1. 调参策略

调参是指在深度学习中通过调整超参数（如学习率、正则化参数、批大小等）的一些列取值来优化模型性能的过程。不同的超参数组合会导致模型有不同的性能，由于深度学习模型具有高度复杂性和非线性性，在实践中很难通过经验和直觉来选择最佳的超参数取值，借助一些调参技术和调参工具可以帮助我们更好地的选择到使模型性能更好的超参数，但在调参过程中需要考虑超参数空间、计算资源的限制、模型的特点等因素，避免过度调整和过拟合。

## 模型参数和超参数

**模型参数**是指神经网络中的权重和偏置等参数，它们是在模型训练过程中被学习出来的，用于描述模型从输入到输出的映射函数。模型参数的值在训练过程中不断调整，最终达到一个最优值并固定下来，用于进行模型的预测和推断。

**超参数**是指在训练模型之前需要设定的参数，例如学习率、正则化参数、批次大小、网络结构等。这些参数不是由模型自动学习得到的，而是需要手动设置或通过试错等方法进行调整。不同的超参数组合会对模型的性能和训练速度产生很大的影响，因此选择合适的超参数非常重要。

以下是深度学习中常见的模型参数和超参数：

（一）模型参数

**权重（weights）：**神经网络中的每个神经元都与上一层中的所有神经元相连，每个连接都有一个权重。权重是神经网络中最重要的参数之一，它决定了神经元之间信号传递的强弱程度。

**偏置（biases）：**偏置是神经元的一个额外参数，它表示神经元的激活阈值。它相当于神经元的输出增加一个偏移量，用于调整输出结果的范围和大小。

（二）超参数

**学习率（learning rate）：**学习率是用于控制每次迭代中模型参数权重和偏置更新的步长，即每次参数调整的幅度。较小的学习率会让模型收敛过慢，而较大的学习率则能够让模型更快地收敛但更容易出现不稳定的震荡甚至发散。

**批次大小（batch size）：**批次大小表示每次迭代时用来训练模型的样本数量。较小的批次大小能够使模型更快地收敛，但也会增加模型的方差；较大的批次大小则能够减少方差，但会使模型更慢地收敛。

**正则化参数（regularization parameter）：**正则化参数是用来控制模型复杂度的参数，通常指L1 正则化和 L2 正则化中的惩罚系数，也称为权重衰减系数。正则化可以有效地防止模型过拟合，一般正则化参数的值越大，正则化的强度就越大，模型的泛化能力就会越强。

激活函数（activation function）：激活函数是神经元内部的非线性函数，它将神经元的输入转化为输出。激活函数有ReLU、sigmoid、tanh等。

**优化算法（optimization algorithm）：**优化算法用于更新模型的参数，使得模型的loss值尽可能地降低，包括随机梯度下降（SGD）、动量法（momentum）、自适应梯度算法（Adam）等。

**网络结构（network architecture）：**网络结构是指神经网络中神经元和层之间的连接方式和拓扑结构，包括全连接层、卷积层、池化层等。

### 手动调参和自动调参

调参分为手动调参和自动调参，手动调参是指通过经验和观察来手动调整超参数的取值，以优化深度学习模型的性能。手动调参通常需要具备一定的领域知识和经验，以及对深度学习模型的理解和掌握。

手动调参的过程可以通过以下几个步骤实现：

1. 确定调整的超参数：首先需要确定要调整的超参数，包括学习率、正则化系数、批量大小、层数等。

2. 设定超参数的初始值：根据经验和先前的工作，设置超参数的初值，然后训练模型并评估性能，根据评估结果调整超参数的值。

3. 不断迭代调整：不断地迭代调整超参数的值，直到获得满意的模型性能为止。在调整的过程中，需要注意避免过拟合和欠拟合等问题。

手动调参虽然需要一定的经验和时间，但它能够有效地优化模型性能，特别是在数据集较小或者模型比较简单的情况下。

自动调参是指利用计算机算法和机器学习技术，自动地探索超参数空间，以寻找最优的超参数组合，从而优化深度学习模型的性能。

常用的自动调参算法包括网格搜索、随机搜索、贝叶斯优化等。这些算法通过不断地在超参数空间中采样，以获取更多的性能信息，并根据性能信息调整超参数的取值。这种自动化的调参方法可以大大减少人工干预的时间和成本，并能够获得更好的模型性能。

