超参调优 - 用户手册

类脑云超参调优能在用户给定的超参数搜索空间中，通过超参数搜索策略，帮助用户在众多的超参数组合中，找出模型性能较优的超参数。

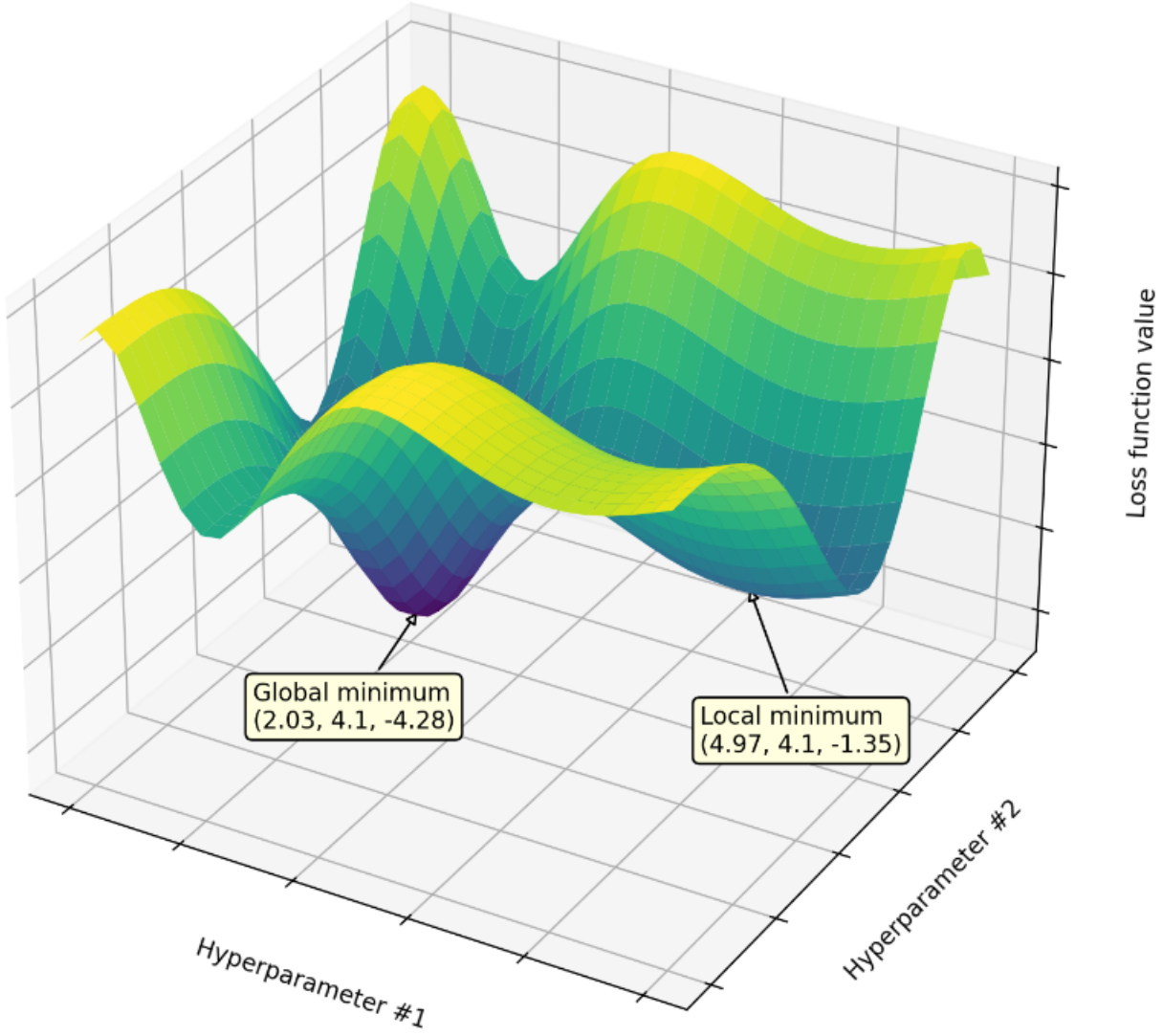
目录

1. 什么是机器学习中的超参调优（超参搜索）

在机器学习中，参数分为**模型参数 （Model Parameters）和超参数（Hyperparameters）**，他们的区别在于：

- **模型参数**在训练反向传播过程中会被自动更新，是**学习到的参数**。比如：神经网络模型的 weights 和 bias，决策树模型的特征切分点等。
- **超参数**是在训练启动时**人为设置的固定的参数**。比如神经网络模型的层数、节点数、优化器、学习率、冲量、batchsize，决策树模型的学习率、最大深度等。

