ModelArts

模型开发用户指南

文档版本 01

发布日期 2022-05-25

华为技术有限公司

版权所有 © 华为技术有限公司 2022。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

 版权所有 © 华为技术有限公司

 版权所有 © 华为技术有限公司

1 模型开发简介

ModelArts提供了模型训练的功能，方便您查看训练情况并不断调整您的模型参数。您还可以基于不同的数据，选择不同规格的资源池用于模型训练。除支持用户自己开发的模型外，ModelArts还提供了从AI Gallery订阅算法，您可以不关注模型开发，直接使用AI Gallery的算法，通过算法参数的调整，得到一个满意的模型。

请参考以下指导在ModelArts上训练模型：

- 您可以将训练数据导入至数据管理模块进行数据标注或者数据预处理，也支持将已标注的数据上传至OBS服务使用，请参考**2 准备数据**。
- 训练模型的算法实现与指导请参考**3 准备算法**章节。
- 使用控制台创建训练作业请参考**4.1 创建训练作业**章节。使用订阅算法创建训练作业示例请参考**使用AI Gallery订阅的算法构建模型**。使用自定义算法构建模型示例请参考**使用自定义算法在ModelArts上构建模型**。
- 关于训练作业日志、训练资源占用等详情请参考**4.2 查看作业详情**。
- 停止或删除模型训练作业，请参考**4.3 停止、重建或查找作业**。
- 模型超参自动调优指南，请参考**8 自动模型优化（AutoSearch）**。
- 如果您在训练过程中遇到问题，文档中提供了部分故障案例供参考，请参考**训练故障排查**。

说明

当前ModelArts同时存在新版训练和旧版训练。新版训练在旧版训练的基础上进行了功能增强、调度优化、API整改等设计，推荐用户使用新版训练进行模型训练。新旧训练的差异请参考**新旧版本训练差异**。

本文档主要介绍新版训练流程。旧版训练文档，请参考**训练管理（即将下线）**。

版权所有 © 华为技术有限公司

2 准备数据

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。

- **OBS简介**
- **使用训练数据的两种方式**

OBS 简介

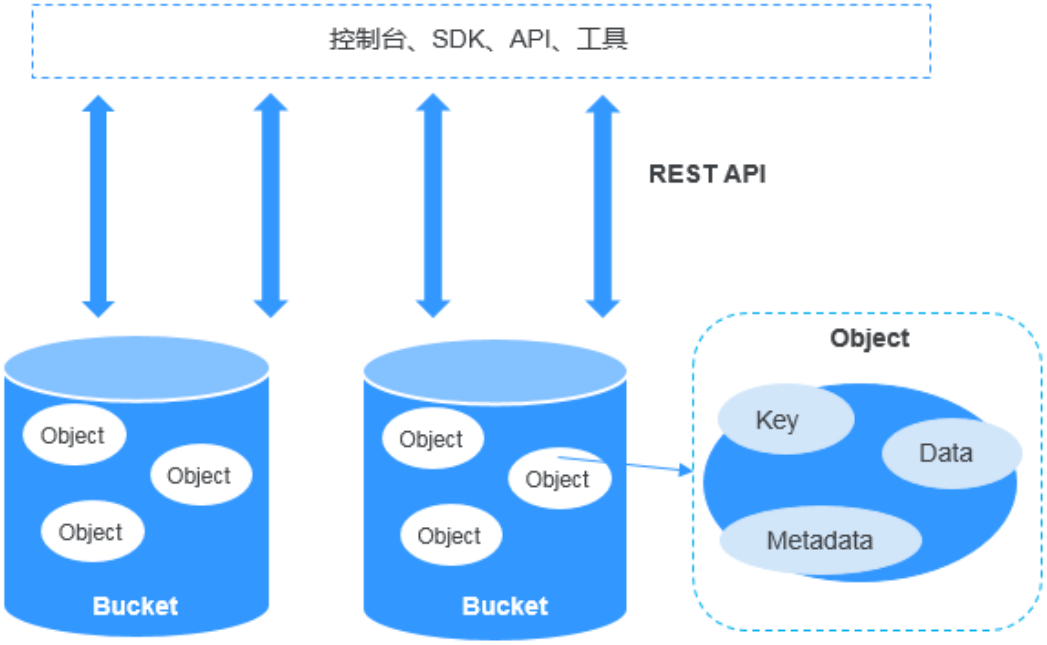
对象存储服务OBS是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力。对象存储服务OBS的基本组成是桶和对象。桶是OBS中存储对象的容器，每个桶都有自己的存储类别、访问权限、所属区域等属性，用户在互联网上通过桶的访问域名来定位桶。对象是OBS中数据存储的基本单位。关于OBS更详细的介绍请参考**《OBS用户指南》**。

对ModelArts来说，obs服务是一个数据存储中心。AI 开发过程中的输入数据、输出数据、中间缓存数据都可以在obs桶中进行存储、读取。

因此，在使用ModelArts之前您需要**创建一个OBS桶**，然后在OBS桶中创建文件夹用于存放数据。

版权所有 © 华为技术有限公司

图 2-1 对象存储服务 OBS



使用训练数据的两种方式

ModelArts模型训练支持2种读取训练数据的方式：

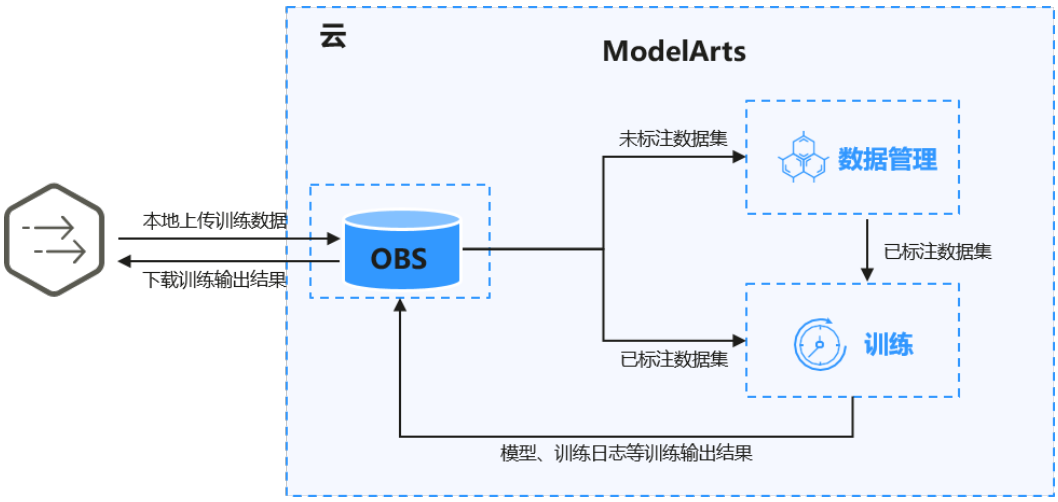
- 使用OBS桶中存储的数据集

  如果您的数据集已完成数据标注和数据预处理，可以将数据上传至OBS桶。当创建训练作业时，在训练输入参数位置填写训练数据所在的OBS桶路径即可完成训练配置。

- 使用数据管理中的数据集

  如果您的数据集未标注或者需要进一步的数据预处理，可以将数据导入ModelArts数据管理模块进行数据预处理。创建数据集详细指导参考**创建数据集**。

图 2-2 准备数据



版权所有 © 华为技术有限公司

3 准备算法

3.1 准备算法简介

机器学习从有限的观测数据中学习一般性的规律，并利用这些规律对未知的数据进行预测。为了获取更准确的预测结果，用户需要选择一个合适的算法来训练模型。针对不同的场景，ModelArts提供大量的的算法样例。以下章节提供了关于业务场景、算法学习方式、算法实现方式的指导。

选择算法的实现方式

ModelArts提供如下方式实现模型训练。

- 使用订阅算法

  ModelArts的AI Gallery，发布了较多官方算法，同时管理了其他开发者分享的算法，不需要进行代码开发，即可使用现成的算法进行模型构建。订阅操作请参考**3.2 使用订阅算法**。

- 在支持的预置框架中使用自定义脚本

  如果您需要使用自己开发的算法，可以选择使用ModelArts预置训练引擎。在ModelArts中，这种方式称为“使用自定义脚本”。ModelArts支持了大多数主流的AI引擎，详细请参见**预置训练引擎**。这些预置引擎预加载了一些额外的python包，例如numpy等；也支持您通过在代码目录中使用“reqiurements.txt”文件安装依赖包。使用自定义脚本创建训练作业请参考**3.3 使用自定义脚本**指导。

- 使用自定义镜像（新版训练请参考**使用自定义镜像训练模型**）

  订阅算法和预置框架涵盖了大部分的训练场景。针对特殊场景，ModelArts支持用户构建自定义镜像用于模型训练。自定义镜像需上传至容器镜像服务（SWR），才能用于ModelArts上训练。由于自定义镜像的制作要求用户对容器相关知识有比较深刻的了解，除非订阅算法和预置引擎无法满足需求，否则不推荐使用。

版权所有 © 华为技术有限公司

选择算法的学习方式

ModelArts支持用户根据实际需求进行不同方式的模型训练。

- 离线学习

  离线学习是训练中最基本的方式。离线学习需要一次性提供训练所需的所有数据，在训练完成后，目标函数的优化就停止了。使用离线学习的优势是模型稳定性高，便于做模型的验证与评估。缺点是时间和空间成本效率低。

- 增量学习

  增量学习是一个连续不断的学习过程。相较于离线学习，增量学习不需要一次性存储所有的训练数据，缓解了存储资源有限的问题；另一方面，增量学习节约了重新训练中需要消耗大量算力、时间以及经济成本。

3.2 使用订阅算法

ModelArts的AI Gallery，发布了较多官方算法，可以帮助AI开发者快速开始训练和部署模型。对于不熟悉ModelArts的用户，可以快速订阅官方推荐算法实现模型训练全流程。

AI Gallery不仅可以订阅官方发布算法，也支持用户发布自定义算法和订阅其他开发者分享的算法。为了使用他人或者ModelArts官方分享的算法，您需要将AI Gallery的算法订阅至您的ModelArts中。

- **查找算法**
- **订阅算法**

查找算法

为了获得匹配您业务的算法，您可以通过多个入口区查找算法。

- 在ModelArts控制台，“算法管理>我的订阅”中，单击“订阅更多算法”，可跳转至“AI Gallery”页面，查找相应的算法。
- 在ModelArts控制台，直接在左侧菜单栏中选择“AI Gallery”，进入“AI Gallery”页面，在“资产集市 > 算法”页面查找相应的算法。

  在AI Gallery的算法模块，根据算法的业务场景，官方发布的算法如**表3-1**所示。

表 3-1 官方发布算法

应用场景	算法	AI引擎	支持服务类型	训练使用的资源类型	部署使用的资源类型
图像分类	**ResNet_v1_50**	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	ResNet_v1_101	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	ResNet_v1_50	TensorFlow	在线服务	Ascend 910	CPU、GPU、Ascend 310
	ResNet50	MindSpore	在线服务	Ascend 910	Ascend 310

版权所有 © 华为技术有限公司

应用场景	算法	AI引擎	支持服务类型	训练使用的资源类型	部署使用的资源类型
	ResNet_v2_50	TensorFlow	在线服务	Ascend 910	Ascend 310
	ResNet_v2_101	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	MobileNet_v1	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	MobileNet_v2	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	Inception_v3	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	ResNet50_AdaInfer	TensorFlow	在线服务	GPU	CPU、GPU
	Res2Net_50	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	Inception_ResNet_v2	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	EfficientNetB0/B4/B7/B8	PyTorch	在线服务	GPU	CPU、GPU
物体检测	RetinaNet_ResNet50	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	YOLOv3_ResNet18	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	YOLOv3_ResNet18	TensorFlow	在线服务	Ascend 910	Ascend 310
	YOLOv3_Darknet53	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	SSD_VGG	Caffe	在线服务	GPU	暂不支持
	EfficientDet	Tensorflow	在线服务	GPU	CPU、GPU
	YOLOv5	Pytorch	在线服务	GPU	CPU、GPU
文本分类	BERT	TensorFlow	在线服务	GPU	CPU、GPU、Ascend 310
	NEZHA	TensorFlow	在线服务	GPU	GPU

版权所有 © 华为技术有限公司

应用场景	算法	AI引擎	支持服务类型	训练使用的资源类型	部署使用的资源类型
图像语义分割	DeepLabV3	TensorFlow	在线服务	GPU	Ascend 310
	DeepLabV3	TensorFlow	在线服务	Ascend 910	Ascend 310
声音分类	声音分类	TensorFlow	在线服务	GPU	CPU、GPU
时序预测	时序预测	PyTorch	在线服务	GPU	暂不支持
强化学习	强化学习 GameAI	TensorFlow	在线服务	GPU	GPU

订阅算法

1. 进入“AI Gallery”，选择“资产集市 > 算法”页签，查找您需要的算法并单击算法名称，进入算法详情页，单击算法详情页右侧的“订阅”，订阅您所需的算法。

   订阅后的算法，状态变为“已订阅”，并且将自动展现在“算法管理 > 我的订阅”页面中。

2. 单击“前往控制台”，选择云服务区域，进入“算法管理 > 我的订阅”页面进入此页面内，单击“产品名称”左侧的小三角，展开算法详情，在“版本列表”区域，单击“创建训练作业”即可进行后续操作。

版权所有 © 华为技术有限公司

图 3-1 订阅算法

3.3 使用自定义脚本

3.3.1 使用自定义脚本简介

如果订阅算法不能满足需求或者用户希望迁移本地算法至ModelArts上训练，可以考虑使用ModelArts支持的预置训练引擎实现算法构建。这种方式在创建算法时被称为“使用自定义脚本”模式。

以下章节介绍了如何使用自定义脚本创建算法。

- 如果需要了解ModelArts模型训练支持的预置引擎和模型，请参考**表3-2**。
- 本地开发的算法迁移至ModelArts需要做代码适配，如何适配请参考**3.3.2 开发自定义脚本**章节。
- 通过ModelArts控制台界面使用自定义脚本创建算法可以参考**3.3.3 创建算法**章节。
- 完成算法开发后，您可以将个人开发算法免费分享给他人使用，请参考**发布免费算法**章节。

预置的训练引擎

当前ModelArts支持的训练引擎及对应版本如下所示。

版权所有 © 华为技术有限公司

表 3-2 新版训练作业支持的 AI 引擎

工作环境	适配芯片	系统架构	系统版本	AI引擎与版本	支持的cuda或Ascend版本
TensorFlow	CPU/GPU	x86_64	Ubuntu18.04	tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64	cuda10.1
PyTorch	CPU/GPU	x86_64	Ubuntu18.04	pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64	cuda10.2
Ascend-Powered-Engine	Ascend910	aarch64	Euler2.8	mindspore_1.3.0-cann_5.0.2-py_3.7-euler_2.8.3-aarch64	cann_5.0.2
				mindspore_1.5.1-cann_5.0.3-py_3.7-euler_2.8.3-aarch64	cann_5.0.3
				tensorflow_1.15-cann_5.0.2-py_3.7-euler_2.8.3-aarch64	cann_5.0.2
				tensorflow_1.15-cann_5.0.3-py_3.7-euler_2.8.3-aarch64	cann_5.0.3
MPI	CPU/GPU	x86_64	Ubuntu18.04	mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64	cuda_10.1
Horovod	GPU	x86_64	ubuntu_18.04	horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64	cuda_10.1
				horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64	cuda_10.2

版权所有 © 华为技术有限公司

说明

- MoXing是ModelArts团队自研的分布式训练加速框架，它构建于开源的深度学习引擎TensorFlow、MXNet、PyTorch、Keras之上，详细说明请参见**MoXing使用说明**。如果您使用的是MoXing框架编写训练脚本，在创建训练作业时，请根据您选用的接口选择其对应的AI引擎和版本。
- “efficient_ai”是华为云ModelArts团队自研的加速压缩工具，它支持对训练作业进行量化、剪枝和蒸馏来加速模型推理速度，详细说明请参见**efficient_ai使用说明**。
- **Ascend-Powered-Engine**仅在“华北-北京四”区域支持。

新旧版训练预置引擎差异

- 新版的预置训练引擎默认安装Moxing2.0.0及以上版本。
- 新版的预置训练引擎统一使用了Python3.7及以上版本。
- 新版镜像修改了默认的HOME目录，由“/home/work”变为“/home/ma-user”，请注意识别训练代码中是否有“/home/work”的硬编码。
- 提供预置引擎类型有差异。新版的预置引擎在常用的训练引擎上进行了升级。

  如果您需要使用旧版训练引擎，单击显示旧版引擎即可选择旧版引擎。新旧版支持的预置引擎差异请参考**表3-3**。详细的训练引擎版本说明请参考**新版训练和旧版训练分别支持的AI引擎**。

表 3-3 新旧版预置引擎差异

工作环境	预置训练I引擎与版本	旧版训练	新版训练
TensorFlow	Tensorflow-1.8.0	√	x
	Tensorflow-1.13.1	√	后续版本支持
	Tensorflow-2.1.0	√	√
MXNet	MXNet-1.2.1	√	x
Caffe	Caffe-1.0.0	√	x
Spark_MLlib	Spark-2.3.2	√	x
Ray	RAY-0.7.4	√	x
XGBoost-Sklearn	XGBoost-0.80-Sklearn-0.18.1	√	x
PyTorch	PyTorch-1.0.0	√	x
	PyTorch-1.3.0	√	x
	PyTorch-1.4.0	√	x
	PyTorch-1.8.0	x	√
Ascend-Powered-Engine	Mindspore-1.1.1	√	x
	Mindspore-1.3.0	x	√

版权所有 © 华为技术有限公司

工作环境	预置训练引擎与版本	旧版训练	新版训练
	Tensorflow-1.15	√	√
MPI	MindSpore-1.3.0	x	√
Horovod	horovod_0.20.0-tensorflow_2.1.0	x	√
	horovod_0.22.1-pytorch_1.8.0	x	√
MindSpore-GPU	MindSpore-1.1.0	√	x
	MindSpore-1.2.0	√	x

3.3.2 开发自定义脚本

当您使用自定义脚本创建算法时，您需要提前完成算法的代码开发。本章详细介绍如何改造本地代码以适配ModelArts上的训练。

创建算法时，您需要在创建页面提供代码目录路径、代码目录路径中的启动文件、输入路径参数和训练输出路径参数。这四种输入搭建了用户代码和ModelArts后台交互的桥梁。

- 代码目录路径

  您需要在OBS桶中指定代码目录，并将训练代码、依赖安装包或者预生成模型等训练所需文件上传至该代码目录下。训练作业创建完成后，ModelArts会将代码目录及其子目录下载至后台容器中。

  请注意不要将训练数据放在代码目录路径下。训练数据比较大，训练代码目录在训练作业启动后会下载至后台，可能会有下载失败的风险。建议训练代码目录大小不超过50MB。

- 代码目录路径中的启动文件

  代码目录路径中的启动文件作为训练启动的入口，当前只支持python格式。

- 输入路径参数

  训练数据需上传至OBS桶或者存储至**数据集**中。在训练代码中，用户需解析**输入路径参数**。系统后台会自动下载输入参数路径中的训练数据至训练容器的本地目录。请保证您设置的桶路径有读取权限。在训练作业启动后，ModelArts会挂载硬盘至“/cache”目录，用户可以使用此目录来存储临时文件。“/cache”目录大小请参考**训练环境中不同规格资源“/cache”目录的大小**。