自动调参的过程通常包括以下几个步骤：

1. 确定超参数空间：需要确定超参数的范围和取值。不同的超参数空间会对调参结果产生不一样的影响。

2. 选择调参算法：根据实际情况选择合适的调参算法，例如网格搜索、随机搜索、贝叶斯优化等。

3. 设定评估指标：根据任务需求和实际情况，设定评估指标，例如准确率、召回率、F1值等。

4. 自动搜索：根据选择的调参算法，在超参数空间中进行自动搜索，获取不同超参数组合的性能评估结果。

5. 选择最优参数：根据评估指标和超参数组合的性能评估结果，选择最优的超参数组合作为模型的最终参数。

自动调参可以大大提高深度学习模型的性能，并可以节省调参的时间和成本。但需要注意，自动调参仍然需要对模型和任务有一定的理解和经验，才能获得更好的结果。在实践中，通常需要手动调参和自动调参结合使用，以获得更好的性能和更高的效率。

### 调参算法

1. 网格搜索（Grid Search）

网格搜索算法是一种常见的超参数优化方法，通过在超参数空间中进行穷举搜索，来寻找最优的超参数组合。网格搜索算法的基本思想是将超参数空间划分为一个个网格，然后对每个网格中的超参数组合进行训练和评估，最终选择最优的超参数组合。

网格搜索算法的步骤如下：

（1）确定要优化的超参数和超参数的取值范围。

（2）根据超参数的取值范围，将超参数空间划分为多个网格。

（3）对于每个网格中的超参数组合，训练模型并在验证集上评估模型性能。

（4）根据模型性能选择最优的超参数组合。

（5）在测试集上测试最优超参数组合下的模型性能。

网格搜索算法的优点是易于实现，对于超参数空间较小的问题效果较好。但是，对于超参数空间较大的问题，网格搜索算法的计算量会非常大，效率较低。所以使用网格搜索时，一般会先使用较广的搜索范围和较大的步长，来寻找全局最优值可能的位置；然后逐渐缩小搜索范围和步长，来寻找更精确的最优值。这种操作方案可以降低所需的时间和计算量，但由于目标函数一般是非凸的，所以很可能会错过全局最优值。此外，网格搜索算法无法利用超参数之间的相关性进行搜索，容易出现过拟合等问题。因此，在实际应用中，网格搜索算法通常会和其他超参数优化算法结合使用，例如随机搜索、贝叶斯优化等。

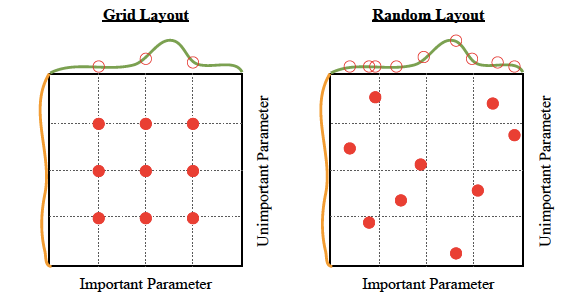


图 11‑1 网格搜索与随机搜索示意图

2. 随机搜索（Random Search）

随机搜索算法与网格搜索算法相似，只是不再测试参数空间中的的所有值，随机搜索算法通过在超参数空间中随机采样来寻找最优的超参数组合，从而避免了网格搜索算法的计算量大和效率低等问题。相比于网格搜索算法，随机搜索算法更加灵活，可以在更广泛的超参数空间内寻找最优超参数组合。

随机搜索算法的步骤如下：

（1）确定要优化的超参数和超参数的取值范围。

（2）根据超参数的取值范围，在超参数空间中进行随机采样。

（3）对于每个随机采样得到的超参数组合，训练模型并在验证集上评估模型性能。

（4）根据模型性能选择最优的超参数组合。

（5）在测试集上测试最优超参数组合下的模型性能。

随机搜索算法的优点也是易于实现，对于超参数空间较大的问题效果较好。但是，随机搜索算法也有一些缺点，例如可能会在超参数空间的某些区域采样过多或过少，导致模型性能的损失。此外，随机搜索算法也无法利用超参数之间的相关性进行搜索，可能会漏掉一些重要的超参数组合。因此，在实际应用中，随机搜索算法通常也会和其他超参数优化算法结合使用，例如网格搜索、贝叶斯优化等。