训练机器学习模型，或者其他最优化问题的求解，都需要反复调整超参数，进行多次模型训练，从而找到模型效果最好的一组超参数。尤其是有些模型对超参数比较敏感，较小的超参数改动都可能导致模型效果的明显变化，因此反复试验找到一组最佳的超参数就比较重要。超参数搜索空间与模型性能关系示例如下图：



传统的方式是，人为设置多组参数，一遍遍地运行，记录每次训练结果，从而找到最佳的超参数组合。缺点：

1. 超参数较多、每个参数的取值范围较大时，手动修改参数反复试验会比较繁琐；
2. 难以评估下一次较优的参数应该设置为多少。老人看经验，新人看运气。

因此，诞生了一些参数搜索的算法和框架，帮助开发者自动寻找最优的超参数组合。该类问题一般称之为超参调优（Hyperparameter Tuning or Hyperparameter Optimization）或超参搜索（Hyperparameter Search）。

注意，一般不进行超参数搜索的场景有：

- 模型对超参数不敏感，不同超参数训练结果接近。
- 计算代价特别大的模型，无法接受多次调参。

2. 如何工作的

2.1. 如何设置超参数

在命令行填写 `${hpo.optimizer}` 格式字符时，并在页面填写超参数搜索范围。提交超参调优作业后，平台将解析到超参数 `optimizer`，并将 `${hpo.optimizer}` 字符串替换为具体的参数值。

比如，填写启动命令如下

```
python train.py --epochs=1 --optimizer=${hpo.optimizer} --lr=${hpo.lr}
```

在页面设置超参数 `optimizer` 可选值为 `Adam` 和 `SGD`，设置超参数 `lr` 取值范围是 0.1-1.0 的浮点数，提交超参调优作业后，平台给出一组超参数，并将命令替换为

```
python train.py --epochs=1 --optimizer=SGD --lr=0.3
```

另外，环境变量的值，也支持设置 `${hpo.xxx}` 格式的超参数。

2.2. 如何输出模型指标

当前只支持从标准输出采集模型指标，可将指标直接输出到 stdout（标准输出），例如

```
for imgs, targets in dataloader:
    ...
    loss = mse_fn(outputs, targets)
    print(f"loss={loss}")
```

输出的格式必须是 指标名=数值，完整的正则表达式匹配规则是 `([\w|-]+)\s*=\s*([+-]?\d*(\\d+)?([Ee][+-]?\d+)?)`

2.3. 代码示例

代码规范只有2条：

1. 代码能读取超参数：从命令行参数中解析，或从环境变量中读取。
2. 按以上格式要求输出指标值。

```
from __future__ import print_function

import argparse

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
from torchvision import datasets, transforms


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
```