- 训练输出路径参数

  建议设置一个空目录为训练输出路径。在训练代码中，您需要解析**输出路径参数**。系统后台会自动上传训练输出至指定的训练输出路径，请保证您设置的桶路径有写入权限和读取权限。

在ModelArts中，训练代码需包含以下步骤：

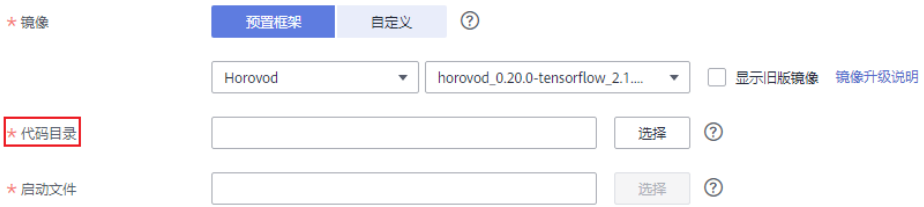
（可选）引入依赖

1. 当您使用自定义脚本创建算法的时候，如果您的模型引用了其他依赖，您需要在创建算法的“代码目录”下放置相应的文件或安装包。

版权所有 © 华为技术有限公司

- 安装python依赖包请参考**模型中引用依赖包时，如何创建训练作业?**
- 安装C++的依赖库请参考**如何安装C++的依赖库?**
- 在预训练模型中加载参数请参考**如何在训练中加载部分训练好的参数?**

图 3-2 选择代码目录并指定模型启动文件



解析输入路径参数、输出路径参数

运行在ModelArts的模型读取存储在OBS服务的数据，或者输出至OBS服务指定路径，输入和输出数据需要配置3个地方：

1. 训练代码中需解析输入路径参数和输出路径参数。ModelArts推荐以下方式实现参数解析。

```
import argparse
# 创建解析
parser = argparse.ArgumentParser(description="train mnist",
                                 formatter_class=argparse.ArgumentDefaultsHelpFormatter)
# 添加参数
parser.add_argument('--train_url', type=str,
                    help='the path model saved')
parser.add_argument('--data_url', type=str, help='the training data')
# 解析参数
args, unkown = parser.parse_known_args()
```

   完成参数解析后，用户使用“data_url”、“train_url”代替算法中数据来源和数据输出所需的路径。

2. 在使用自定义脚本创建自定义算法时，根据步骤1中的代码参数设置，您需要在创建算法页面配置已定义的输入输出参数。
   - 训练数据是算法开发中必不可少的输入。输入数据默认配置为“数据来源”，代码参数为“data_url”，也支持用户根据步骤1的算法代码自定义代码参数。

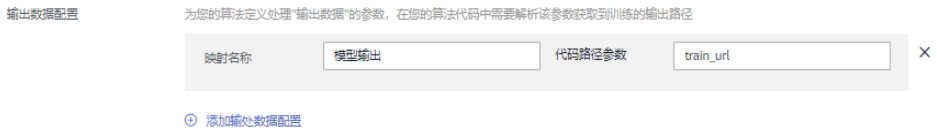
     图 3-3 解析输入路径参数“data_url”

   - 模型训练结束后，训练模型以及相关输出信息需保存在OBS路径。输出数据默认配置为“模型输出”，代码参数为“train_url”，也支持用户根据步骤1的算法代码自定义输出路径参数。

版权所有 © 华为技术有限公司

图 3-4 解析输出路径参数“train_url”



3. 在创建训练作业时，填写输入路径和输出路径。
   训练输入选择对应的OBS路径或者数据集路径；训练输出选择对应的OBS路径。

   图 3-5 训练输入和输出设置

训练代码正文和保存模型

训练代码正文和保存模型涉及的代码与您使用的AI引擎密切相关。以下案例以Tensorflow框架为例，训练代码中解析参数方式采用tensorflow接口tf.flags.FLAGS接受命令行参数：

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

import moxing as mox

tf.flags.DEFINE_integer('max_steps', 1000, 'number of training iterations.')
tf.flags.DEFINE_string('data_url', '/home/jnn/nfs/mnist', 'dataset directory.')
tf.flags.DEFINE_string('train_url', '/home/jnn/temp/delete', 'saved model directory.')

FLAGS = tf.flags.FLAGS


def main(*args):
    # Train model
    print('Training model...')
    mnist = input_data.read_data_sets(FLAGS.data_url, one_hot=True)
    sess = tf.InteractiveSession()
    serialized_tf_example = tf.placeholder(tf.string, name='tf_example')
    feature_configs = {'x': tf.FixedLenFeature(shape=[784], dtype=tf.float32),}
    tf_example = tf.parse_example(serialized_tf_example, feature_configs)
    x = tf.identity(tf_example['x'], name='x')
    y_ = tf.placeholder('float', shape=[None, 10])
    w = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
    sess.run(tf.global_variables_initializer())
    y = tf.nn.softmax(tf.matmul(x, w) + b, name='y')
    cross_entropy = -tf.reduce_sum(y_ * tf.log(y))

    tf.summary.scalar('cross_entropy', cross_entropy)

    train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```


版权所有 © 华为技术有限公司

```
    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))
    tf.summary.scalar('accuracy', accuracy)
    merged = tf.summary.merge_all()
    test_writer = tf.summary.FileWriter(FLAGS.train_url, flush_secs=1)

    for step in range(FLAGS.max_steps):
        batch = mnist.train.next_batch(50)
        train_step.run(feed_dict={x: batch[0], y_: batch[1]})
        if step % 10 == 0:
            summary, acc = sess.run([merged, accuracy], feed_dict={x: mnist.test.images, y_: mnist.test.labels})
            test_writer.add_summary(summary, step)
            print('training accuracy is:', acc)
    print('Done training!')

    builder = tf.saved_model.builder.SavedModelBuilder(os.path.join(FLAGS.train_url, 'model'))

    tensor_info_x = tf.saved_model.utils.build_tensor_info(x)
    tensor_info_y = tf.saved_model.utils.build_tensor_info(y)

    prediction_signature = (
        tf.saved_model.signature_def_utils.build_signature_def(
            inputs={'images': tensor_info_x},
            outputs={'scores': tensor_info_y},
            method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME))

    builder.add_meta_graph_and_variables(
        sess, [tf.saved_model.tag_constants.SERVING],
        signature_def_map={
            'predict_images':
                prediction_signature,
        },
        main_op=tf.tables_initializer(),
        strip_default_attrs=True)

    builder.save()

    print('Done exporting!')


if __name__ == '__main__':
    tf.app.run(main=main)
```

新旧版训练代码适配的差异

旧版训练中，用户需要在输入输出数据上做如下配置：

```
#解析命令行参数
import argparse
parser = argparse.ArgumentParser(description='MindSpore Lenet Example')
parser.add_argument('--data_url', type=str, default="./Data",
                    help='path where the dataset is saved')
parser.add_argument('--train_url', type=str, default="./Model", help='if is test, must provide\
                    path where the trained ckpt file')
args = parser.parse_args()
...
#下载云上数据参数至容器本地，在代码中使用local_data_path代表训练输入位置
mox.file.copy_parallel(args.data_url, local_data_path)
...
#上传容器本地数据至云上obs路径
mox.file.copy_parallel(local_output_path, args.train_url)
```

3.3.3 创建算法

针对您在本地或使用其他工具开发的算法，支持上传至ModelArts中统一管理。在创建自定义算法过程中，您需要关注以下内容：

 版权所有 © 华为技术有限公司

前提条件

- 数据已完成准备：已在ModelArts中创建可用的数据集，或者您已将用于训练的数据集上传至OBS目录。
- 请准备好自定义脚本，并上传至OBS目录。训练脚本开发指导参见**3.3.2 开发自定义脚本**。
- 已在OBS创建至少1个空的文件夹，用于存储训练输出的内容。
- 由于训练作业运行需消耗资源，确保帐户未欠费。
- 确保您使用的OBS目录与ModelArts在同一区域。

进入创建算法页面

1. 登录ModelArts管理控制台，单击左侧菜单栏的“算法管理”。
2. 在“我的算法”管理页面，单击“创建”，进入“创建算法”页面。

设置算法基本信息

基本信息包含“名称”和“描述”。

图 3-6 设置算法基本信息

设置创建方式

选择“预置镜像”创建算法。

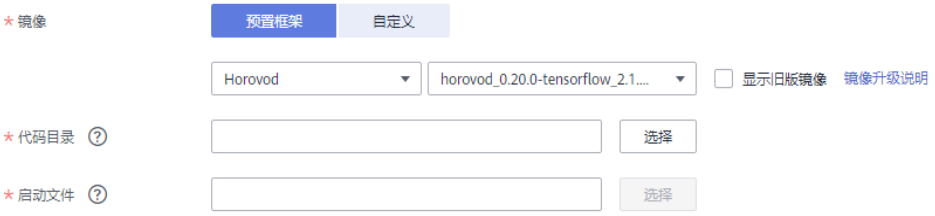
用户需根据实际算法代码情况设置“镜像”、“代码目录”和“启动文件”。选择的AI引擎和编写算法代码时选择的框架必须一致。例如编写算法代码使用的是TensorFlow，则在创建算法时也要选择TensorFlow。

版权所有 © 华为技术有限公司

表 3-4 创建方式参数说明

参数	说明
“预置镜像”	默认显示新版训练支持的AI引擎，勾选“显示旧版引擎”，可以选择旧版训练支持的引擎，具体请参考**3.3.1 使用自定义脚本简介**中的AI引擎介绍。
“代码目录”	算法代码存储的OBS路径。训练代码、依赖安装包或者预生成模型等训练所需文件上载至该代码目录下。 代码目录下不能存在他人上传的文件和目录，也不能存在无关的文件和目录，否则可能导致失败。 请注意不要将训练数据放在代码目录路径下。训练数据比较大，训练代码目录在训练作业启动后会下载至后台，可能会有下载失败的风险。 训练作业创建完成后，ModelArts会将代码目录及其子目录下载至后台容器中。 **说明** • 编程语言不限。 • 文件数（含文件、文件夹数量）不超过1024个。 • 文件总大小不超过5GB。
“启动文件”	必须为“代码目录”下的文件，且以“.py”或“.pyc”结尾，即ModelArts目前只支持使用Python语言编写的启动文件。 代码目录路径中的启动文件为训练启动的入口。

图 3-7 使用自定义脚本创建算法



配置输入输出数据

训练过程中，自定义算法需要从OBS桶或者数据集中获取数据进行模型训练，训练产生的输出结果也需要存储至OBS桶中。用户的算法代码中需解析输入输出参数实现ModelArts后台与OBS的数据交互，用户可以参考**3.3.2 开发自定义脚本**完成适配ModelArts训练的代码开发。

创建自定义算法时，用户需要将算法代码中定义的输入输出参数进行配置。

- 输入数据配置

版权所有 © 华为技术有限公司

表 3-5 输入数据配置

参数	参数说明
映射名称	输入参数的说明，用户可以自定义描述。默认为“数据来源”。
代码路径参数	如果您的算法代码中使用argparse解析data_url为输入数据参数，则在创建的算法需要配置输入数据代码参数为data_url。根据实际代码中的输入数据参数定义此处的名称。 此处设置的代码路径参数必须与算法代码中解析的输入数据参数保持一致，否则您的算法代码无法获取正确的输入数据。
是否添加约束	用户可以根据实际情况限制数据来源的方式，可以设置为支持数据存储位置或者ModelArts数据集。 如果用户选择数据来源为ModelArts数据集，还可以约束以下三种： • 标注类型。数据类型请参考**标注数据**。 • 数据格式。可选“Default”和“CarbonData”，支持多选。其中“Default”代表Manifest格式。 • 数据切分。仅“图像分类”、“物体检测”、“文本分类”和“声音分类”类型数据集支持进行数据切分功能。 可选“仅支持切分的数据集”、“仅支持未切分数据集”和“无限制”。数据切分详细内容可参考**发布数据版本**。
添加输入数据配置	用户可以根据实际算法确定多个输入数据来源。

图 3-8 输入数据配置

- 输出数据配置

版权所有 © 华为技术有限公司

表 3-6 输出数据配置

参数	参数说明
映射名称	输出参数的说明，用户可以自定义描述。默认为“输出数据”。
代码路径参数	如果您的算法代码中使用argparse解析train_url为训练输出参数，则在创建的算法需要配置输出数据参数为train_url。根据实际代码中的训练输出参数定义此处的名称。 此处设置的代码路径参数必须与算法代码中解析的训练输出参数保持一致，否则您的算法代码无法获取正确的输出路径。
添加输出数据配置	用户可以根据实际算法确定多个输出数据路径。

图 3-9 输出数据配置

定义超参

使用常用框架创建算法时，ModelArts支持用户自定义超参，方便用户查阅或修改。定义超参后会体现在启动命令中，以命令行参数的形式传入您的启动文件中。

1. 导入超参

   您可以单击“增加超参”手动添加超参。请注意超参输入的内容应该只包含大小写字母、中文、数字、空格、中划线、下划线、逗号和句号。

   图 3-10 添加超参

2. 编辑超参。超参参数说明参见表3-7。

   表 3-7 超参编辑参数

参数	说明
名称	填入超参名称。 超参名称支持64个以内字符，仅支持大小写字母、数字、下划线和中划线。

版权所有 © 华为技术有限公司

参数	说明
描述	填入超参的描述说明。 超参描述支持大小写字母、中文、数字、空格、中划线、下划线、逗号和句号。
类型	填入超参的数据类型。支持String、Integer、Float和Boolean。
默认值	填入超参的默认值。创建训练作业时，默认使用该值进行训练。
约束	单击约束。在弹出对话框中，支持用户设置默认值的取值范围或者枚举值范围。
必需	可选是或者否。如果您选择否，在使用该算法创建训练作业时，支持在创建训练作业页面删除该超参。如果您选择是，则不支持删除操作。

支持的策略

ModelArts支持用户使用自动化搜索功能。自动化搜索功能在零代码修改的前提下，自动找到最优的超参，有助于提高模型精度和收敛速度。详细的参数配置请参考**超参搜索配置**。

自动搜索目前仅支持pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64, tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64引擎。

添加训练约束

用户可以根据实际情况定义此算法的训练约束。

- 资源类型：可选“CPU”、“GPU”和“Ascend”，支持多选。
- 多卡训练：可选“支持”和“不支持”。
- 分布式训练：可选“支持”和“不支持”。

图 3-11 算法训练约束

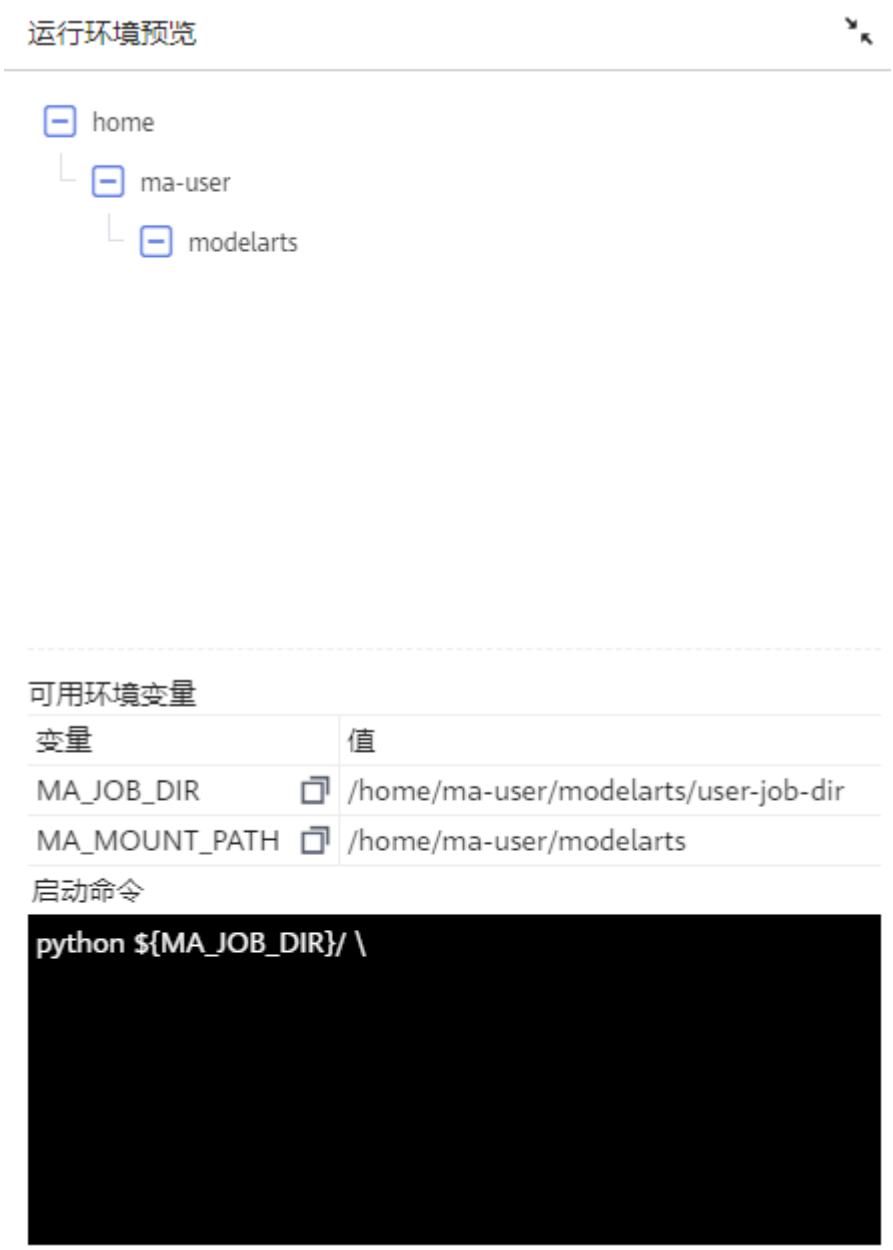
运行环境预览

创建训练作业时，可以打开创建页面右下方的运行环境预览窗口

，辅助您了解代码目录、启动文件、输入输出等数据配置在训练容器中的路径。

版权所有 © 华为技术有限公司

图 3-12 运行环境预览



后续操作

创建算法完成后，在“算法管理”页面，等待算法就绪。当新创建的算法状态变更为“就绪”时，即可执行其他操作。

可以使用算法快速创建训练作业，构建模型，详细操作请参见**4.1 创建训练作业**。

3.4 使用自定义镜像

订阅算法和预置镜像涵盖了大部分的训练场景。针对特殊场景，ModelArts支持用户构建自定义镜像用于模型训练。自定义镜像需上传至容器镜像服务（SWR），才能用于ModelArts上训练。

版权所有 © 华为技术有限公司

自定义镜像的制作要求用户对容器相关知识有比较深刻的了解，除非订阅算法和预置引擎无法满足需求，否则不推荐使用。

新版训练支持的自定义镜像使用说明，请参考**使用自定义镜像训练模型**。

3.5 旧版训练迁移至新版训练注意事项

新版训练和旧版训练的差异主要体现在以下3点:

- **新旧版创建训练作业方式差异**
- **新旧版训练代码适配的差异**
- **新旧版训练预置引擎差异**

新旧版创建训练作业方式差异

旧版训练支持使用自定义算法、使用预置算法、使用常用框架、使用自定义镜像方式创建训练作业。

新版训练支持使用市场订阅算法和自定义算法创建训练作业。

新版训练的创建方式有了更明确的类别划分，不影响用户的原本的训练作业，只是选择方式存在区别。

旧版中使用自定义算法和使用常用框架创建训练作业的用户，可以在新版训练中**使用自定义算法方式**创建训练作业。

旧版中使用预置算法的用户，可以在新版训练中**使用订阅算法**创建训练作业。

旧版中使用自定义镜像创建训练作业的用户，新版训练暂不支持在控制台中使用自定义镜像创建训练作业，建议使用开发环境创建自定义镜像，并调用SDK提交训练作业。

新旧版训练代码适配的差异

旧版训练中，用户需要在输入输出数据上做如下配置：

```
#解析命令行参数
import argparse
parser = argparse.ArgumentParser(description='MindSpore Lenet Example')
parser.add_argument('--data_url', type=str, default="./Data",
                    help='path where the dataset is saved')
parser.add_argument('--train_url', type=str, default="./Model", help='if is test, must provide\
                    path where the trained ckpt file')
args = parser.parse_args()
...
#下载云上数据参数至容器本地，在代码中使用local_data_path代表训练输入位置
mox.file.copy_parallel(args.data_url, local_data_path)
...
#上传容器本地数据至云上obs路径
mox.file.copy_parallel(local_output_path, args.train_url)
```

新版训练中，用户配置输入输出数据，无需书写下载数据的代码，在代码中把arg.data_url和arg.train_url当做本地路径即可，详情参考**开发自定义脚本**。

```
#解析命令行参数
import argparse
parser = argparse.ArgumentParser(description='MindSpore Lenet Example')
parser.add_argument('--data_url', type=str, default="./Data",
                    help='path where the dataset is saved')
parser.add_argument('--train_url', type=str, default="./Model", help='if is test, must provide\
```


版权所有 © 华为技术有限公司

```
                    path where the trained ckpt file')
args = parser.parse_args()
...
# 下载的代码无需设置，后续涉及训练数据和输出路径数据使用data_url和train_url即可
#下载云上数据参数至容器本地，在代码中使用local_data_path代表训练输入位置
#mox.file.copy_parallel(args.data_url, local_data_path)
...
#上传容器本地数据至云上obs路径
#mox.file.copy_parallel(local_output_path, args.train_url)
```

新旧版训练预置引擎差异

- 新版的预置训练引擎默认安装Moxing2.0.0及以上版本。
- 新版的预置训练引擎统一使用了Python3.7及以上版本。
- 新版镜像修改了默认的HOME目录，由“/home/work”变为“/home/ma-user”，请注意识别训练代码中是否有“/home/work”的硬编码。
- 提供预置引擎类型有差异。新版的预置引擎在常用的训练引擎上进行了升级。

  如果您需要使用旧版训练引擎，单击显示旧版引擎即可选择旧版引擎。新旧版支持的预置引擎差异请参考**表3-8**。详细的训练引擎版本说明请参考**新版训练和旧版训练分别支持的AI引擎**。

表 3-8 新旧版预置引擎差异

工作环境	预置训练I引擎与版本	旧版训练	新版训练
TensorFlow	Tensorflow-1.8.0	√	x
	Tensorflow-1.13.1	√	后续版本支持
	Tensorflow-2.1.0	√	√
MXNet	MXNet-1.2.1	√	x
Caffe	Caffe-1.0.0	√	x
Spark_MLlib	Spark-2.3.2	√	x
Ray	RAY-0.7.4	√	x
XGBoost-Sklearn	XGBoost-0.80-Sklearn-0.18.1	√	x
PyTorch	PyTorch-1.0.0	√	x
	PyTorch-1.3.0	√	x
	PyTorch-1.4.0	√	x
	PyTorch-1.8.0	x	√
Ascend-Powered-Engine	Mindspore-1.1.1	√	x
	Mindspore-1.3.0	x	√
	Tensorflow-1.15	√	√

版权所有 © 华为技术有限公司

工作环境	预置训练引擎与版本	旧版训练	新版训练
MPI	MindSpore-1.3.0	x	√
Horovod	horovod_0.20.0-tensorflow_2.1.0	x	√
	horovod_0.22.1-pytorch_1.8.0	x	√
MindSpore-GPU	MindSpore-1.1.0	√	x
	MindSpore-1.2.0	√	x

版权所有 © 华为技术有限公司

4 完成一次简单训练

4.1 创建训练作业

训练管理模块是ModelArts不可或缺的功能模块，用于创建训练作业、查看训练情况以及管理训练版本。模型训练是一个不断迭代和优化的过程。在训练模块的统一管理下，方便用户试验算法、数据和超参数的各种组合，便于追踪最佳的模型与输入配置，您可以通过不同版本间的评估指标比较，确定最佳训练作业。

前提条件

- 数据已完成准备：已在ModelArts中创建可用的数据集，或者您已将用于训练的数据上传至OBS目录。
- “算法管理”中，已完成**3.3 使用自定义脚本**或者已**3.2 使用订阅算法**。
- 已在OBS创建至少1个空的文件夹，用于存储训练输出的内容。
- 由于训练作业运行需消耗资源，确保帐户未欠费。
- 确保您使用的OBS目录与ModelArts在同一区域。

创建训练作业

1. 登录ModelArts管理控制台，在左侧导航栏中选择“全局配置”，检查是否配置了访问授权。若未配置，请参考**使用委托授权**完成操作。

版权所有 © 华为技术有限公司

图 4-1 配置授权

2. 登录ModelArts管理控制台，在左侧导航栏中选择“训练管理 > 训练作业（New）”，默认进入“训练作业”列表。
3. 单击“创建训练作业”，进入“创建训练作业”页面，在该页面填写训练作业相关参数。

图 4-2 进入创建训练作业页面

4. 填写基本信息。基本信息包含“名称”和“描述”。
5. 填写算法配置参数。请根据您的算法类型选择算法来源。

表 4-1 作业参数说明

参数名称	子参数	说明
算法	自定义算法	选择“自定义算法”页签，设置“镜像来源”、“代码目录”和“启动文件”，详细操作指导参见3.3.3 创建算法。

版权所有 © 华为技术有限公司

参数名称	子参数	说明
算法	我的算法	选择“我的算法”页签，勾选“算法名称”左侧按钮，即选中该算法。 如果没有创建算法，请单击“创建”进入创建算法页面，详细操作指导参见**3.3.3 创建算法**。
	我的订阅	选择“我的订阅”页签，单击下拉三角标选择算法版本。勾选算法版本，即选中该算法。 如果没有订阅算法，请单击“订阅更多算法”进入算法订阅页面，查找算法并订阅。
训练输入	-	请选择OBS中数据存储位置作为训练输入。 此处训练输入与您选择算法的“输入数据配置”匹配，参见**表3-5**。例如您的训练代码中使用argparse解析data_url为输入数据超参，则在创建的算法需要配置输入数据代码参数为data_url。在此处，您可以选择数据集或者OBS存储位置作为输入的路径。训练启动时，会下载输入路径中的数据至运行容器。
	数据集	从ModelArts数据管理中选择可用的数据集及其版本。 单击“选择数据集”，在弹出的对话框中，勾选目标数据集并选择对应的版本。 **说明** 如果该选项置灰，表示该算法的输入不支持选择数据集。
	数据存储位置	从OBS桶中选择训练数据。 单击“数据存储位置”，从弹出的对话框中，选择数据存储的OBS桶及其文件夹。 **说明** 如果该选项置灰，表示该算法的输入不支持选择数据存储位置。
训练输出	-	选择训练结果的存储位置（OBS路径）。为避免出现错误，建议选择一个空目录用作“训练输出”。 模型输出参数与您选择算法的“输出数据配置”匹配，参见**表3-6**。例如您的训练代码中使用argparse解析train_url为输出路径参数，则在创建的算法需要配置输入数据代码参数为train_url。在此处，您可以选择OBS路径作为云上存储位置。训练过程中时，会上传训练输出至指定OBS路径。
超参	-	此参数根据您选择的算法不同而不同。 如果您在创建算法时已经定义超参，此处会显示该算法所有超参。超参可修改、可删除状态取决于您在创建算法时的超参约束设置，详见**定义超参**。 如果您在创建算法时设置支持自定义超参，您可以单击“增加超参”，添加多条超参用于调优。
环境变量	-	开启此参数用于增加用户自行设置的环境变量。

版权所有 © 华为技术有限公司

参数名称	子参数	说明
故障自动重启	-	开启此参数后，训练作业失败时会自动重新下发并运行训练作业。此处设置训练重启次数。

说明

根据您选择的算法不同，训练输入、训练输出和调优参数显示不同。

当“训练输入”显示为“当前算法没有输入通道”，表明您不需要在该页面配置输入数据。

当“训练输出”显示为“当前算法没有输出通道”，表明您不需要在该页面配置输出路径。

当“超参”显示为“当前所选算法不支持自定义超参”，表明您不需要在该页面配置超参用于调优。

6. 选择训练资源的规格。训练参数的可选范围与已有算法的使用约束保持一致。

表 4-2 资源参数说明

参数名称	说明
资源池	选择训练作业资源池。训练作业支持选择“公共资源池”和“专属资源池”。
资源类型	可选CPU、GPU、Ascend。
规格	针对不同的资源类型，选择资源规格。如果您的算法已定义使用CPU或GPU，根据已有算法约束条件，您可以在有效规格选择合适的资源规格，无效选项置灰不可选。 不同的资源类型的数据盘容量是不同的，为避免训练过程中出现内存不足的情况，请参考**训练环境中不同规格资源“/cache”目录的大小**。
计算节点个数	选择计算节点的个数。默认值为“1”。
作业日志路径	• 如果用户选择CPU或者GPU规格，可以选择是否打开永久保存按钮。 打开永久保存按钮后，用户需要选择作业运行中产生的日志文件存储路径。 • 如果用户选择Ascend规格，用户需要选择作业运行中产生的日志文件存储路径。 请选择一个空的OBS目录用于存储日志文件。 **说明** 请注意选择的OBS目录有读写权限。

版权所有 © 华为技术有限公司

参数名称	说明
自动停止	使用付费资源时会出现此参数。 • 启用该参数并设置时间后，训练作业将在指定时间自动停止。 • 如果不启用此参数，训练作业将一直运行，同时一直收费，自动停止功能可以帮您避免产生不必要的费用。 自动停止时间支持设置为“1小时后”、“2小时后”、“4小时后”、“6小时后”、“自定义”。

7. 单击“提交”，完成训练作业的创建。

   训练作业一般需要运行一段时间。要查看训练作业实时情况，您可以前往训练作业列表，查看训练作业的基本情况。在训练作业列表中，刚创建的训练作业“状态”为“初始化”，当训练作业的“状态”变为“运行成功”时，表示训练作业运行结束，其生成的模型将存储至对应的“训练输出”中。当训练作业的“状态”变为“运行失败”时，您可以单击训练作业的名称，进入详情页面，通过查看日志等手段处理问题，详情请参考**4.2 查看作业详情**。

   说明

   训练作业创建完成后，将立即启动，运行过程中将按照您选择的资源按需计费。

4.2 查看作业详情

训练作业运行时，支持查看以下训练配置、训练过程日志、资源占用和模型评估结果四个场景，判断此训练作业是否满意。

4.2.1 训练作业详情

1. 登录ModelArts管理控制台，在左侧导航栏中选择“训练管理 >训练作业（New）”，默认进入“训练作业”列表。
2. 在训练作业列表中，您可以单击作业名称，查看该作业的基本信息与训练作业参数。

版权所有 © 华为技术有限公司

图 4-3 查看训练作业详情

3. 在训练作业详情页的左侧，可以查看此次训练作业的配置、算法的配置的相关信息。

   a. 训练作业基本信息

   表 4-3 训练作业基本信息

参数	说明
作业ID	训练作业唯一标识。
状态	训练作业状态。
创建时间	记录训练作业创建时间。
运行时间	记录训练作业运行时长。

   b. 训练作业参数

   表 4-4 训练作业参数

参数	说明
算法名称	本次训练作业使用的算法。
AI引擎	本次训练作业使用的AI引擎。
代码目录	训练作业脚本所在的代码目录。如果您是市场订阅算法，该参数不显示内容。
启动文件	训练作业启动文件位置。如果您是市场订阅算法，该参数不显示内容。
计算节点个数	本次训练作业设置的计算节点个数。
规格	本次训练作业使用的训练规格。

版权所有 © 华为技术有限公司

参数	说明
作业日志路径	设置的训练作业日志输出路径。
训练输入-输入路径	本次训练中，输入数据的OBS路径。
训练输入-训练参数名称	算法代码中，输入路径指代的参数。
训练输入-本地路径（训练参数值）	训练启动后，ModelArts将OBS路径中的数据下载至后台容器，本地路径指ModelArts后台容器中存储输入数据的路径。
训练输出-输出路径	本次训练中，输出数据的OBS路径。
训练输出-训练参数名称	算法代码中，输出路径指代的参数。
训练输出-本地路径（训练参数值）	ModelArts后台容器中存储训练输出的路径。
超参	本次训练作业使用的超参。
环境变量	本次训练作业设置的环境变量。

4.2.2 训练作业事件

训练作业的（从用户可看见训练任务开始）整个生命周期中，每一个关键事件点在系统后台均有记录，用户可随时在对应训练作业的详情页面进行查看。

方便用户更清楚的了解训练作业运行过程，遇到任务异常时，更加准确的排查定位问题。可查看的事件点包括：

- 训练作业准备中，查看当前训练作业都已完成了哪些准备动作或都有哪些关键事件点。
- 训练作业运行中，查看都有哪些关键阶段信息。

训练运行到结束的过程中，关键事件支持手动/自动刷新。

查看操作

1. 在ModelArts管理控制台的左侧导航栏中选择“训练管理 >训练作业”，在训练作业列表中，您可以单击作业名称，进入训练作业详情页面。
2. 在训练作业详情页面的右上角，单击“查看事件”，查看事件信息。

4.2.3 训练日志详情

在训练作业详情页，训练日志窗口提供日志预览、日志下载、日志中搜索关键字能力和训练故障识别能力。

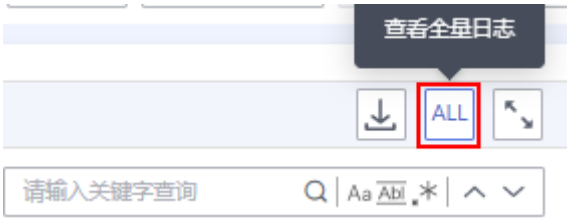
- 预览

版权所有 © 华为技术有限公司

系统日志窗口提供训练日志预览功能，并支持查看不同计算节点日志，您可以通过右侧下拉框选择目标节点预览。

当日志文件过大时，系统日志窗口仅加载最新的部分日志，并在日志窗口上方提供全量日志访问链接。打开该链接可在新页面查看全部日志。

图 4-4 查看全量日志



说明

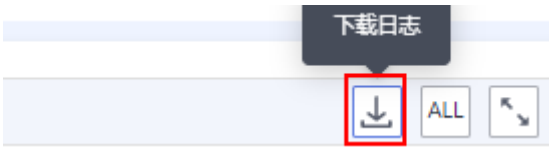
- 如果全部日志超过500M，可能会引起浏览页面卡顿，建议您直接下载日志查看。
- 预览链接在生成后的一小时内，支持任何人打开并查看。您可以分享链接至他人。请注意日志中不能包含隐私内容，否则会造成信息泄露。

- 下载

  训练日志仅保留30天，超过30天会被清理。如果用户需要永久保存日志，请单击系统日志窗口右上角下载按钮下载日志至本地保存，支持批量下载多节点日志。用户也可以在**创建训练作业**时打开永久保存日志按钮，保存训练日志至指定OBS路径。

  针对使用Ascend规格创建的训练作业，部分系统日志暂不支持直接在训练日志窗口下载，请用户在创建训练作业时指定OBS路径用于保存训练日志。

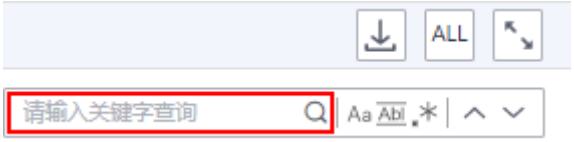
图 4-5 下载日志



- 搜索关键字

  用户可以在系统日志右上角的搜索框搜索关键字。系统支持高亮关键字并实现搜索结果间的跳转。

图 4-6 搜索关键字



版权所有 © 华为技术有限公司

说明

- 搜索功能仅支持搜索当前页面加载的日志，若日志加载不全（请关注页面提示）则需要下载或者通过打开全量日志访问链接进行搜索
- 全量日志访问链接打开的新页面可以通过Ctrl+F进行搜索

- 训练故障识别

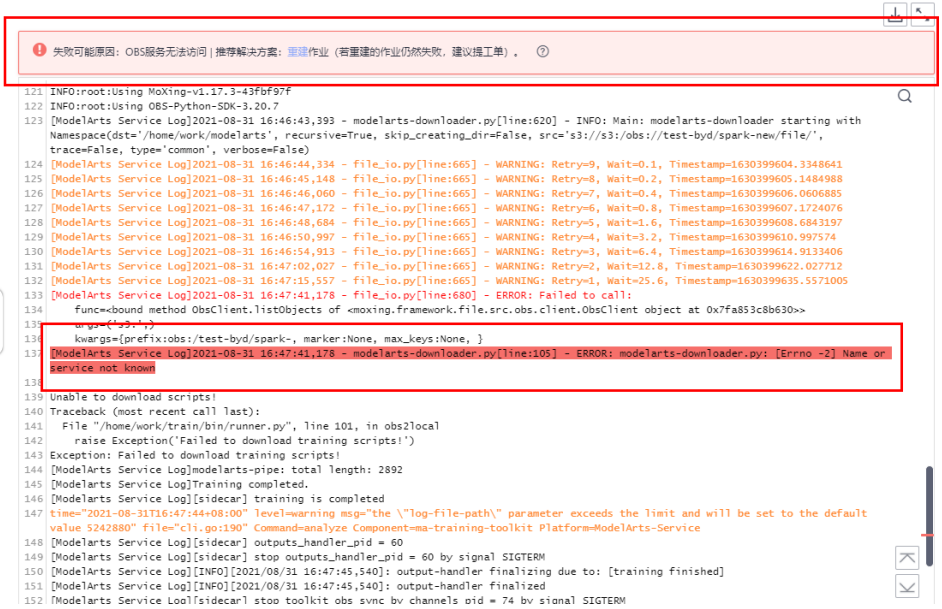
  如果您的训练作业运行失败，我们会自动识别导致您作业失败的原因，在界面上给出提示。提示包括三部分：失败的可能原因、推荐的解决方案以及对应的日志（底色标红）。解决方案可能会有以下类型：