3. 贝叶斯优化（Bayesian Optimization）

贝叶斯优化是一种基于贝叶斯推断的超参数优化方法，它通过在超参数空间中建立代理模型来推断最优超参数组合的位置，从而在尽可能少的训练次数内找到最优的超参数组合。相对于网格搜索和随机搜索等方法，贝叶斯优化具有更高的搜索效率和更好的性能。

贝叶斯优化的核心思想是，在已有的超参数组合和模型性能的数据集上，通过先验分布和后验分布的贝叶斯公式计算下一次尝试的超参数组合的概率，并根据概率进行采样。这样，在尝试少量超参数组合的情况下，就能够不断地优化代理模型，从而找到最优的超参数组合。

贝叶斯优化的基本流程如下：

（1）定义代理模型，例如高斯过程（Gaussian Process）模型、随机森林（Random Forest）模型等。

（2）建立先验分布，例如高斯分布等。

（3）通过先验分布和已有的超参数组合及模型性能的数据集，计算超参数组合的后验分布。

（4）根据后验分布进行采样，得到下一次要尝试的超参数组合。

（5）训练模型并在验证集上评估模型性能，将新的超参数组合及模型性能加入数据集中。

（6）重复步骤（3）至（5），直到达到指定的训练次数或找到最优的超参数组合。

贝叶斯优化的优点是能够充分利用已有的超参数组合和模型性能的数据，减少搜索次数和训练时间，同时具有高效、鲁棒性好等特点。但是，贝叶斯优化也有一些缺点，例如建立代理模型和计算后验分布需要较高的计算复杂度，而且需要足够多的训练次数才能找到最优的超参数组合。因此，在实际应用中，贝叶斯优化通常会和其他超参数优化算法结合使用，例如网格搜索、随机搜索等。

4. Tree-structured Parzen estimators (TPE)

Tree-structured Parzen estimators (TPE)是一种贝叶斯优化的变种方法，它通过对参数空间进行二叉划分来建立一个树形结构，并利用样本的分类信息来优化树形结构中各节点的分割点，从而在较少的训练次数内快速找到最优的超参数组合。

TPE 的基本思路是将参数空间分成两个部分：一部分是符合某种分布的“优质”区域，另一部分是符合其他分布的“普通”区域。然后，通过不断地优化两种分布的超参数，逐渐缩小“优质”区域的范围，最终找到最优的超参数组合。

具体来说，TPE的流程如下：

（1）定义两个分布和，其中表示“优质”区域的分布，表示“普通”区域的分布。

（2）利用已有的样本数据，分别拟合和的超参数。

（3）利用和来生成新的超参数，其中，生成的超参数在“优质”区域采样，生成的超参数在“普通”区域采样。

（4）对生成的新的超参数进行训练和评估，并将其添加到样本数据中。

（5）根据评估结果，更新和的超参数。

（6）重复步骤（3）至（5），直到达到指定的训练次数或找到最优的超参数组合。

相对于传统的贝叶斯优化方法，TPE 的优点在于其树形结构的特殊形式，能够更快地找到最优的超参数组合。而且，TPE 的实现比较简单，可以通过改变分布的形式和超参数的选择来适应不同的问题。不过，TPE 也有一些缺点，例如可能会出现过度拟合的情况，需要在实践中加以注意。

### 调参工具

1. Scikit-learn

Scikit-learn是一个流行的Python机器学习库，它具有网格搜索和随机搜索的实现，如果您使用sklearn构建模型，它是一个很好的起点。

使用Scikit-learn进行网格搜索，使用机器学习算法SVM对鸢尾花数据集分类，在parameters变量中设置好超参数的搜索空间，GridSearchCV类来执行网格搜索，它会自动执行交叉验证，并返回最佳参数组合的模型。