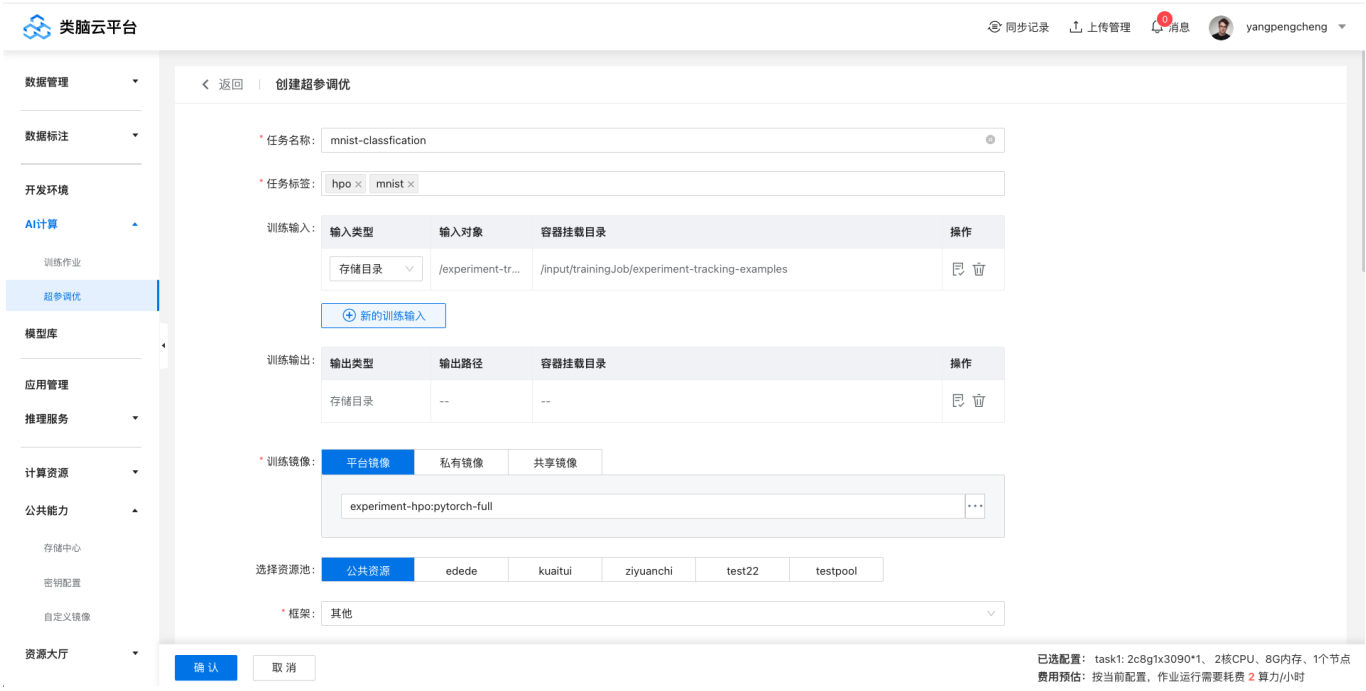
```
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            # loss=xxx 格式
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tloss={:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break


def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target,
reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of
the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()
```

```
    test_loss /= len(test_loader.dataset)

    # loss=xxx accuracy=xxx 格式
    print('\nTest set: Average loss={:.4f}, accuracy={:.4f}
({}/{})\n'.format(
        test_loss, correct / len(test_loader.dataset),
        correct, len(test_loader.dataset)))


def main():
    # Training settings
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
    parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                        help='input batch size for training (default:
64)')
    parser.add_argument('--test-batch-size', type=int, default=1000,
metavar='N',
                        help='input batch size for testing (default:
1000)')
    parser.add_argument('--epochs', type=int, default=14, metavar='N',
                        help='number of epochs to train (default: 14)')
    parser.add_argument('--optimizer', type=str, default='Adadelta',
metavar='O',
                        help='optimizer name in torch.optim (default:
Adadelta)')
    parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                        help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                        help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--no-mps', action='store_true', default=False,
                        help='disables macOS GPU training')
    parser.add_argument('--dry-run', action='store_true', default=False,
                        help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10,
metavar='N',
                        help='how many batches to wait before logging
training status')
    parser.add_argument('--save-model', action='store_true',
default=False,
                        help='For Saving the current Model')
    args = parser.parse_args()
    print(args)

    use_cuda = not args.no_cuda and torch.cuda.is_available()
    use_mps = not args.no_mps and torch.backends.mps.is_available()

    torch.manual_seed(args.seed)

    if use_cuda:
        device = torch.device("cuda")
```

```
    elif use_mps:
        device = torch.device("mps")
    else:
        device = torch.device("cpu")

    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])
    dataset1 = datasets.MNIST('../data', train=True, download=True,
                              transform=transform)
    dataset2 = datasets.MNIST('../data', train=False,
                              transform=transform)
    train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    model = Net().to(device)
    optimizer = getattr(optim, args.optimizer)
    optimizer = optimizer(model.parameters(), lr=args.lr)

    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    if args.save_model:
        torch.save(model.state_dict(), "mnist_cnn.pt")


if __name__ == '__main__':
    main()
```