  1.文档：会提供当前故障对应的一个指导文档，请按照文档提示修复问题。

  2.重建作业：建议重建作业进行重试，大概率能修复问题。若重试后仍然失败，建议提工单。

  3.提工单：建议提工单解决。

图 4-7 训练故障识别



说明

- 系统会对部分常见训练错误给出分析建议，目前还不能识别所有错误。
- 提供的失败可能原因仅供参考。
- 如果提供的解决方案不能解决您的问题，可以提工单进行人工咨询。
- 针对分布式作业，只会显示当前节点的一个分析结果，作业的失败需要综合各个节点的失败原因做一个综合判断。

训练日志说明

如果选择Ascend 910规格创建训练作业，日志结构举例说明如下：

```
obs://dgg-test-user/ascend910-test-cases/log-out/                                  # 作业日志路径
├── modelarts-job-9ccf15f2-6610-42f9-ab99-059ba049a41e-proc-rank-0-device-0.txt    # proc log  单卡训练日志
└── modelarts-job-9ccf15f2-6610-42f9-ab99-059ba049a41e-worker-0.log                # training log 训练日志合集
```


版权所有 © 华为技术有限公司

```
└─── ascend                                          # Ascend日志
   └─── log                  # 默认不生成该文件
      ├─── device        # device log  默认不生成
      │   └─── ...
      └─── plog
         └─── ...
```

主要生成日志类型如下：

- **训练平台日志**

  training log是训练日志合集，proc log是单卡训练日志重定向文件，方便用户快速定位对应计算节点的日志。

- **Ascend 日志**

  默认设置 plog 日志打屏，“ascend/log” 文件夹及相关文件默认不生成日志文件。

  您可以通过ma-pre-start 脚本修改默认环境变量配置。

  ```
  ASCEND_GLOBAL_LOG_LEVEL=3     # 日志级别设置  debug级别为0;info 级别为1;warning级别为 2;error级别为4
  ASCEND_SLOG_PRINT_TO_STDOUT=1  # plog日志是否打屏
  ASCEND_GLOBAL_EVENT_ENABLE=1   # 设置事件级别  不开启Event日志级别为0；开启Event日志级别为1
  ```

  说明

  ma-pre-start脚本

  在与训练启动文件同级的目录下放置 ma-pre-start.sh or ma-pre-start.py 脚本。

  在训练启动文件被执行前，系统会在 /home/work/user-job-dir/ 目录下执行上述 pre-start 脚本

  使用该机制可以更新容器镜像内安装的 Ascend RUN 包，或者设置一些训练运行时额外需要的全局环境变量。

训练平台日志

```
样例：modelarts-job-9ccf15f2-6610-42f9-ab99-059ba049a41e-worker-0.log
统一日志格式：modelarts-job-[job id]-[task id].log
```

其中 task id 为分布式作业中的节点id，单机作业的 task id 默认为 worker-0。

training log中包括“pip-requirement.txt 安装日志”、“ma-pre-start 日志”、“davincirun 日志”、“训练进程日志”、“MA平台日志”等。

- **MA平台日志**

  类型一：[ModelArts Service Log] xxx

  ```
  [ModelArts Service Log][init] download code_url: s3://dgg-test-user/ascend910-test-cases/mindspore/lenet/
  ```

  类型二：time=“xxx” level=“xxx” msg=“xxx” file=“xxx” Command=xxx Component=xxx Platform=xxx

  ```
  time="2021-07-26T19:24:11+08:00" level=info msg="start the periodic upload task, upload period = 5 seconds " file="upload.go:46" Command=obs/upload Component=ma-training-toolkit Platform=ModelArts-Service
  ```

  单机作业只会生成一个 training log 文件。MA平台日志主要用于运维人员定位平台问题，如果不涉及可以通过上述类型模式规则进行筛除。

- **proc log**

  proc log是单卡训练日志重定向文件，方便用户快速定位对应计算节点的日志。

  ```
  样例：modelarts-job-9b43aba3-ed9c-48c1-9f5e-2d61d20dce65-proc-rank-6-device-6.txt
  日志文件格式：  [modelarts-job-uuid]-proc-rank-[rank id]-device-[device logic id].txt
  ```


版权所有 © 华为技术有限公司

device id 为顺序编号 id。

Ascend 日志

样例：

```
└─── ascend
   └─── log
      ├─── device       # device log
      │    └─── ...
      └─── plog       # plog
           └─── ...
```

表 4-5 Ascend 日志说明

日志说明	日志产生源	日志落盘位置
用户进程,在HOST侧产生的日志（例如:ACL /GE）	HOST侧	HOST侧: Linux: ~/ascend/log/plog
用户进程，在DEVICE侧产生的AICPU、HCCP的日志，回传到HOST侧	DEVICE侧	HOST侧: Linux:~/ascend/log/device-{device-id}

说明

由于 Device Log 在训练进程结束后获取；因此假若出现如下情况，Ascend Device Log 会获取不到。

- 节点异常重启
- 被主动停止的节点

4.2.4 资源占用情况

用户可以通过资源占用情况窗口查看计算节点的资源使用情况。

操作一：如果训练作业使用多个计算节点，可以通过实例名称的下拉框切换节点。

操作二：单击图例“cpuUsage”、“gpuMemUsage”、“gpuUtil”、“memUsage”“npuMemUsage”、“npuUtil”、可以添加或取消对应参数的使用情况图。

操作三：鼠标悬浮在图片上的时间节点，可查看对应时间节点的占用率情况。

版权所有 © 华为技术有限公司

图 4-8 资源占用情况

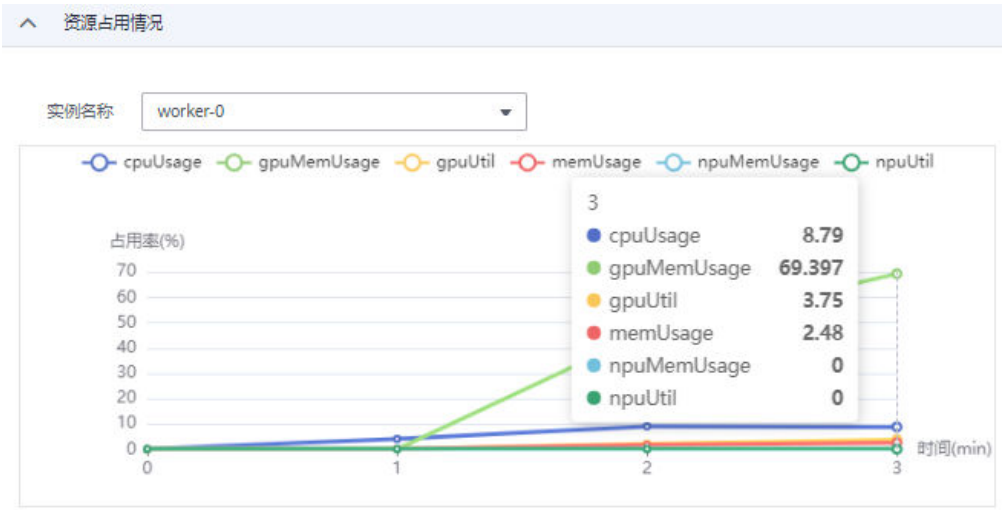


表 4-6 参数说明

参数	说明
cpuUsage	cpu使用率。
gpuMemUsage	gpu内存使用率。
gpuUtil	gpu使用情况
memUsage	内存使用率。
npuMemUsage	npu内存使用率。
npuUtil	npu使用情况

4.2.5 评估结果

针对使用ModelArts官方发布的算法创建训练作业时，其训练作业详情支持查看评估结果。根据模型训练的业务场景不同，会提供不同的评估结果。由于每个模型情况不同，系统将自动根据您的模型指标情况，给出一些调优建议，请仔细阅读界面中的建议和指导，对您的模型进行进一步的调优。

如果您的训练脚本中按照ModelArts规范添加了相应的评估代码，在训练作业运行结束后，也可在作业详情页面查看评估结果。

版权所有 © 华为技术有限公司

图 4-9 评估结果

4.3 停止、重建或查找作业

停止训练作业

在训练作业列表中，针对“创建中”、“等待中”、“运行中”的训练作业，您可以单击“操作”列的“终止”，停止正在运行中的训练作业。

训练作业停止后，ModelArts将停止计费。

运行结束的训练作业，如“已完成”、“运行失败”、“已终止”、“异常”的作业，不涉及“终止”操作。

重建训练作业

当对创建的训练作业不满意时，您可以单击操作列的重建，重新创建训练作业。在重创训练作业页面，会自动填入上一次训练作业设置的参数，您仅需在原来的基础上进行修改即可重新创建训练作业。

查找训练作业

当用户使用IAM账号登录时，训练作业列表会显示IAM账号下所有训练作业。ModelArst提供查找训练作业功能帮助用户快速查找训练作业。

操作一：打开“只显示自己”按钮。打开按钮后，训练作业列表仅显示当前子账号下创建的训练作业。

操作二：支持筛选训练作业状态。

操作三：支持查询训练作业名称。

操作四：支持高级搜索。高级搜索包括算法名称筛选、创建日期筛选。

图 4-10 查找训练作业

4.4 清除训练作业资源

如果不再需要使用此训练任务，建议清除相关资源，避免产生不必要的费用。

版权所有 © 华为技术有限公司

- 在“训练作业”页面，“删除”运行结束的训练作业。您可以单击“操作”列的“删除”，删除对应的训练作业。
- 进入OBS，删除本示例使用的OBS桶及文件。

完成资源清除后，您可以在总览页面的使用情况确认资源删除情况。

图 4-11 查看使用情况

版权所有 © 华为技术有限公司

5 训练进阶

5.1 训练故障自动恢复

5.1.1 容错检查

用户在训练模型过程中，存在因硬件故障而产生的训练失败场景。针对硬件故障场景，ModelArts提供容错检查功能，帮助用户隔离故障节点，优化用户训练体验。

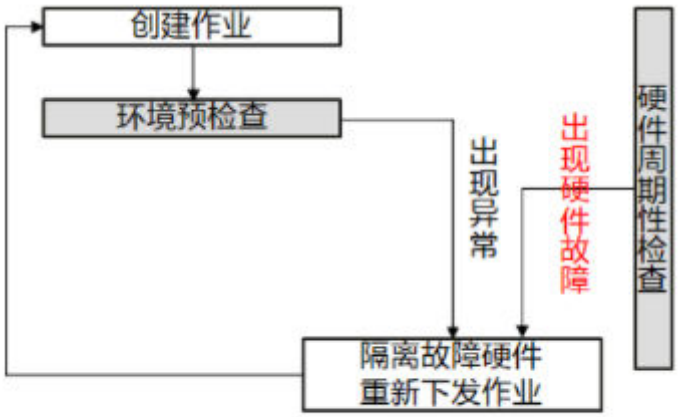
容错检查包括两个检查项：环境预检测与硬件周期性检查。当环境预检查或者硬件周期性检查任一检查项出现故障时，隔离故障硬件并重新下发训练作业。针对于分布式场景，容错检查会检查本次训练作业的全部计算节点。

下图中有四个场景，其中场景四为正常训练作业失败场景，其他三个场景下可开启容错功能进行训练作业自动恢复。

- 场景一：环境预检测失败、硬件检测出现故障，隔离所有故障节点并重新下发训练作业。

 版权所有 © 华为技术有限公司

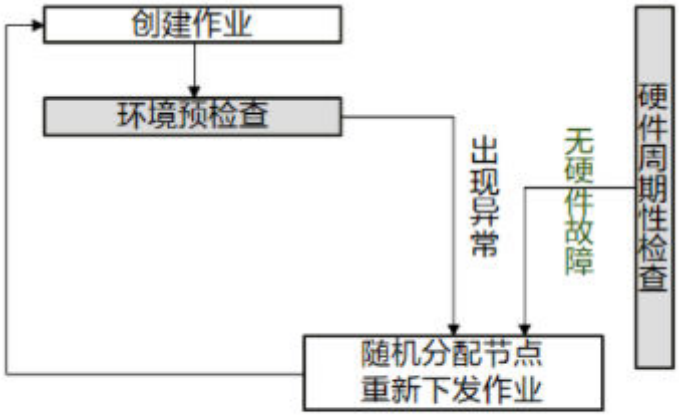
图 5-1 预检失败&硬件故障



1. 预检失败&硬件故障

- 场景二：环境预检测失败、硬件无故障，隔离所有故障节点并重新下发训练作业。

图 5-2 预检失败&硬件正常

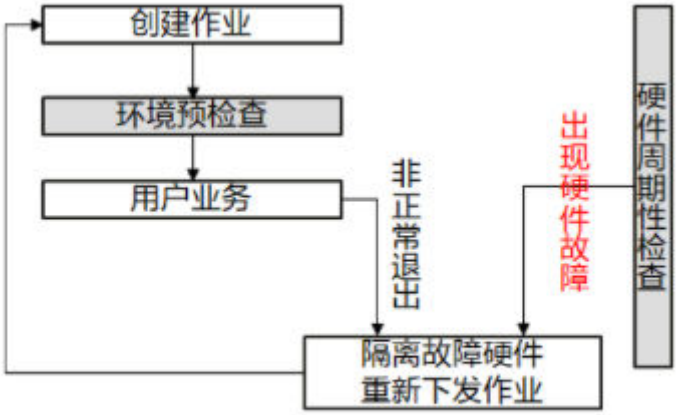


2. 预检失败&硬件正常

- 场景三：环境预检测成功并进入用户业务阶段，硬件检测出现故障并且用户业务非正常退出，隔离所有故障节点并重新下发训练作业。

版权所有 © 华为技术有限公司

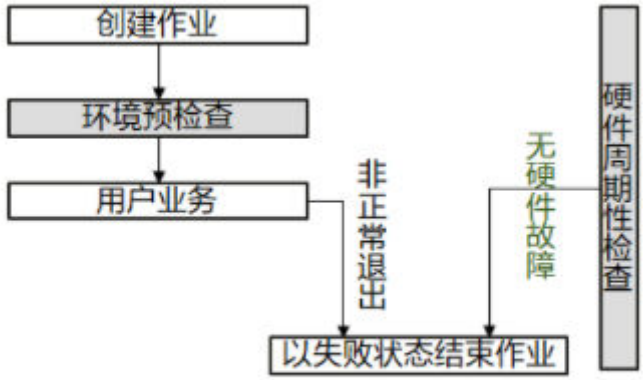
图 5-3 业务失败&硬件故障



3. 业务失败&硬件故障

- 场景四：环境预检测成功并进入用户业务阶段，硬件无故障，当用户业务异常时以失败状态结束作业。

图 5-4 业务失败&硬件正常



4. 业务失败&硬件正常

隔离故障节点后，系统会在新的计算节点上重新创建训练作业。如果资源池规格紧张，重新下发的训练作业会以第一优先级进行排队。如果排队时间超过30分钟，训练任务会自动退出。该现象表明资源池规格任务紧张，训练作业无法正常启动，推荐您购买专属资源池补充计算节点。

如果您使用专属资源池创建训练作业，容错检查识别的故障节点会被剔除。系统自动补充健康的计算节点至专属资源池。（该功能即将上线）

容错检查详细介绍请参考：

1. **开启容错检查**
2. **检测项目与执行条件**
3. **触发容错环境检测达到的效果**

版权所有 © 华为技术有限公司

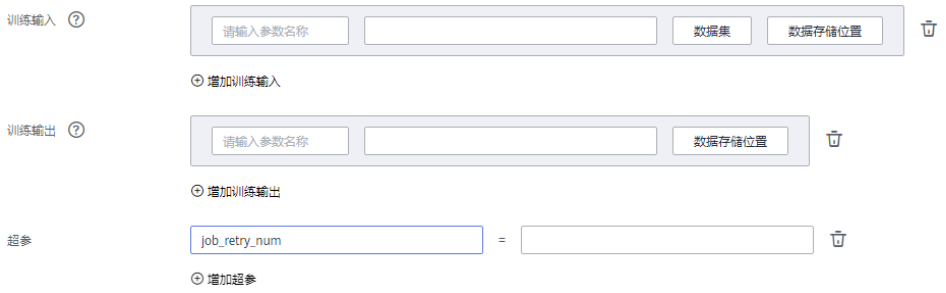
4. 环境预检查通过后，如果发生硬件故障会导致用户业务中断。您可以在训练中补充reload ckpt的代码逻辑，使能读取训练中断前保存的预训练模型。指导请参考**5.1.2 使用reload ckpt恢复中断的训练**。

开启容错检查

用户可以在创建训练作业时通过传入内置超参的方式开启容错检查。

通过添加“ job_retry_num”字段，value的范围可以设置为1~3的整数。value值表示最大允许重新下发作业的次数。如果不传则默认为0，表示不做重新下发作业，也不会启用环境检测。

- 使用控制台页面设置容错检查：用户可以在控制台页面通过设置超参的方式开启运维模式。



- 使用API接口设置容错检查：用户可以通过API接口的方式在创建训练接口的超参“paramters”数组中传入以下代码。

```
{
    "kind": "job",
    "metadata": {
        "name": "trainjob-swqwtd3g"
    },
    "algorithm": {
        "parameters": [
            {
                "name": "dist",
                "value": "True"
            },
            {
                "name": "job_retry_num",
                "value": "2"
            }
        ],
        "inputs": [
```


版权所有 © 华为技术有限公司

检测项目与执行条件

检测项目	item（日志关键字）	执行条件	检测成功要求
域名检测	dns	无	volcano容器的域名都解析成功（/etc/volcano下的.host文件中的域名解析成功）
磁盘空间-容器根目录	disk-size root	无	大于32GB
磁盘空间-/dev/shm目录	disk-size shm	无	大于1GB
磁盘空间-/cache 目录	disk-size cache	无	大于32GB
ulimit检查	ulimit	使用IB网络时	• max locked memory > 16000 • open files > 1000000 • stack size > 8000 • max user processes > 1000000
gpu检查	gpu-check	使用gpu，且使用v2训练引擎时（北京四暂无）	检测到gpu

触发容错环境检测达到的效果

- 容错检查正常通过时，会打印检测项目的日志，表示具体涉及的检查项目成功。您可以通过在日志中搜索“item”关键字查看。当容错检查正常通过时，可以减少运行报障问题。

```
[Modelarts Service Log][task] Detect
[Modelarts Service Log][INFO][detect] code: 0, message: ok, item: dns
[Modelarts Service Log][INFO][detect] code: 0, message: ok, item: disk-size root
[Modelarts Service Log][INFO][detect] code: 0, message: ok, item: disk-size shm
[Modelarts Service Log][INFO][detect] code: 0, message: ok, item: disk-size cache
[Modelarts Service Log][init] download code_url: s3://test-qianjiajun/tolerance_test/
```