1. from sklearn import svm, datasets
2. from sklearn.model\_selection import GridSearchCV
3. iris = datasets.load\_iris()
4. parameters = {'kernel':('linear','poly','rbf'), 'C':[1, 10, 100],'gamma': [0.001, 0.01, 0.1, 1, 10]}
5. svc = svm.SVC()
6. clf = GridSearchCV(svc, parameters,cv=3,iid=False)
7. search = clf.fit(iris.data, iris.target)
8. print(search.best\_params\_)
9. print("Best parameters: {}".format(search.best\_params\_))

使用逻辑回归对鸢尾花数据集分类，与网格搜索类似，定义好超参数空间后，RandomizedSearchCV类进行随机搜索以探索最佳参数。

1. from sklearn.datasets import load\_iris
2. from sklearn.linear\_model import LogisticRegression
3. from sklearn.model\_selection import RandomizedSearchCV
4. from scipy.stats import uniform
5. iris = load\_iris()
6. logistic = LogisticRegression(solver='saga', tol=1e-2, multi\_class='auto',max\_iter=200, random\_state=0)
7. distributions = dict(C=uniform(loc=0, scale=4), penalty=['l2', 'l1'])
8. clf = RandomizedSearchCV(logistic, distributions, cv=3, random\_state=0)
9. search = clf.fit(iris.data, iris.target)
10. print(search.best\_params\_)
11. Optuna

OptunaOptuna是一个用于自动超参数优化的Python库，使用路径详细信息的历史记录来确定有前途的区域来搜索优化超参数，从而在最短的时间内找到最佳超参数。 它具有修剪功能，可以在训练的早期阶段自动停止没有希望的路径。Optuna提供的一些关键特性是：

（1）轻量级、多功能且与平台无关的架构

（2）Pythonic 搜索空间

（3）高效的优化算法

（4）易于并行化

（5）快速可视化

Optuna工具的使用方法：

（1）使用如下命令安装Optuna

1. pip install optuna

（2）定义优化目标函数：您需要定义一个评估超参数配置的目标函数objective(trial)。这个函数的输入是超参数的字典，输出是一个评估指标（例如模型的准确率或损失函数值）。

（3）定义超参数空间：您需要定义每个超参数的搜索范围。Optuna支持离散、连续和条件参数类型。

（4）创建一个研究对象并优化目标函数。

（5）输出调优结果。

Optuna工具对Pytorch框架的调优模板：

1. import torch
2. import optuna
3. # 定义优化目标函数
4. def objective(trial):
5. # trial对象定义超参数的搜索范围。
6. n\_layers = trial.suggest\_int('n\_layers', 1, 3)
7. layers = []
8. in\_features = 28 \* 28
9. for i in range(n\_layers):
10. out\_features = trial.suggest\_int(f'n\_units\_l{i}', 4, 128)
11. layers.append(torch.nn.Linear(in\_features, out\_features))
12. layers.append(torch.nn.ReLU())
13. in\_features = out\_features
14. layers.append(torch.nn.Linear(in\_features, 10))
15. layers.append(torch.nn.LogSoftmax(dim=1))
16. model = torch.nn.Sequential(\*layers).to(torch.device('cpu'))
17. ...
18. # 输出模型准确率作为评估指标
19. return accuracy
20. # 创建一个研究对象并优化目标函数。
21. study = optuna.create\_study(direction='maximize')
22. study.optimize(objective, n\_trials=100)
23. # 打印最佳准确率及超参数组合
24. print('Accuracy: {}'.format(study.best\_trial.value))
25. for key, value in trial.params.items():
26. print(" {}: {}".format(key, value))
27. Ray

Ray是一个开源的分布式计算框架，它提供了很多用于调参的工具和功能，其中Ray Tune是Ray的一个子模块，专门用于超参数优化和调参。它提供了一系列的算法和搜索空间定义方式，包括网格搜索、随机搜索、超参数优化算法（如Hyperopt、Bayesian Optimization、Optuna等）。

除此之外，Ray Tune支持多种框架，如 Sklearn、XGBoost、TensorFlow、PyTorch 等。Ray的强大功能可以帮助你更轻松地进行参数调优和模型训练。与Optuna工具调参十分相似，你可以在Ray的官方文档中找到更详细的说明和示例代码来了解如何使用这些工具。