        best_loss = val_loss
        torch.save(model.state_dict(), './model.pt')
```

其中每个Epoch的训练代码如下：

```
    # 切换模型为训练模式
    model.train()

    for i, (input, target) in enumerate(train_loader):
        c0, c1, c2, c3, c4, c5 = model(data[0])
        loss = criterion(c0, data[1][:, 0]) + \
                criterion(c1, data[1][:, 1]) + \
                criterion(c2, data[1][:, 2]) + \
                criterion(c3, data[1][:, 3]) + \
                criterion(c4, data[1][:, 4]) + \
                criterion(c5, data[1][:, 5])
        loss /= 6
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

其中每个Epoch的验证代码如下：

```
def validate(val_loader, model, criterion):
    # 切换模型为预测模型
    model.eval()
    val_loss = []

    # 不记录模型梯度信息
    with torch.no_grad():
        for i, (input, target) in enumerate(val_loader):
            c0, c1, c2, c3, c4, c5 = model(data[0])
            loss = criterion(c0, data[1][:, 0]) + \
                    criterion(c1, data[1][:, 1]) + \
                    criterion(c2, data[1][:, 2]) + \
                    criterion(c3, data[1][:, 3]) + \
                    criterion(c4, data[1][:, 4]) + \
                    criterion(c5, data[1][:, 5])
            loss /= 6
            val_loss.append(loss.item())
    return np.mean(val_loss)
```

3 模型保存与加载

在Pytorch中模型的保存和加载非常简单，比较常见的做法是保存和加载模型参数：

```
//模型保存
torch.save(model_object.state_dict(), 'model.pt')

//模型加载
model.load_state_dict(torch.load(' model.pt'))
```

4 模型调参流程

深度学习原理少但实践性非常强，基本上很多的模型的验证只能通过训练来完成。同时深度学习有众多的网络结构和超参数，因此需要反复尝试。训练深度学习模型需要GPU的硬件支持，也需要较多的训练时间，如何有效的训练深度学习模型逐渐成为了一门学问。

深度学习有众多的训练技巧，比较推荐的阅读链接有：

- http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html
- http://karpathy.github.io/2019/04/25/recipe/

下面翻译了上面第二个网站中的一部分技巧来分享给大家：

摒弃一些对神经网络错误的理解

(1) 神经网络训练并不是一个十全十美的抽象

据称，训练神经网络很容易上手。许多库和框架都以显示30行奇迹片段来解决您的数据问题为荣，给人以假的印象，即这些东西是即插即用的。

```
>>> your_data = # plug your awesome dataset here
>>> model = SuperCrossValidator(SuperDuper.fit, your_data, ResNet50, SGDOptimizer)
# conquer world here
```

比如说，以上代码给人一种假象，即只通过一行代码就可以完成对神经网络的训练。但事实上神经网络的训练是一件很复杂的事情。如果您坚持使用该技术而不了解其工作原理，则很可能会失败。

(2) 神经网络的训练不知道怎么的就失败了

当你破坏或错误配置代码时，您通常会遇到某种异常。你插入了一个整数，该整数应为字符串。该函数仅需要3个参数。导入失败。该密钥不存在。两个列表中的元素数量不相等。此外，通常可以为某些功能创建单元测试。

这只是训练神经网络的开始。可能在语法上，所有内容都是正确的，但还是会训练失败，而且也很难看出来到底哪里错了。例如，也许您在数据增强过程中左右翻转图像时忘记了翻转标签。您的网络仍然可以工作良好，因为您的网络可以在内部学习检测翻转的图像，然后左右翻转预测。也许由于一个错误，您的自回归模型意外地将它试图预测的东西作为输入。或者，您尝试修剪渐变，但修剪了损失，导致在训练过程中忽略了异常示例。或者，您从预先训练的检查点初始化了权重，但未使用原始均值。或者，您只是搞砸了正则化强度，学习率，其衰减率，模型大小等设置。在大多数情况下，它会训练，但默默地工作会更糟。