- 容错检查失败时，会打印检查失败的日志。您可以通过在日志中搜索“item”关键字查看失败信息。

```
[Modelarts Service Log][init] running
[Modelarts Service Log][init] ip of the pod: 172.16.0.160
[Modelarts Service Log][INFO]detect item: dns; json:{"code": 0, "message": "ok"}
[Modelarts Service Log][INFO][task][detect] code: 0, message: ok, item: dns
[Modelarts Service Log][INFO]detect item: disk-size root; json:{"code": 13, "message": "the disk space of the path
\"/\" is 4892852224, which is less than 34359738368"}
[Modelarts Service Log][ERROR][task][detect] code: 13, message: the disk space of the path "/" is 4892852224, which is
less than 34359738368, item: disk-size root
[Modelarts Service Log][init] exiting...
[Modelarts Service Log][init] wait python processes exit...
```

如果作业重启次数没有达到设定的job_retry_num，则会自动做重新下发作业。您可以通过搜索“error,exiting”关键字查找作业重启失败结束的日志。

版权所有 © 华为技术有限公司

5.1.2 使用 reload ckpt 恢复中断的训练

用户训练业务运行时，如果硬件发生故障，则会导致训练任务失败。这些失败往往不是因为代码本身的错误，需要在失败后自动重启，即容错机制。开启容错机制可以参考**开启容错检查**指导。

在容错机制下，如果因为硬件问题导致训练作业重启，用户可以在代码中读取预训练模型，恢复至重启前的训练状态。用户需要在代码里加上reload ckpt的代码，使能读取训练中断前保存的预训练模型。以下介绍Pytorch和MindSpore的两个示例。

Pytorch 版 reload ckpt

1. 简单介绍一下pytorch模型保存的两种方式
   a. 仅保存模型参数
      ```
      state_dict = model.state_dict()
      torch.save(state_dict, path)
      ```
   b. 保存整个Module（不推荐）
      ```
      torch.save(model, path)
      ```
2. 保存模型的训练过程的产物

   将模型训练过程中的网络权重、优化器权重、以及epoch进行保存，便于中断后继续训练恢复。

   ```
   checkpoint = {
           "net": model.state_dict(),
           "optimizer": optimizer.state_dict(),
           "epoch": epoch
   }
   if not os.path.isdir('model_save_dir'):
       os.makedirs('model_save_dir')
   torch.save(checkpoint,'model_save_dir/ckpt_{}.pth'.format(str(epoch)))
   ```
3. 详细代码

```
# 判断输出obs路径中是否有模型文件。若无文件则默认从头训练，若有模型文件，则加载epoch值最大的ckpt文件，当做预训练模型
if mox.file.list_directory(args.obs_resume_model):
    print('> load last ckpt and continue training!!')
    last_ckpt = sorted([file for file in mox.file.list_directory(args.obs_resume_model) if file.endswith(".pth")])[-1]
    local_ckpt_file = os.path.join(args.resume_model, last_ckpt)
    print('last_ckpt:', last_ckpt)
    # 云上环境需要将obs中保存的预训练模型文件拷贝至容器内，供后续操作
    mox.file.copy(os.path.join(args.obs_resume_model, last_ckpt), local_ckpt_file)
    # 加载断点
    checkpoint = torch.load(local_ckpt_file)
    # 加载模型可学习参数
    model.load_state_dict(checkpoint['net'])
    # 加载优化器参数
    optimizer.load_state_dict(checkpoint['optimizer'])
    # 获取保存的epoch，模型会在此epoch的基础上继续训练
    start_epoch = checkpoint['epoch']
start = datetime.now()
total_step = len(train_loader)
for epoch in range(start_epoch + 1, args.epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.cuda(non_blocking=True)
        labels = labels.cuda(non_blocking=True)
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```


版权所有 © 华为技术有限公司

```
    ...

# 保存模型训练过程中的网络权重、优化器权重、以及epoch
checkpoint = {
      "net": model.state_dict(),
      "optimizer": optimizer.state_dict(),
      "epoch": epoch
    }
if not os.path.isdir(args.resume_model):
    os.makedirs(args.resume_model)
    torch.save(checkpoint, os.path.join(args.resume_model, 'ckpt_best_{}.pth'.format(epoch)))
```

MindSpore 版 reload ckpt

```
# 初始定义的网络、损失函数及优化器
net = resnet50(args_opt.batch_size, args_opt.num_classes)
ls = SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
opt = Momentum(filter(lambda x: x.requires_grad, net.get_parameters()), 0.01, 0.9)
# 首次训练的epoch初始值，mindspore1.3及以后版本会支持定义epoch_size初始值
# cur_epoch_num = 0
# 判断输出obs路径中是否有模型文件。若无文件则默认从头训练，若有模型文件，则加载epoch值最大的ckpt文件，当做预训练模型
if mox.file.list_directory(args_opt.obs_train_url):
    last_ckpt = sorted([file for file in mox.file.list_directory(args_opt.obs_train_url) if file.endswith(".ckpt")])[-1]
    print('last_ckpt:', last_ckpt)
    last_ckpt_file = os.path.join(args_opt.train_url, last_ckpt)
    mox.file.copy(os.path.join(args_opt.obs_train_url, last_ckpt), last_ckpt_file)
    # 加载断点
    param_dict = load_checkpoint(last_ckpt_file)
    print('> load last ckpt and continue training!!')
    # 加载模型参数到net
    load_param_into_net(net, param_dict)
    # 加载模型参数到opt
    load_param_into_net(opt, param_dict)

    # 获取保存的epoch值，模型会在此epoch的基础上继续训练,此参数在mindspore1.3及以后版本会支持
    # if param_dict.get("epoch_num"):
    #     cur_epoch_num = int(param_dict["epoch_num"].data.asnumpy())
model = Model(net, loss_fn=ls, optimizer=opt, metrics={'acc'})
# as for train, users could use model.train
if args_opt.do_train:
    dataset = create_dataset()
    batch_num = dataset.get_dataset_size()
    config_ck = CheckpointConfig(save_checkpoint_steps=batch_num,
                                 keep_checkpoint_max=35)
    # append_info=[{"epoch_num": cur_epoch_num}],mindspore1.3及以后版本会支持append_info参数，保存当前时刻的epoch值
    ckpoint_cb = ModelCheckpoint(prefix="train_resnet_cifar10",
                                 directory=args_opt.train_url,
                                 config=config_ck)
    loss_cb = LossMonitor()
    model.train(epoch_size, dataset, callbacks=[ckpoint_cb, loss_cb])
    # model.train(epoch_size-cur_epoch_num, dataset, callbacks=[ckpoint_cb, loss_cb])，mindspore1.3及以后版本支持从断点恢复训练
```

5.1.3 故障临终遗言

使用场景

随着模型规模和数据集的急剧增长，需要利用大规模的训练集训练大规模的神经网络。在大规模集群分布式训练时，会遇到集群中某个芯片、某台服务器故障，导致分布式训练任务失败。临终遗言是指中断的训练任务支持自动恢复，并可以在上一次训练中断的基础上继续训练，而不用从头开始。

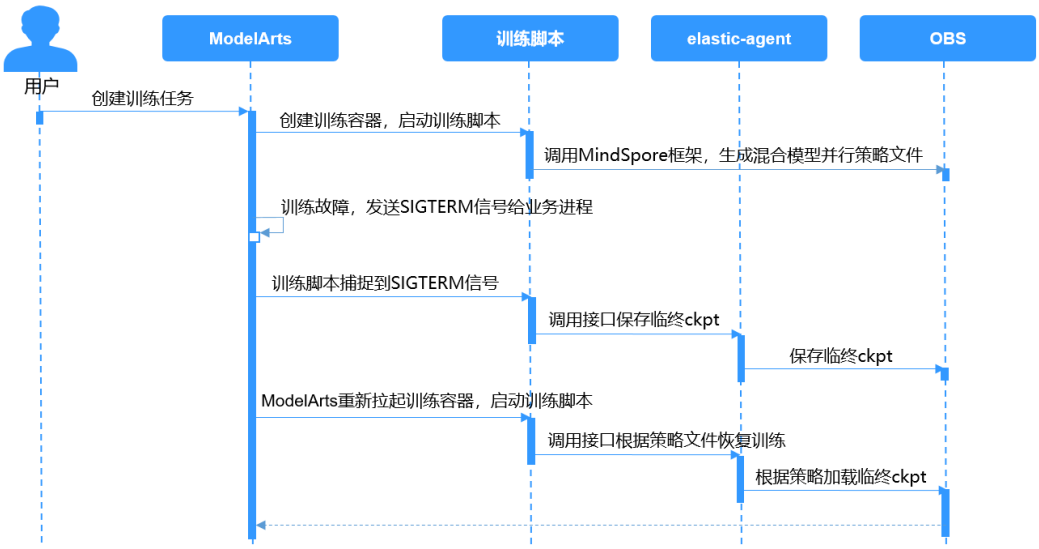
 版权所有 © 华为技术有限公司

约束限制

表 5-1 约束限制

服务器	Atlas 800训练服务器（型号：9000/9010）/Atlas 900 PoD
训练框架	MindSpore

特性原理



临终遗言处理流程如下：

1. 用户在ModelArts管理控制台创建训练任务。
2. 训练引擎创建训练容器，启动训练脚本。
3. 训练脚本启动后，调用MindSpore框架，生成混合并行策略文件strategy.proto，该文件记录了混合并行场景下，算子在NPU卡上的分布情况。
4. 训练出现故障后，ModelArts训练组件对当前业务进程发送SIGTERM信号。
5. 训练脚本捕获到SIGTERM信号，调用elastic-agent模块，该模块会调用mindspore框架，生成临终ckpt。
6. ModelArts训练组件重新拉起训练容器，启动训练脚本。
7. 训练脚本调用elastic-agent模块，该模块根据configmap中故障NPU信息和strategy.proto文件生成策略恢复文件。
8. 训练脚本根据策略恢复文件，加载临终ckpt进行续训练。

在数据并行场景下，也是类似的流程，只是更为简单，无需生成并行策略文件和策略恢复文件，只要保存和加载临终ckpt文件即可。

特性使用操作

1. **安装临终遗言二进制包**

   通过ma_pre_start.sh安装whl包。

   ```
   echo "[ma-pre-start] Enter the input directory"cd /home/ma-user/modelarts/inputs/data_url_0/echo "[ma-pre-start] Start to install mindx-elastic 0.0.1版本"export PATH=/home/ma-user/anaconda/bin:
   ```


版权所有 © 华为技术有限公司

```
$PATHpip install ./mindx_elastic-0.0.1-py3-none-any.whl echo "[ma-pre-start] Clean run
package"sudo rm -rf ./script ./*.run ./run_package *.whl echo "[ma-pre-start] Set ENV"export
GLOG_v=2    # 当前使用诊断模式需要用户手动设置成INFO日志级别 echo "[ma-pre-start] End"
```

2. **创建训练任务**
   - 约束：MindSpore版本要求1.6.0及以上。
   - 修改样例代码，增加如下内容：

```
# 载入依赖接口
from mindx_elastic.terminating_message import ExceptionCheckpoint
...

if args_opt.do_train:
dataset = create_dataset()
loss_cb = LossMonitor()
cb = [loss_cb]
if int(os.getenv('RANK_ID')) == 0:
batch_num = dataset.get_dataset_size()
# 开启临终遗言保存
config_ck = CheckpointConfig(save_checkpoint_steps=batch_num,
keep_checkpoint_max=35,
async_save=True,
append_info=[{"epoch_num": cur_epoch_num}],
exception_save=True)

ckpoint_cb = ModelCheckpoint(prefix="train_resnet_cifar10",
directory=args_opt.train_url,
config=config_ck)
# 定义临终遗言ckpt保存callback
ckpoint_exp = ExceptionCheckpoint(
prefix="train_resnet_cifar10",
directory=args_opt.train_url,
config=config_ck)
# 添加临终遗言ckpt保存callback
cb += [ckpoint_cb, ckpoint_exp]
```

5.2 训练模式选择

针对MindSpore类引擎，ModelArts提供训练模式选择，支持用户根据实际场景获取不同的诊断信息。

在训练作业创建页面，支持普通模式、高性能模式和故障诊断模式，默认设置为普通模式。普通模式的调测信息可参考**查看训练日志**。

针对于新增的两种模式，推荐以下两种场景使用：

- 高性能模式：最小化调测信息，最大程度地提升运行速度，适合于网络稳定并追求高性能的场景。
- 故障诊断模式：收集更多的信息用于定位，适合于执行出现问题需要收集故障信息进行定位的场景。此模式提供故障诊断，用户可以根据实际需求选择诊断类别。

图 5-5 模式选择

各模式获取的调测信息见下表。

版权所有 © 华为技术有限公司

表 5-2 MindSpore 引擎各模式的调测信息

调测信息	普通模式	高性能模式	故障诊断模式	说明
MindSpore框架日志级别	Info级别	error级别	Info级别	MindSpore框架运行时日志。
RDR(Running Data Recorder)	关闭	关闭	开启	出现运行异常会自动地导出MindSpore中预先记录的数据以辅助定位运行异常的原因。不同的运行异常将会导出不同的数据。 RDR详细的介绍请参考**MindSpore官网说明**。
analyze_fail.dat	默认提供，上传至训练作业日志路径中			图编译失败自动导出故障信息，用于infer过程分析。
dump数据	默认提供，上传至训练作业日志路径中			后端执行期异常触发dump数据。

在故障诊断模式下，开启故障诊断功能后，支持用户查看以下故障诊断数据。以下数据存储至训练日志路径的OBS目录下。

故障诊断模式的训练输出日志文件说明：

```
{obs-log-path}/
    modelarts-job-{job-id}-worker-{index}.log # 打屏日志（汇总）
    modelarts-job-{job-id}-proc-rank-{rank-id}-device-{device-id}.txt # 每个 device 的打屏日志
    modelarts-job-{job-id}/
        ascend/
            npu_collect/rank_{id}/   # TFAdapter DUMP GRAPH 与 GE DUMP GRAPH 的输出路径，仅在使用TensorFlow框架时生成
            process_log/rank_{id}/   # Plog 日志路径
            msnpureport/{task-index}/  #msnpureport工具执行日志，用户无需关注
        mindspore/
            log/  # MindSpore 框架日志与 MindSpore 故障诊断数据
```

表 5-3 故障诊断数据一览表（MindSpore）

故障诊断分类	故障诊断内容
CANN框架日志和故障诊断数据	HOST侧的INFO及INFO以上级别日志，包括HOST侧CANN软件栈日志、HOST侧驱动日志文件等。
MindSpore框架日志和故障诊断数据	MindSpore框架生成的日志，INFO及INFO以上级别日志。
	RDR(Running Data Recorder)文件。 出现运行异常会自动地导出MindSpore中预先记录的数据以辅助定位运行异常的原因。不同的运行异常将会导出不同的数据。
	analyze_fail.dat，图编译失败自动导出故障信息，用于infer过程分析。

版权所有 © 华为技术有限公司

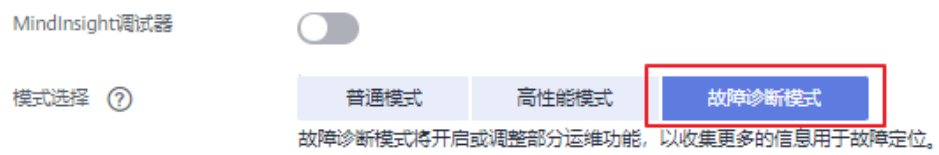
故障诊断分类	故障诊断内容
	dump数据，后端执行期异常触发dump数据。

在创建训练作业页面，选择算法为MindSpore，资源类型为Ascend，可以开启故障诊断模式。

图 5-6 选择算法

图 5-7 选择资源类型

图 5-8 开启故障诊断



版权所有 © 华为技术有限公司

6 模型训练可视化

6.1 可视化训练作业介绍

ModelArts支持在新版开发环境中开启TensorBoard和MindInsight可视化工具。在开发环境中通过小数据集训练调试算法，主要目的是验证算法收敛性、检查是否有训练过程中的问题，方便用户调测。

ModelArts可视化作业支持创建TensorBoard类型和MindInsight两种类型。

TensorBoard和MindInsight能够有效地展示训练作业在运行过程中的变化趋势以及训练中使用到的数据信息。

- TensorBoard

  TensorBoard是一个可视化工具，能够有效地展示TensorFlow在运行过程中的计算图、各种指标随着时间的变化趋势以及训练中使用到的数据信息。TensorBoard相关概念请参考**TensorBoard官网**。

  TensorBoard可视化训练作业，当前仅支持基于TensorFlow2.1、Pytorch1.4/1.8以上版本镜像，CPU/GPU规格的资源类型。请根据实际局点支持的镜像和资源规格选择使用。

- MindInsight

  MindInsight能可视化展现出训练过程中的标量、图像、计算图以及模型超参等信息，同时提供训练看板、模型溯源、数据溯源、性能调试等功能，帮助您在更高效地训练调试模型。MindInsight当前支持基于MindSpore引擎的训练作业。MindInsight相关概念请参考**MindSpore官网**。

  MindInsight可视化训练作业，当前仅支持的镜像有

  mindspore1.2.0版本，CPU/GPU规格的资源类型。

  mindspore1.5.x以上版本，Ascend规格的资源类型。

您可以使用模型训练时产生的Summary文件在开发环境Notebook中创建可视化作业。

 版权所有 © 华为技术有限公司

6.2 MindInsight 可视化作业

ModelArts支持在新版开发环境中开启MindInsight可视化工具。在开发环境中通过小数据集训练调试算法，主要目的是验证算法收敛性、检查是否有训练过程中的问题，方便用户调测。

MindInsight能可视化展现出训练过程中的标量、图像、计算图以及模型超参等信息，同时提供训练看板、模型溯源、数据溯源、性能调试等功能，帮助您在更高效地训练调试模型。MindInsight当前支持基于MindSpore引擎的训练作业。MindInsight相关概念请参考**MindSpore官网**。

MindSpore支持将数据信息保存到Summary日志文件中，并通过可视化界面MindInsight进行展示。

前提条件

使用MindSpore引擎编写训练脚本时，为了保证训练结果中输出Summary文件，您需要在脚本中添加收集Summary相关代码。

将数据记录到Summary日志文件中的具体方式请参考**收集Summary数据**。

注意事项

- 请将模型训练时产生的Summary文件上传到OBS并行文件系统，并确保OBS并行文件系统与ModelArts在同一区域。
- 在开发环境跑训练任务，在开发环境使用MindInsight，要求先启动MIndInsight，后启动训练进程。
- 仅支持单机单卡训练。
- 运行中的可视化作业不单独计费，当停止Notebook实例时，计费停止。
- 数据存放在OBS中，由OBS单独收费。任务完成后请及时停止Notebook实例，清理OBS数据，避免产生不必要的费用。

在开发环境中创建 MidnightInsight 可视化作业流程

Step1 创建开发环境并在线打开

Step2 上传Summary数据

Step3 启动MindInsight

Step4 查看训练看板中的可视化数据

Step1 创建开发环境并在线打开

在ModelArts控制台，进入“开发环境> Notebook”新版页面，创建MindSpore引擎的开发环境实例。创建成功后，单击开发环境实例操作栏右侧的“打开 ”，在线打开运行中的开发环境。

图 6-1 支持 MindInsight 的 MindSpore 引擎

版权所有 © 华为技术有限公司

Step2 上传 Summary 数据

在开发环境中使用MindInsight可视化功能，需要用到Summary数据。

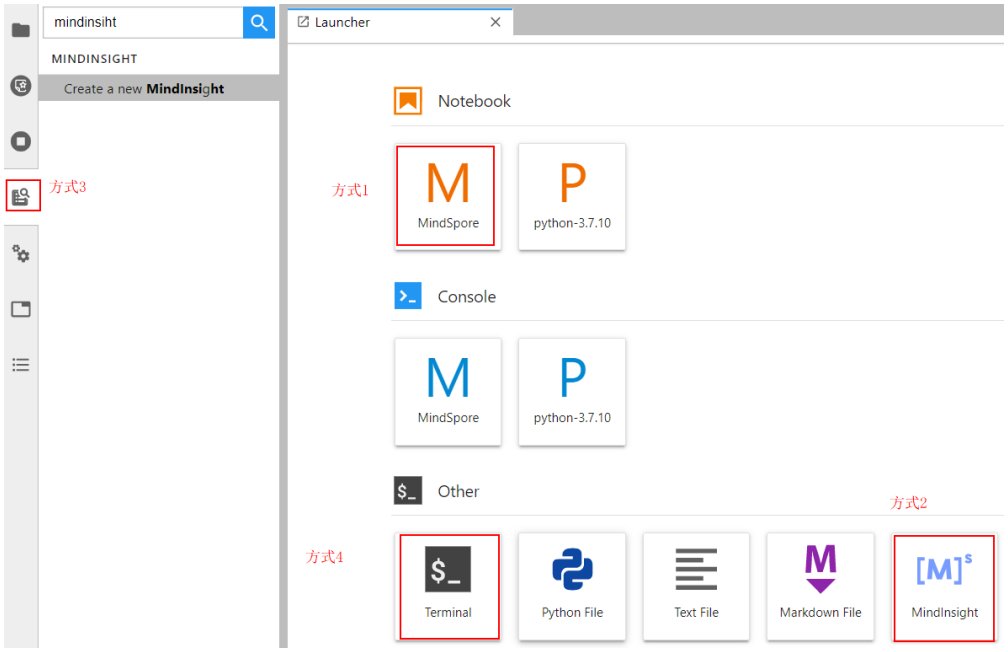
Summary数据可以直接传到开发环境的这个路径下/home/ma-user/work/，也可以放到OBS并行文件系统中。

建议将Summary数据存放在OBS并行文件系统中，在开发环境中启动MindInsight时，开发环境会自动挂载OBS并行文件系统目录读取Summary数据。

Step3 启动 MindInsight

在开发环境的JupyterLab中打开MindInsight有多种方法。可根据使用习惯选择。

图 6-2 JupyterLab 中打开 MindInsight 的方法



方式1：

1. 单击入口1 

，进入JupyterLab开发环境，并自动创建.ipynb文件。
2. 在对话框中输入MindInsight相应命令，即可展示界面。