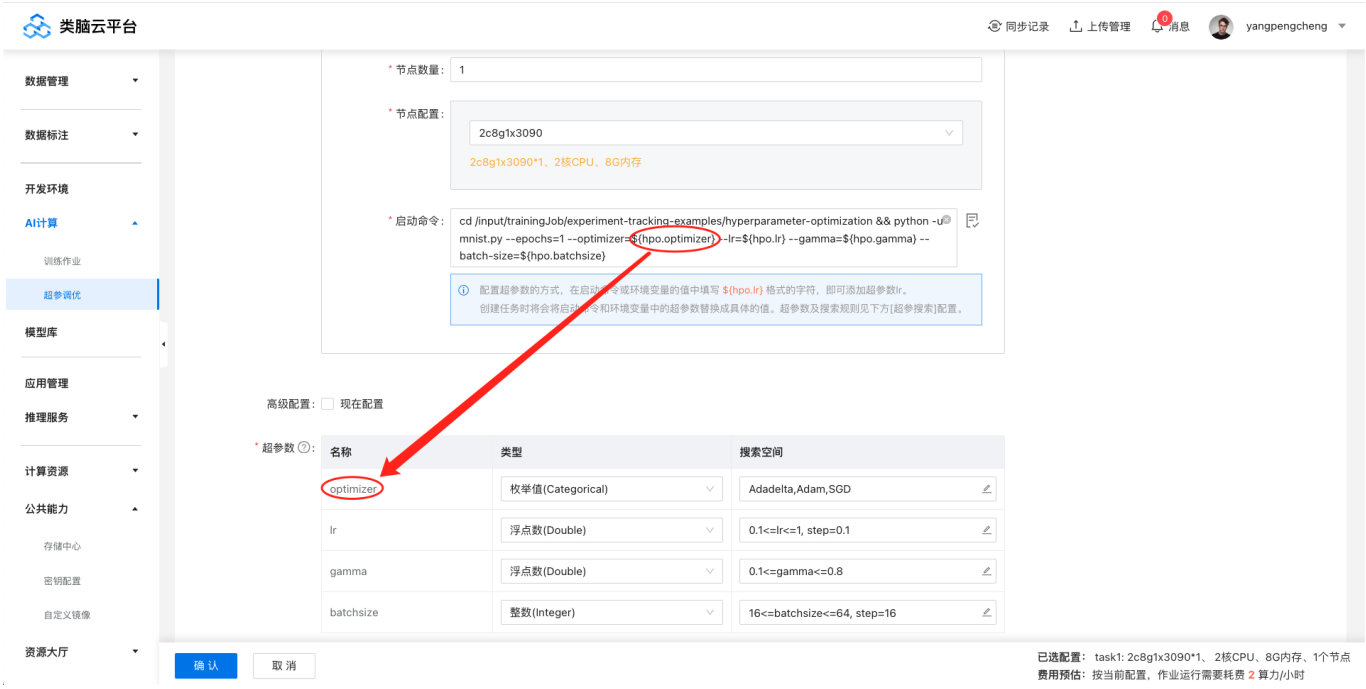
3. 创建超参调优作业

填写超参调优的基本信息，与训练作业的填写方式一致。



填写启动命令，并在启动命令中设置需要搜索的超参数。

1. 在启动命令或环境变量的值中填写 ${hpo.lr} 格式的字符，即可添加超参数lr。
2. 创建任务时将会将启动命令和环境变量中的超参数替换成具体的值。超参数及搜索规则见下方“超参搜索”配置。



其他超参调优配置

- 优化指标：评估模型质量的指标名称，比如 accuracy、loss、mAP 等，需要将指标及其值输出到日志或其他目的地，见“输出指标”配置。
- 优化方向：最大化或最小化优化指标，比如最小化 loss，最大化 accuracy。
- 目标值：当某次参数搜索，优化指标达到此目标值时，提前结束超参搜索。
- 计算方式：
    - 最新输出的指标数值：取最后一次输出的指标及其值作为本次超参搜索模型的指标；

    - 过程中最优的指标数值：优化方向为最小化时，取优化指标历史值的最小值为最优值，否则取优化指标历史值的最大值为最优值。
- 输出指标到：当前仅支持输出到标准输出，例如代码里执行 `print(f"accuracy=0.862")`，平台将采集到当前优化指标 accuracy 为 0.862.
- 搜索算法：当前支持以下4种算法
    - 随机搜索：纯随机生成超参组合的方法，与网络搜索原理相似，将超参数搜索空间分成网格，但随机搜索为每个trial会随机选择一组超参数。对于非线性、高维且计算代价较大的问题，它作为一个高效的初筛方法，缩小网格搜索的范围。
    - 网格搜索：网格搜索算法，将所搜空间均匀分成网格，然后遍历所有可能的组合来确定最佳组合。当搜索空间较小时，可以使用该方法，找到绝对最优组合。
    - 贝叶斯优化：使用一个概率模型（高斯过程）来估计目标函数，适用于在参数空间很大或目标函数很复杂时。
    - TPE：Tree-structured Parzen Estimator，是一个无需附加依赖的轻量级算法，可以支持所有的搜索空间类型，为HPO中使用的默认算法。 它可以处理复杂、非线性、高纬度且计算代价较大的问题。TPE的缺点是无法发现不同参数之间的联系。参考文献：Algorithms for Hyper-Parameter Optimization
- 最大搜索次数：搜索参数的次数，即运行训练作业的次数，达到最大次数时，即停止本次超参搜索。
- 最大并发数：能同时运行的训练作业数量。