结果，**训练神经网络的快速而暴力的方法行不通**，这往往会令人感到难受。事实上，使得神经网络正常工作本身就是一件比较困难的事情，我们可以通过缜密的思考，提高警惕，坚持到底和利用可视化来帮助我们训练神经网络。

训练神经网络的流程

(1) 好好检查数据

训练神经网络的第一步是完全不接触任何神经网络代码，而是从彻底检查数据开始。此步骤至关重要。花时间去检查数据是一件比较重要的工作。因为数据中往往可能存在异常值，而且了解它们的分布可以有利于我们找到一个更好的模型。

(2) 设置端到端的培训/评估框架并得到一个并不完美的baseline

此阶段的提示和技巧：

- **固定随机种子**：始终使用固定的随机种子来确保两次运行代码时您将获得相同的结果。
- **简化**：在此阶段，请务必关闭任何数据扩充功能。数据扩充是我们稍后可能会采用的一种正则化策略，但是目前这只是引入一种错误的尝试。
- **验证损失**：验证您的损失是否从正确的损失值开始。
- **设定一个好的初始化**
- **人类基线**：监控除损失之外的指标，这些指标是人类可以解释和检查的（例如准确性）。尽可能评估自己（人类）的准确性并与之进行比较。
- **可视化预测动态**。我喜欢在培训过程中可视化固定测试批次上的模型预测。这些预测如何运动的“动力”将使您对培训的进行方式有非常好的直觉。如果网络以某种方式过度摆动，可能会感觉网络“努力”以适应您的数据，这表明不稳定。抖动量也很容易注意到非常低或非常高的学习率。

(3) 过度拟合

我想采用的找到一个好的模型的方法有两个阶段：首先获得一个足够大的模型以使其可以过度拟合（即专注于训练损失），然后适当地对其进行正则化（放弃一些训练损失以提高验证损失）。

此阶段的一些提示和技巧：

- **选择模型**：为了减少训练损失，您需要为数据选择合适的体系结构。
- **Adam是安全的**。在设定基准的早期阶段，我喜欢以3e-4的学习率使用Adam。以我的经验，亚当更宽容超参数，包括不良的学习速度。对于ConvNets，调整良好的SGD几乎总是比Adam稍胜一筹，但是最佳学习率区域要狭窄得多且针对特定问题。
- **一次只使一个复杂化**。如果您有多个信号要插入您的分类器，我建议您将它们一个接一个地插入，并每次确保获得预期的性能提升。
- **不要相信学习率衰减的默认值**。如果您要重新使用其他领域的代码，请务必小心学习率。

(4) 正则化

此阶段的一些提示和技巧：

- **获取更多数据**
- **数据扩充**
- **创意增强**：如果半假数据没有做到这一点，伪造数据也可能会有所作为。人们正在寻找扩展数据集的创新方法。例如，领域随机化，模拟的使用，巧妙的混合，例如将（潜在模拟的）数据插入场景，甚至GAN。
- **使用预训练网络**
- **坚持监督学习**
- **减小输入维数**
- **减小模型尺寸**
- **减小批量大小**

- **Dropout**
- **提早停止训练**。根据您测得的验证损失提前停止训练，以在模型快要过拟合的时候捕获模型。
- **尝试更大的模型**。我最后提到这一点，只是在提早停止后才提到，但是过去我发现，大型模型当然最终会过拟合得多，但是它们的“早期停止”性能通常会比小型模型好得多。

(5) 微调

此阶段的一些提示和技巧：

- **随机网格搜索**
- **超参数优化**

(6) 进一步提高精确率

- **模型集成**

5 总结

本章以深度学习模型的训练和验证为基础，讲解了验证集划分方法、模型训练与验证、模型保存和加载以及模型调参流程。

需要注意的是模型复杂度是相对的，并不一定模型越复杂越好。在有限设备和有限时间下，需要选择能够快速迭代训练的模型。