```
%reload_ext mindinsight
%mindinsight --port {PORT} --summary-base-dir {SUMMARY_BASE_DIR}
```

参数解释：

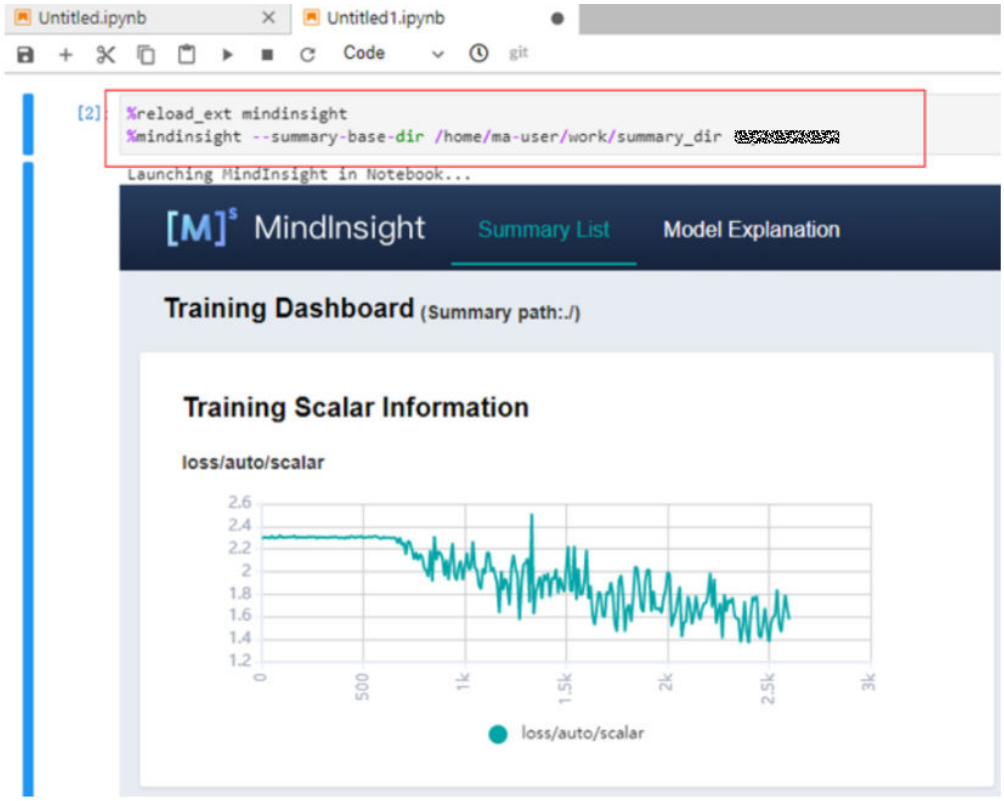
- --port {PORT}：指定Web可视化服务端口。可以不设置，默认使用8080端口。如果8080端口被占用了，需要在1~65535任意指定一个端口。
- --summary-base-dir {SUMMARY_BASE_DIR}：表示数据在开发环境中的存储路径。

版权所有 © 华为技术有限公司

- OBS并行文件系统桶的路径：obs://xxx/
- 开发环境本地路径：“./work/xxx”（相对路径）或/home/ma-user/work/xxx（绝对路径）

```
例如：
#Summary数据如果是存在OBS并行文件系统中，执行下面这条命令，开发环境会自动挂载该OBS并行文件系统路径并读取数据
%mindinsight --summary-base-dir obs://xxx/
或者
#Summary数据如果是在开发环境的这个路径下/home/ma-user/work/，执行下面这条命令
%mindinsight --summary-base-dir /home/ma-user/work/xxx
```

图 6-3 MindInsight 界面（1）



方式2：

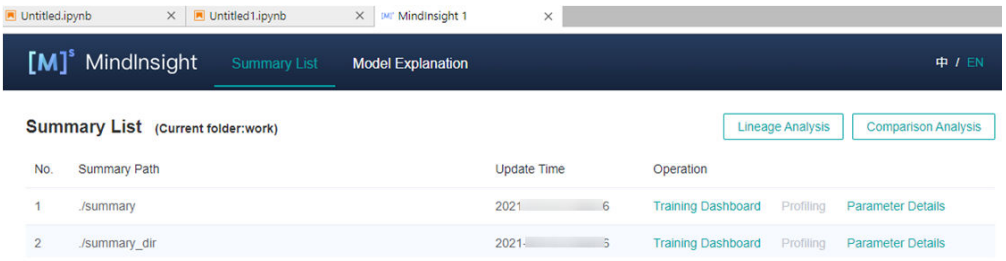
单击入口2，直接进入MindInsight可视化界面。

默认读取路径/home/ma-user/work/

当存在两个以及以上工程的log时，界面如下。通过Runs下选择查看相对应的log。

版权所有 © 华为技术有限公司

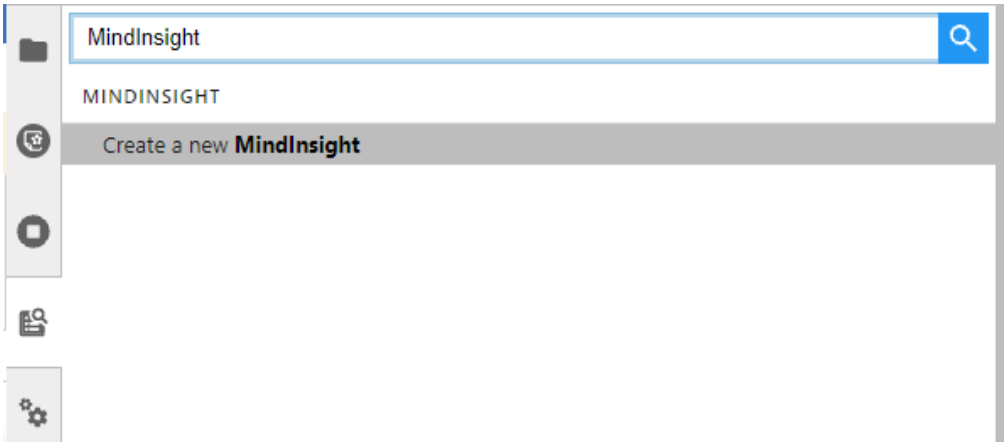
图 6-4 MindInsight 界面（2）



方式3：

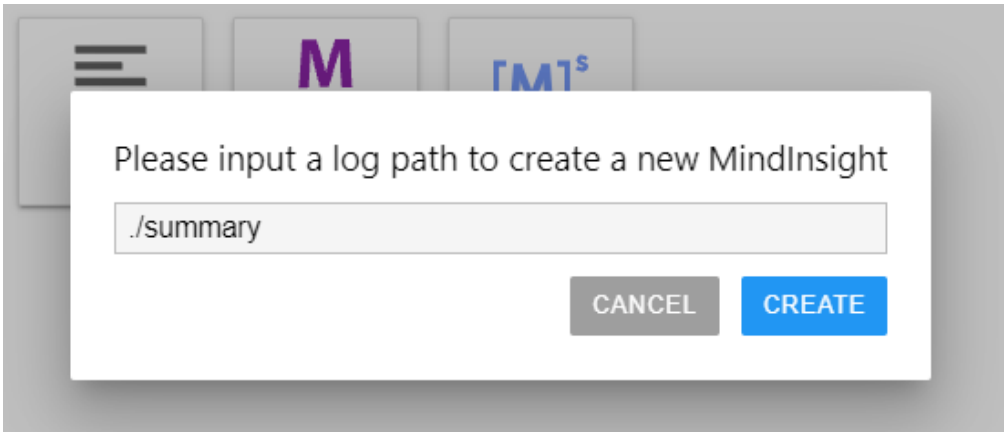
1. 单击入口3  ，在搜索框中输入“MindInsight”，再单击“Create a new MindInsight”。

   图 6-5 Create a new MindInsight

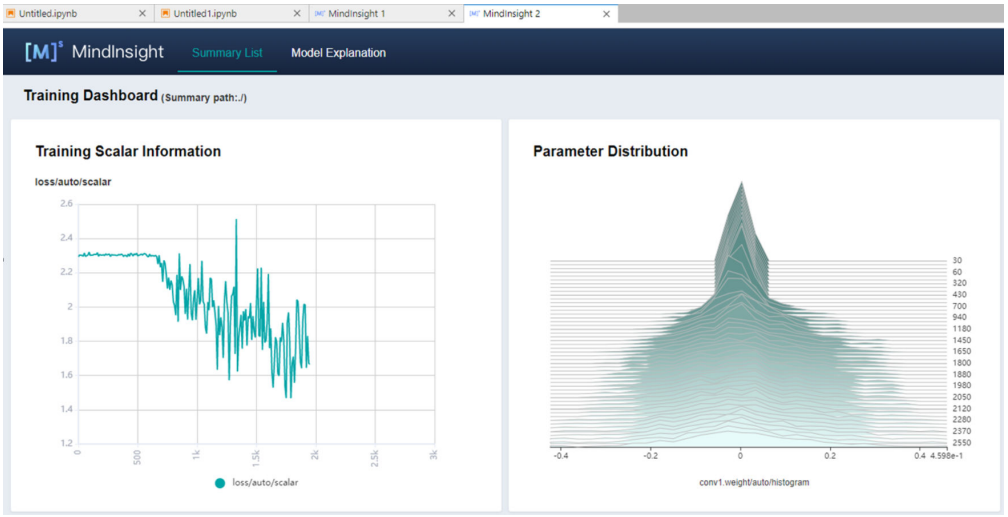
2. 填写需要查看的Summary数据路径，或者OBS并行文件系统桶的路径，单击CREATE。
   - 开发环境本地路径：./summary（相对路径）或/home/ma-user/work/summary（绝对路径）
   - OBS并行文件系统的路径：obs://xxx/

   图 6-6 输入 Summary 数据路径

版权所有 © 华为技术有限公司

图 6-7 MindInsight 界面（3）



说明

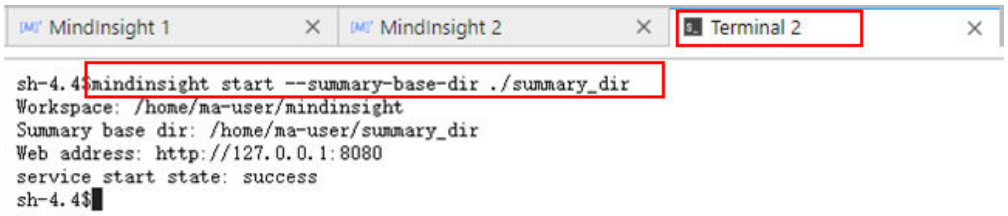
方式2及方式3最多可以启动10个MindInsight实例。

方式4：

单击入口4 Terminal ，输入并执行以下命令，但启动后不能显示UI界面。

```
mindinsight start --summary-base-dir ./summary_dir
```

图 6-8 Terminal 方式打开 MindInsight



Step4 查看训练看板中的可视化数据

训练看板是MindInsight的可视化组件的重要组成部分，而训练看板的标签包含：标量可视化、参数分布图可视化、计算图可视化、数据图可视化、图像可视化和张量可视化等。

更多功能介绍请参见MindSpore官网资料：**查看训练看板中可视的数据**。

相关操作

关闭MindInsight方式如下：

 版权所有 © 华为技术有限公司

- 方式1：在云上开发环境JupyterLab中的.ipynb文件窗口中输入命令，关闭MindInsight。端口号在**启动MindInsight**中设置，默认使用8080，需要替换为实际开启MindInsight时的端口。

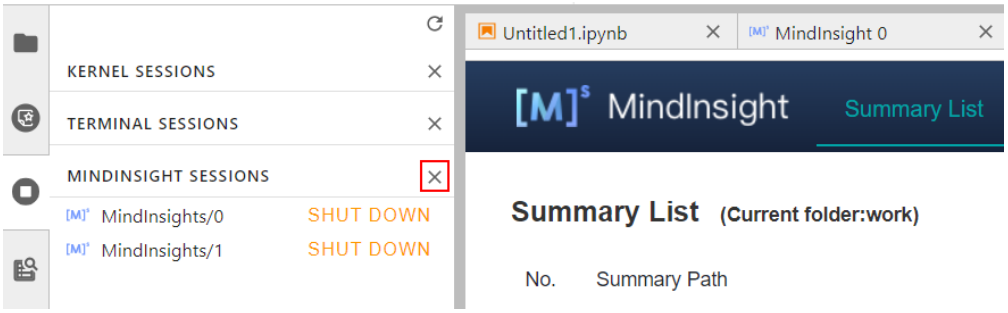
```
!mindinsight stop --port 8080
```

- 方式2：单击下方  按钮进入MindInsight实例管理界面，该界面记录了所有启动的MindInsight实例，单击对应实例后面的SHUT DOWN即可停止该实例。

图 6-9 单击 SHUT DOWN 停止实例

- 方式3：单击下方红框中的的  按钮可以关闭所有启动的MindInsight实例。

图 6-10 关闭所有启动的 MindInsight 实例



- 方式4（不推荐）：直接在JupyterLab中上关闭MindInsight窗口，此方式仅是关闭MindInsight可视化窗口，并未关闭后台。

6.3 TensorBoard 可视化作业

ModelArts支持在开发环境中开启TensorBoard可视化工具。TensorBoard是TensorFlow的可视化工具包，提供机器学习实验所需的可视化功能和工具。

TensorBoard能够有效地展示TensorFlow在运行过程中的计算图、各种指标随着时间的变化趋势以及训练中使用到的数据信息。TensorBoard当前只支持基于TensorFlow2.1以上版本引擎的训练作业。TensorBoard相关概念请参考**TensorBoard官网**。

前提条件

为了保证训练结果中输出Summary文件，在编写训练脚本时，您需要在脚本中添加收集Summary相关代码。

TensorFlow引擎的训练脚本中添加Summary代码，具体方式请参见**TensorFlow官方网站**。

版权所有 © 华为技术有限公司

注意事项

- 请将模型训练时产生的Summary文件上传到OBS并行文件系统，并确保OBS并行文件系统与ModelArts在同一区域。
- 运行中的可视化作业不单独计费，当停止Notebook实例时，计费停止。
- 数据存放在OBS中，由OBS单独收费。任务完成后请及时停止Notebook实例，清理OBS数据，避免产生不必要的费用。

在开发环境中创建 TensorBoard 可视化作业流程

Step1 创建开发环境并在线打开

Step2 上传Summary数据

Step3 启动TensorBoard

Step4 查看训练看板中的可视化数据

Step1 创建开发环境并在线打开

在ModelArts控制台，进入“开发环境> Notebook”新版页面，创建TensorFlow或者PyTorch镜像的开发环境实例。创建成功后，单击开发环境实例操作栏右侧的“打开 > 打开Notebook”，在线打开运行中的开发环境。

开发环境的PyTorch镜像中也预置了TensorFlow引擎，所以支持创建TensorBoard可视化作业。

图 6-11 支持 TensorBoard 的开发环境镜像

名称	描述
pytorch1.4-cuda10.1-cudnn7-ubuntu18.04	CPU、GPU通用算法开发和训练基础镜像，预置AI引擎PyTorch1.4
tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04	CPU and GPU general algorithm development and training, preconfigured

Step2 上传 Summary 数据

在开发环境中使用TensorBoard可视化功能，需要用到Summary数据。

Summary数据可以直接传到开发环境的这个路径下/home/ma-user/work/，也可以放到OBS并行文件系统中。

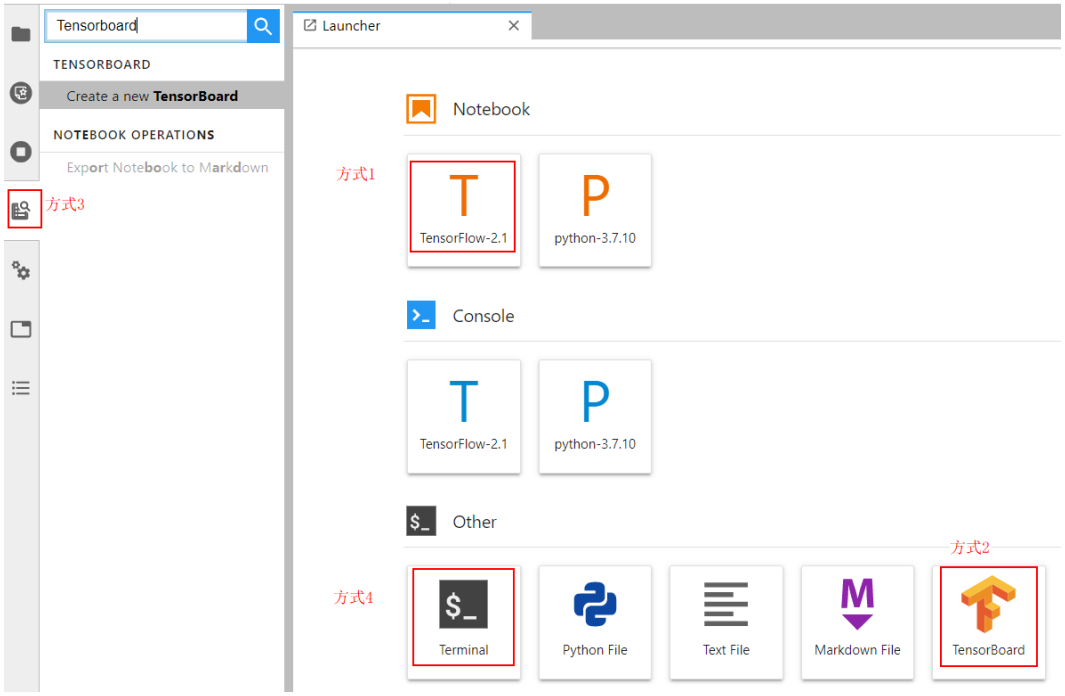
建议将Summary数据存放在OBS并行文件系统中，在开发环境中启动TensorBoard时，开发环境会自动挂载OBS并行文件系统目录读取Summary数据。

Step3 启动 TensorBoard

在开发环境的JupyterLab中打开TensorBoard有多种方法。可根据使用习惯选择。

版权所有 © 华为技术有限公司

图 6-12 JupyterLab 中打开 TensorBoard 的方法



方式1：

1. 单击入口1，进入JupyterLab开发环境中，并自动创建.ipynb文件。
2. 在对话框中输入TensorBoard相应命令，即可展示界面。

```
%reload_ext tensorboard
%tensorboard  --port {PORT} --logdir {BASE_DIR}
```

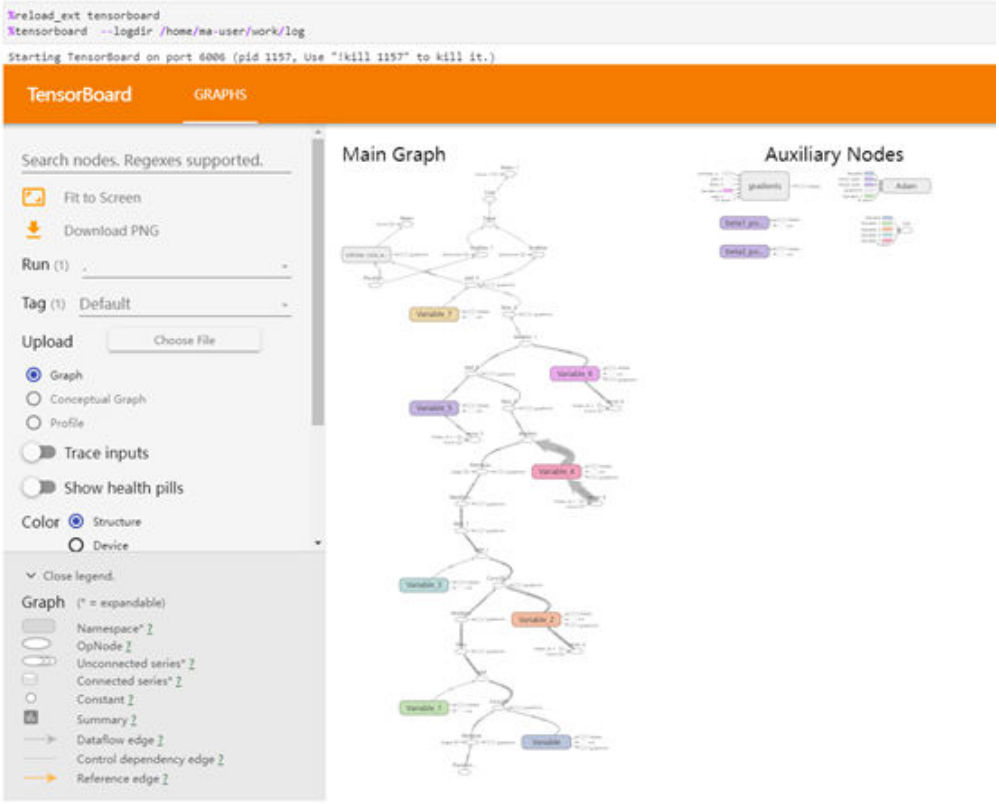
参数解释：

- --port {PORT}：指定Web可视化服务端口。可以不设置，默认使用8080端口。如果8080端口被占用了，需要在1~65535任意指定一个端口。
- --logdir {BASE_DIR}：表示数据在开发环境中的存储路径
  - OBS并行文件系统的路径：obs://xxx/
  - 开发环境本地路径：“./work/xxx”（相对路径）或/home/ma-user/work/xxx（绝对路径）

```
例如：
#Summary数据如果是存在OBS并行文件系统中，执行下面这条命令，开发环境会自动挂载该OBS并行文件系统路径并读取数据。
%tensorboard  --port {PORT} --logdir obs://xxx/
或者
#Summary数据如果是在开发环境的这个路径下/home/ma-user/work/，执行下面这条命令。
%tensorboard  --port {PORT} --logdir /home/ma-user/work/xxx
```


版权所有 © 华为技术有限公司

图 6-13 TensorBoard 界面（1）



方式2：

单击入口2 

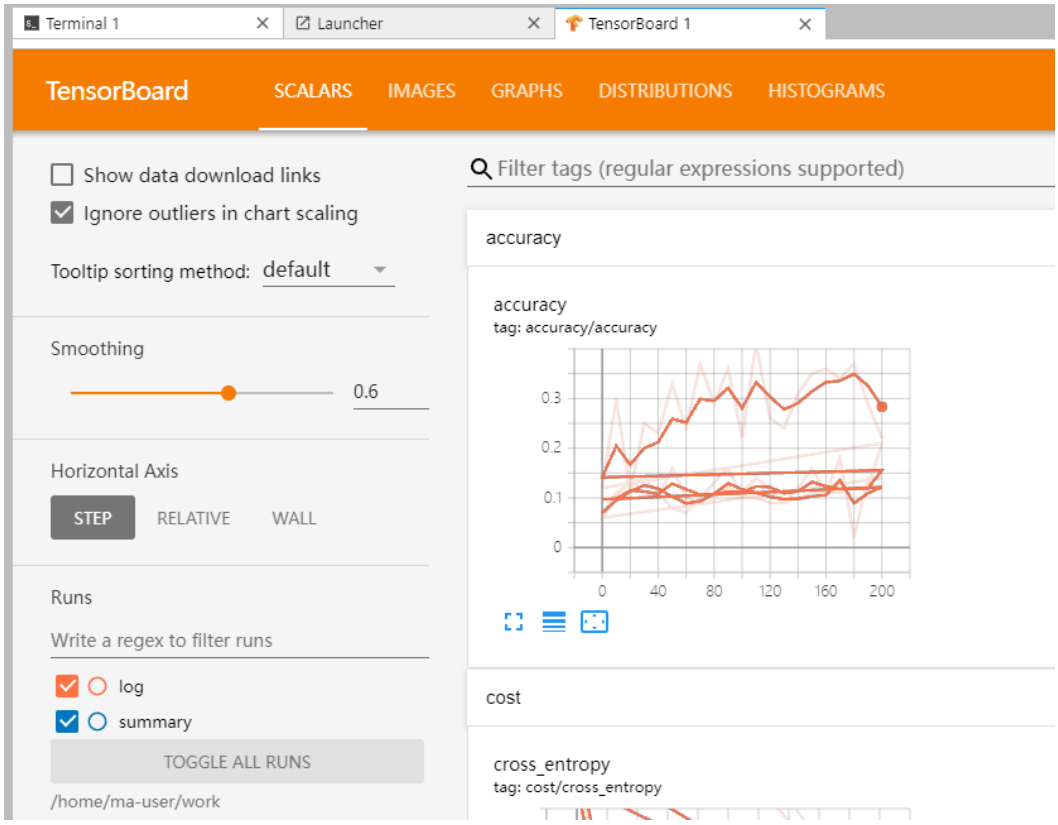
，直接进入TensorBoard可视化界面。

默认读取路径/home/ma-user/work/

当存在两个以及以上工程的log时，界面如下。可以在左侧Runs栏目下选择查看相对应的log。

 版权所有 © 华为技术有限公司

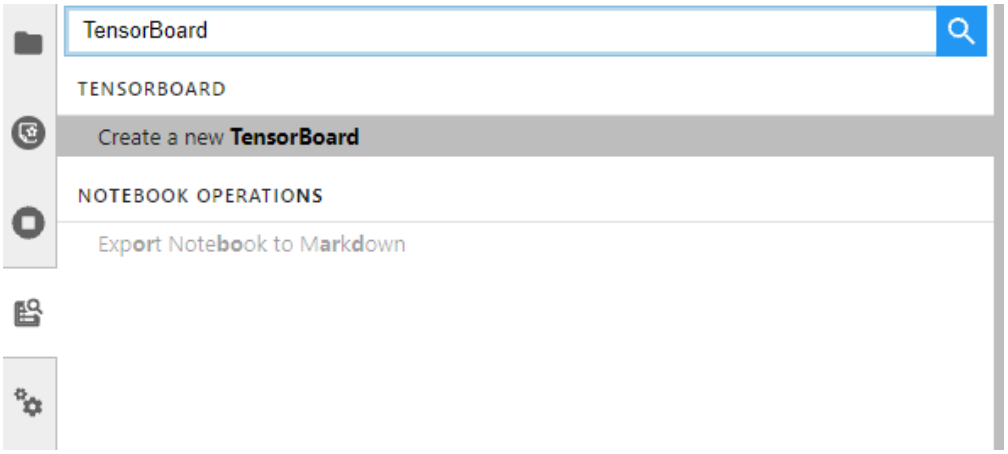
图 6-14 TensorBoard 界面（2）



方式3：

1. 单击入口3  ，在搜索框中输入“TensorBoard”，再单击“Create a new TensorBoard”。

   图 6-15 Create a new TensorBoard

2. 填写需要查看的Summary数据路径，或者OBS并行文件系统桶的路径。
   - 开发环境本地路径：./summary（相对路径）或/home/ma-user/work/summary（绝对路径）

 版权所有 © 华为技术有限公司

– OBS并行文件系统桶的路径：obs://xxx/

图 6-16 输入 Summary 数据路径

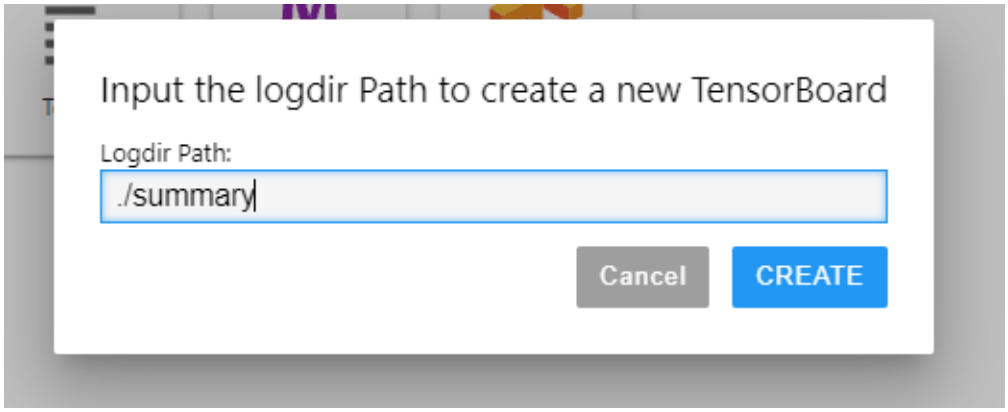
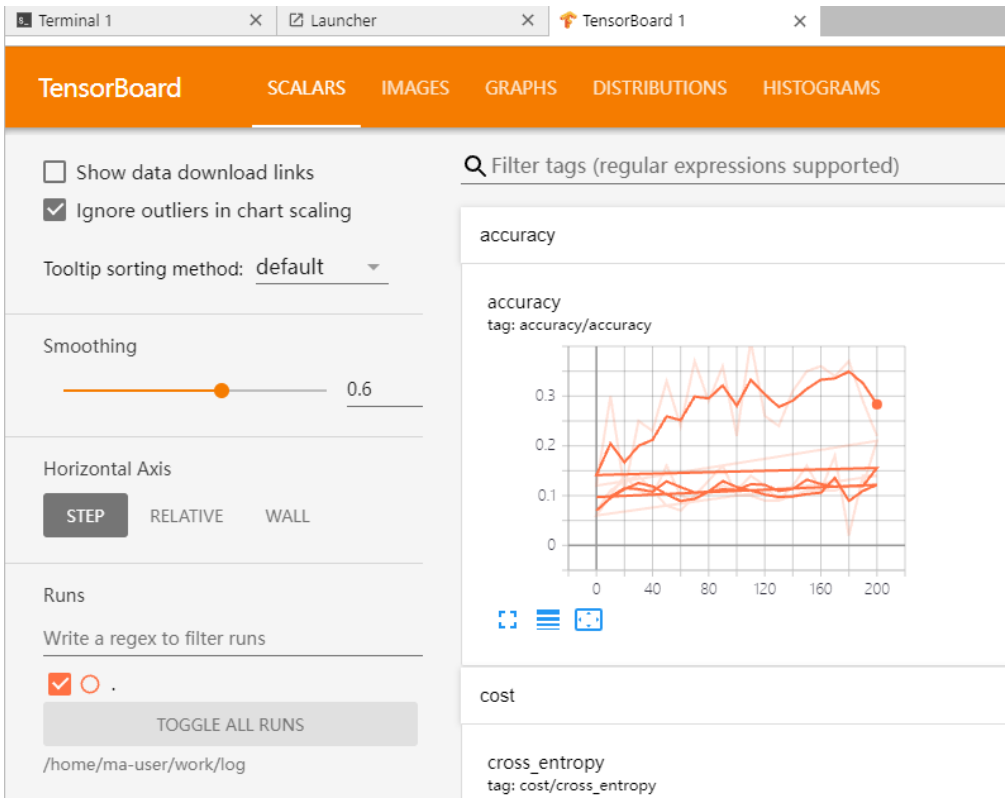


图 6-17 TensorBoard 界面（3）



方式4：

单击入口4 Terminal ，输入命令执行，但启动后不能显示UI界面。

```
tensorboard --logdir ./log
```

 版权所有 © 华为技术有限公司

图 6-18 Terminal 方式打开 TensorBoard

Step4 查看训练看板中的可视化数据

训练看板是TensorBoard的可视化组件的重要组成部分，而训练看板的标签包含：标量可视化、图像可视化和计算图可视化等。

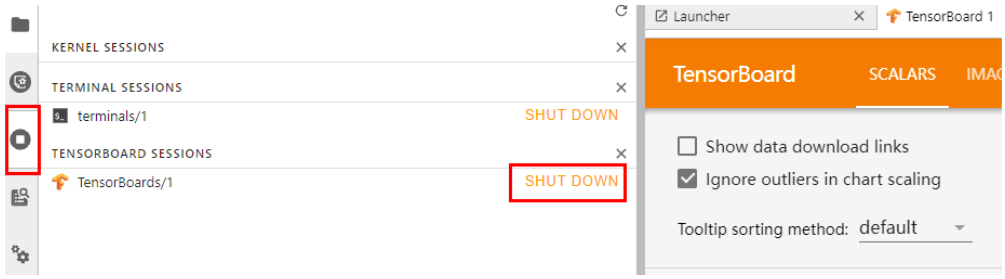
更多功能介绍请参见TensorBoard官网资料：**开始使用 TensorBoard | TensorFlow (google.cn)**。

相关操作

关闭TensorBoard方式如下：

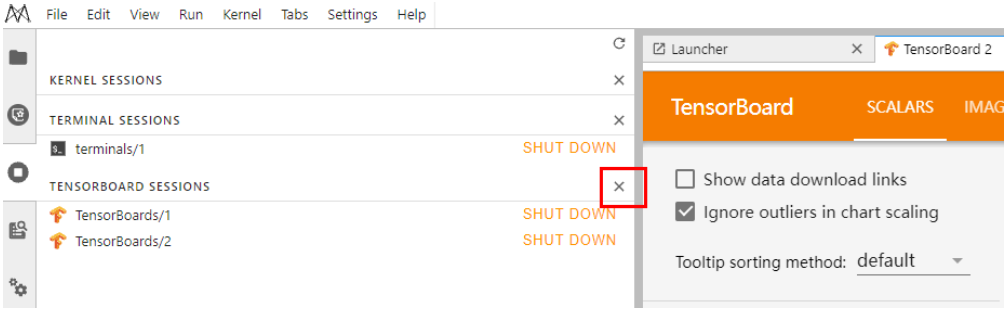
- 方式1：在云上开发环境JupyterLab中的.ipynb文件窗口中输入命令，关闭TensorBoard。PID在启动界面有提示或者通过 ps -ef | grep tensorboard 查看。
  ```
  !kill PID
  ```
- 方式2：单击下方  ，进入TensorBoard实例管理界面，该界面记录了所有启动的TensorBoard实例，单击对应实例后面的SHUT DOWN即可停止该实例。

  图 6-19 单击 SHUT DOWN 停该实例

- 方式3：单击下方红框中的的  按钮可以关闭所有启动的TensorBoard实例。

  图 6-20 关闭所有启动的 TensorBoard 实例

- 方式4（不推荐）：直接在JupyterLab中上关闭TensorBoard窗口，此方式仅关闭可视化窗口，并未关闭后台。

版权所有 © 华为技术有限公司

7 分布式训练

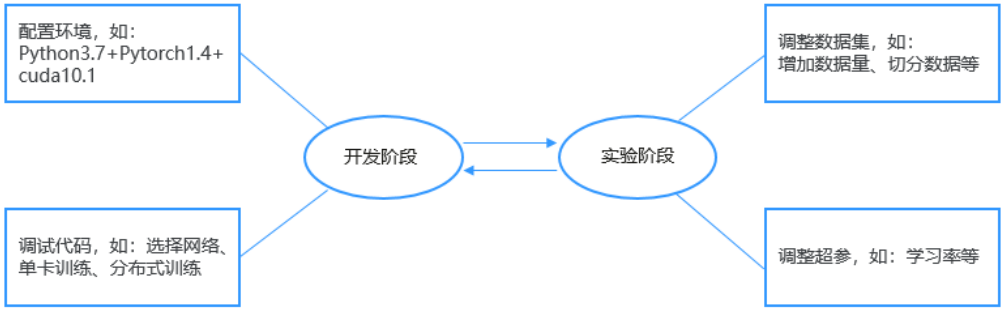
7.1 功能介绍

AI模型开发的过程，称之为Modeling，一般包含两个阶段：

- 开发阶段：准备并配置环境，调试代码，使代码能够开始进行深度学习训练，推荐在ModelArts开发环境中调试。
- 实验阶段：调整数据集、调整超参等，通过多轮实验，训练出理想的模型，推荐在ModelArts训练中进行实验。

两个过程可以相互转换。如开发阶段代码稳定后，则会进入实验阶段，通过不断尝试调整超参来迭代模型；或在实验阶段，有一个可以优化训练的性能的想法，则会回到开发阶段，重新优化代码

其部分过程可参考下图：



 版权所有 © 华为技术有限公司

ModelArts提供了如下能力：

- 丰富的官方预置镜像，满足用户的需求。
- 支持基于预置镜像自定义制作专属开发环境，并保存使用。
- 丰富的教程，帮助用户快速适配分布式训练，使用分布式训练极大减少训练时间。
- 分布式训练调测的能力，可在PyCharm/VSCode/JupyterLab等开发工具中调试分布式训练。

约束限制

- 开发环境指的是ModelArts提供的新版Notebook，不包括旧版Notebook。
- 总览页面打开的CodeLab不支持此项功能，但是如果用户在AI Gallery中打开了可用的案例，会自动跳转到CodeLab中，此时是可以使用这项功能的。
- 如果切换了Notebook的规格，那么只能在Notebook进行单机调测，不能进行分布式调测，也不能提交远程训练任务。
- 当前仅支持Pytorch和MindSpore AI框架，如果MindSpore要进行多机分布式训练调试，则每台机器上都必须有8张卡。
- 本文档提供的调测代码中涉及到的OBS路径，请用户替换为自己的实际OBS路径。
- 本文档提供的调测代码是以Pytorch为例编写的，不同的AI框架之间，整体流程是完全相同的，只需要修改个别的参数即可。

7.2 基于开发环境使用 SDK 调测

7.2.1 使用 SDK 调测单机训练作业

代码中涉及到的OBS路径，请用户替换为自己的实际OBS路径。

代码是以Pytorch为例编写的，不同的AI框架之间，整体流程是完全相同的，仅需修改**步骤6**和**步骤10**中的 framework_type参数值即可，例如：MindSpore框架，此处framework_type=Ascend-Powered-Engine。

步骤1 Session初始化。

华为云上代码如下：这里只列出最常用的一种方式，更多方式请参考《**Session鉴权章节**》

```
from modelarts.session import Session
session = Session()
```

步骤2 准备训练数据，这里支持三种形式，用户可根据自己的情况选择一种。

```
import os
from modelarts.train_params import InputData
base_bucket_path = "obs://modelarts-cn-north-4-a0de02a6/dis-train/cifar10/"
base_local_path = "/home/ma-user/work/cifar10/"

# 形式1，数据在OBS上，且是一个压缩文件
obs_path = os.path.join(base_bucket_path, "dataset-zip/dataset.zip")
data_local = os.path.join(base_local_path, "dataset/")
input_data = InputData(obs_path=obs_path, local_path=data_local, is_local_source=False)

# 形式2，数据在OBS上，且是一个目录
#obs_path = os.path.join(base_bucket_path, "dataset/")
```


版权所有 © 华为技术有限公司

```
#data_local = os.path.join(base_local_path, "dataset/")
#input_data = InputData(obs_path=obs_path, local_path=data_local, is_local_source=False)

# 形式3，数据在Notebook中，且是一个目录，一般是使用SFS挂载磁盘的场景
#obs_path = os.path.join(base_bucket_path, "dataset-local/")
#data_local = os.path.join(base_local_path, "dataset/")
#input_data = InputData(obs_path=obs_path, local_path=data_local, is_local_source=True)
```

参数解释：

- is_local_source：可选参数，默认为False，指定训练数据的保存位置。
  - False：训练数据保存在参数obs_path指定的位置中；
  - True：训练数据保存在notebook中，由local_path指定。
- obs_path：obs地址。根据is_local_source值的变化，有不同的含义。
  - is_local_source=False，此时是必选参数，代表训练数据位置，支持文件夹和压缩文件。
  - is_local_source=True，此时是可选参数。如果用户填写了该参数，则开始训练时会将Notebook中的训练数据压缩并上传到该位置，不可重复上传。如果第一次上传后，建议将is_local_source修改为False，obs_path指向刚才上传的压缩数据文件位置；如果用户没有填写，则不会进行压缩上传。
- local_path：必选参数，Notebook中的路径。用户的训练脚本需要从该目录中读取数据，完成训练任务。根据is_local_source值的变化，有不同的含义。
  - is_local_source=True，此时代表训练数据位置，仅支持文件夹。
  - is_local_source=False，训练过程中SDK会帮助用户将数据下载到该位置，如果训练数据是压缩文件，下载完成后会进行解压缩。

步骤3 准备训练脚本。

```
from modelarts.train_params import TrainingFiles
code_dir = os.path.join(base_local_path, "train/")

# 这里提前将训练脚本放在了obs中，实际上训练脚本可以是任何来源，只要能够放到Notebook里边就行

session.obs.download_file(os.path.join(base_bucket_path, "train/test-pytorch.py"), code_dir)
training_file = TrainingFiles(code_dir=code_dir, boot_file="test-pytorch.py", obs_path=base_bucket_path + 'train/')
```

参数解释：

- code_dir：必选参数，训练脚本所在的目录。在训练任务调测的情况下，必须是notebook中的目录，不能是OBS目录。
- boot_file：必选参数，训练启动文件路径，路径格式为基于code_dir目录的相对路径，如实例代码中boot_file的完整路径为/home/ma-user/work/cifar10/train/test-pytorch.py，这里就只需要填写test-pytorch.py。
- obs_path：可选参数，一个OBS目录。仅在本地单机调试时不需要该参数，提交远程训练时必选，会将训练脚本压缩并上传到该路径。

步骤4 准备训练输出，如果用户不需要将训练输出上传到OBS，可以省略这一步。

```
from modelarts.train_params import OutputData
output = OutputData(local_path=os.path.join(base_local_path, "output/"),
obs_path=os.path.join(base_bucket_path, 'output/'))
```

- local_path：必选参数，一个notebook中的路径，训练脚本需要将输出的模型或其他数据保存在该目录下。
- obs_path：必选参数，一个OBS目录。SDK会将local_path中的模型文件自动上传到这里。

 版权所有 © 华为技术有限公司

步骤5 查看训练支持的AI框架。

```
from modelarts.estimatorV2 import Estimator
Estimator.get_framework_list(session)
```

参数session即是第一步初始化的数据。如果用户知道要使用的AI框架，可以略过这一步。

步骤6 Estimator初始化。

```
from modelarts.estimatorV2 import Estimator
parameters = []
parameters.append({"name": "data_url", "value": data_local})
parameters.append({"name": "output_dir", "value": os.path.join(base_local_path, "output/")})
parameters.append({"name": "epoc_num", "value": 2})
estimator = Estimator(session=session,
                      training_files=training_file,
                      outputs=[output],
                      parameters=parameters,
                      framework_type='PyTorch',
                      train_instance_type='local',
                      train_instance_count=1,
                      script_interpreter="/home/ma-user/anaconda3/envs/PyTorch-1.4/bin/python",
                      log_url=base_bucket_path + 'log/',
                      job_description='This is a image net train job')
```

参数解释：

- session：必选参数，**步骤1**中初始化的参数。
- training_files：必选参数，**步骤3**中初始化的训练文件。
- outputs：可选参数，这里传入的是一个list，每个元素都是**步骤4**中初始化的训练输出。
- parameters：可选参数，一个list，每个元素都是一个字典，包含"name"和"value"两个字段，以"--name=value"的形式传递给训练启动文件。value支持字符串，整数，布尔等类型。对于布尔类型，建议用户在训练脚本中使用action='store_true'的形式来解析。
- framework_type：必选参数，训练作业使用的AI框架类型，可参考步骤5查询的返回结果。
- train_instance_type：必选参数，训练实例类型，这里指定'local'即为在notebook中进行训练。
- train_instance_count：必选参数，训练使用的worker个数，单机训练时为1，训练作业只在当前使用的notebook中运行。
- script_interpreter：可选参数，指定使用哪个python环境来执行训练任务，如果未指定，会默认使用当前的kernel。
- log_url：可选参数，一个OBS地址，训练过程中，SDK会自动将训练的日志上传到该位置。但是如果训练任务运行在Ascend上，则是必选参数。
- job_description：可选参数，训练任务的描述。

步骤7 开始训练。

```
estimator.fit(inputs=[input_data], job_name="cifar10-dis")
```

参数解释：

- inputs：可选参数，一个list，每个元素都是**步骤2**生成的实例。
- job_name：可选参数，训练任务名，便于区分和记忆。

本地单机调试训练任务开始后，SDK会依次帮助用户完成以下流程：

 版权所有 © 华为技术有限公司

1. 初始化训练作业，如果**步骤2**指定的训练数据在OBS上，这里会将数据下载到local_path中。
2. 执行训练任务，用户的训练代码需要将训练输出保存在**步骤4**中指定的local_path中。
3. 将训练任务得到的输出上传到**步骤4**指定的obs_path中，日志上传到第六步指定的log_url中。

同时，可以在任务名后增加时间后缀，区分不同的任务名称。

```
from datetime import datetime, timedelta
import time
base_name = "cifar10-dis"
job_name = base_name + '-' + (datetime.now() + timedelta(hours=8)).strftime('%Y%m%d-%H%M%S')
estimator.fit(inputs=[input_data], job_name=job_name)
```

步骤8 多次调试。

上一步执行过程中，训练脚本的日志会实时打印到控制台，如果用户的代码或者参数有误的话，可以很方便的看到。在Notebook中经过多次调试，得到想要的结果后，可以进行下一步。

步骤9 查询训练支持的计算节点类型和最大个数。

```
from modelarts.estimatorV2 import Estimator
Estimator.get_spec_list(session=session)
```

参数session即是**步骤1**初始化的数据。返回的是一个字典，其中flavors值是一个列表，描述了训练服务支持的所有规格的信息。每个元素中flavor_id是可直接用于远程训练任务的计算规格，max_num是该规格的最大节点数。如果用户知道要使用的计算规格，可以略过这一步。

步骤10 提交远程训练作业。

```
from modelarts.estimatorV2 import Estimator
parameters = []
parameters.append({"name": "data_url", "value": data_local})
parameters.append({"name": "output_dir", "value": os.path.join(base_local_path, "output/")})
parameters.append({"name": "epoch_num", "value": 2})
estimator = Estimator(session=session,
                      training_files=training_file,
                      outputs=[output],
                      parameters=parameters,
                      framework_type='PyTorch',
                      train_instance_type='modelarts.vm.cpu.8u',
                      train_instance_count=1,
                      script_interpreter="/home/ma-user/anaconda3/envs/PyTorch-1.4/bin/python",
                      log_url=base_bucket_path + 'log/',
                      job_description='This is a image net train job')
estimator.fit(inputs=[input_data], job_name="cifar10-dis")
```

在本地调测完成的基础上，只需要Estimator初始化时将参数train_instance_type修改为训练服务支持的规格即可（即第10步查询出来的flavor_id的值）。执行fit函数后，即可提交远程训练任务。

训练任务提交后，SDK会依次帮助用户完成以下流程：

1. 将训练脚本打包成zip文件，上传到**步骤3**中指定的obs_path中。
2. 当训练数据保存在Notebook中，则将其打包成zip文件并上传到指定的obs_path中。
3. 向ModelArts训练服务提交自定义镜像训练作业，使用的镜像为当前Notebook的镜像，这样保证了远程训练作业和在Notebook中的训练作业使用的运行环境一致。

 版权所有 © 华为技术有限公司

4. 训练任务得到的输出上传到**步骤4**指定的obs_path中，日志上传到这一步log_url指定的位置中。

   在这一步中需要注意的一个问题：

   如果用户在自己的训练脚本中要创建新的目录或文件，请在以下几种目录中创建：

   - /home/ma-user/work；
   - /cache；
   - inputs或者outputs中指定的local_path，如在步骤2中初始化InputData时，填写了local_path="/home/ma-user/work/xx/yy/"，则在该目录下也可以创建新目录或文件。

----结束

7.2.2 使用 SDK 调测多机分布式训练作业

代码中涉及到的OBS路径，请用户替换为自己的实际OBS路径。

代码是以Pytorch为例编写的，不同的AI框架之间，整体流程是完全相同的，仅需修改**步骤7**和**步骤11**中的 framework_type参数值即可，例如：MindSpore框架，此处framework_type=Ascend-Powered-Engine。

步骤1 Session初始化，与**使用SDK调测单机训练作业中的1**相同。

步骤2 准备训练数据，与**使用SDK调测单机训练作业中的2**相同，唯一的不同在于obs_path参数是必选的。

步骤3 准备训练脚本。

```
from modelarts.train_params import TrainingFiles
code_dir = os.path.join(base_local_path, "train/")

# 这里提前将训练脚本放在了obs中，实际上训练脚本可以是任何来源，只要能够放到Notebook里边就行

session.obs.download_file(os.path.join(base_bucket_path, "train/test-pytorch.py"), code_dir)
training_file = TrainingFiles(code_dir=code_dir, boot_file="test-pytorch.py", obs_path=base_bucket_path + 'train/')
```

参数解释：

- code_dir：必选参数，训练脚本所在的目录。在本地调试的情况下，必须是notebook目录，不能是OBS目录。
- boot_file：必选参数，训练启动文件，在code_dir目录下。
- obs_path：在多机分布式调测时必选参数，一个OBS目录，SDK会将notebook目录code_dir打包上传到obs_path中。

步骤4 准备训练输出，与单机训练作业调试**步骤4**相同。

步骤5 查看训练支持的AI框架，与单机训练作业调试**步骤5**相同。

步骤6 保存当前Notebook为新镜像，与单机训练作业调试**步骤9**相同。

步骤7 Estimator初始化。

```
from modelarts.estimatorV2 import Estimator
parameters = []
parameters.append({"name": "data_url", "value": data_local})
parameters.append({"name": "output_dir", "value": os.path.join(base_local_path, "output/")})
parameters.append({"name": "epoc_num", "value": 2})
# 启动脚本以parser.add_argument('--dist', action='store_true')的形式来接收该布尔类型的参数，如果要传入True，则以本行代码的形式传递；
```


版权所有 © 华为技术有限公司

```
parameters.append({"name": "dist"})
estimator = Estimator(session=session,
                      training_files=training_file,
                      outputs=[output],
                      parameters=parameters,
                      framework_type='PyTorch',
                      train_instance_type='local',
                      train_instance_count=2,
                      script_interpreter="/home/ma-user/anaconda3/envs/PyTorch-1.4/bin/python",
                      log_url=base_bucket_path + 'log/',
                      job_description='This is a image net train job')
```

参数解释：

- session：必选参数，**步骤1**中初始化的参数。
- training_files：必选参数，**步骤3**中初始化的训练文件。
- outputs：可选参数，这里传入的是一个list，每个元素都是步骤4中初始化的训练输出。
- parameters：可选参数，一个list，每个元素都是一个字典，包含"name"和"value"两个字段，以"–name=value"的形式传递给训练启动文件。value支持字符串，整数，布尔等类型。对于布尔类型，建议用户在训练脚本中使用action='store_true'的形式来解析。
- framework_type：必选参数，训练作业使用的AI框架类型，可参考步骤5的返回结果。
- train_instance_type：必选参数，训练实例类型，这里指定’local’即为本地训练。
- train_instance_count：必选参数，训练使用的worker个数，分布式调测时为2，训练开始时SDK还会再创建一个Notebook，与当前的Notebook组成一个2节点的分布式调试环境。
- script_interpreter：可选参数，指定使用哪个python环境来执行训练任务，如果未指定，会默认使用当前的kernel。
- log_url：可选参数，一个OBS地址，本地训练过程中，SDK会自动将训练的日志上传到该位置；但是如果训练任务运行在Ascend上，则是必选参数。
- job_description：可选参数，训练任务的描述。

步骤8 开始训练。

```
estimator.fit(inputs=[input_data], job_name="cifar10-dis")
```

参数解释：

- inputs：可选参数，一个list，每个元素都是步骤2中生成的实例；
- job_name：可选参数，训练任务名，便于区分和记忆。

本地分布式训练任务开始后，SDK会依次帮助用户完成以下流程：

1. 将训练脚本打包成zip文件，上传到**步骤3**中指定的obs_path中。
2. 如果训练数据保存在Notebook中，则将其打包成zip文件并上传到指定的obs_path中。
3. 创建一个附属Notebook，与当前使用的Notebook组成分布式训练的两个worker。
4. 初始化训练作业，将数据下载到local_path中。
5. 执行训练任务，用户的代码需要将训练输出保存在**步骤4**指定的local_path中。

 版权所有 © 华为技术有限公司

6. 将训练任务得到的输出上传到**步骤4**指定的obs_path中，日志上传到**步骤7**指定的log_url中。

步骤9 多次调试，与单机调测时**步骤8**作用相同。

步骤10 查询训练支持的工作节点类型，与单机调测时**步骤9**相同。

步骤11 提交远程训练作业。

```
from modelarts.estimatorV2 import Estimator
parameters = []
parameters.append({"name": "data_url", "value": data_local})
parameters.append({"name": "output_dir", "value": os.path.join(base_local_path, "output/")})
parameters.append({"name": "epoc_num", "value": 2})
# 启动脚本以parser.add_argument('--dist', action='store_true')的形式来接收该布尔类型的参数，如果要传入True，则以本行代码的形式传递；
parameters.append({"name": "dist"})
estimator = Estimator(session=session,
                      training_files=training_file,
                      outputs=[output],
                      parameters=parameters,
                      framework_type='PyTorch',
                      train_instance_type='modelarts.p3.large.public.distributed',
                      train_instance_count=2,
                      script_interpreter="/home/ma-user/anaconda3/envs/PyTorch-1.4/bin/python",
                      log_url=base_bucket_path + 'log/',
                      job_description='This is a image net train job')
estimator.fit(inputs=[input_data], job_name="cifar10-dis-1")
```

Estimator初始化时与本地训练的区别在于参数train_instance_type，需要从**步骤10**得到的结果中选择一个；参数train_instance_count的值取决于第10步中的max_num。

训练任务提交后，SDK会依次帮助用户完成以下流程：

1. 将训练脚本打包成zip文件，上传到**步骤3**中指定的obs_path中；
2. 如果训练数据保存在Notebook中，则将其打包成zip文件并上传到指定的obs_path中；
3. 将训练作业提交到ModelArts训练服务中，训练作业会使用当前Notebook的镜像来执行训练作业；
4. 训练任务得到的输出上传到**步骤4**指定的obs_path中，日志上传到log_url指定的位置中。

   在这一步中需要注意的一个问题：

   如果用户在自己的训练脚本中要创建新的目录或文件，请在以下几种目录中创建：

   （1）/home/ma-user/work；

   （2）/cache；

   （3）inputs或者outputs中指定的local_path，如在**步骤2**中初始化InputData时，填写了local_path="/home/ma-user/work/xx/yy/"，则在该目录下也可以创建新目录或文件；

----结束

7.3 单机多卡数据并行-DataParallel(DP)

本章节介绍基于Pytorch引擎的单机多卡数据并行训练。

MindSpore引擎的分布式训练参见**MindSpore官网**。

 版权所有 © 华为技术有限公司

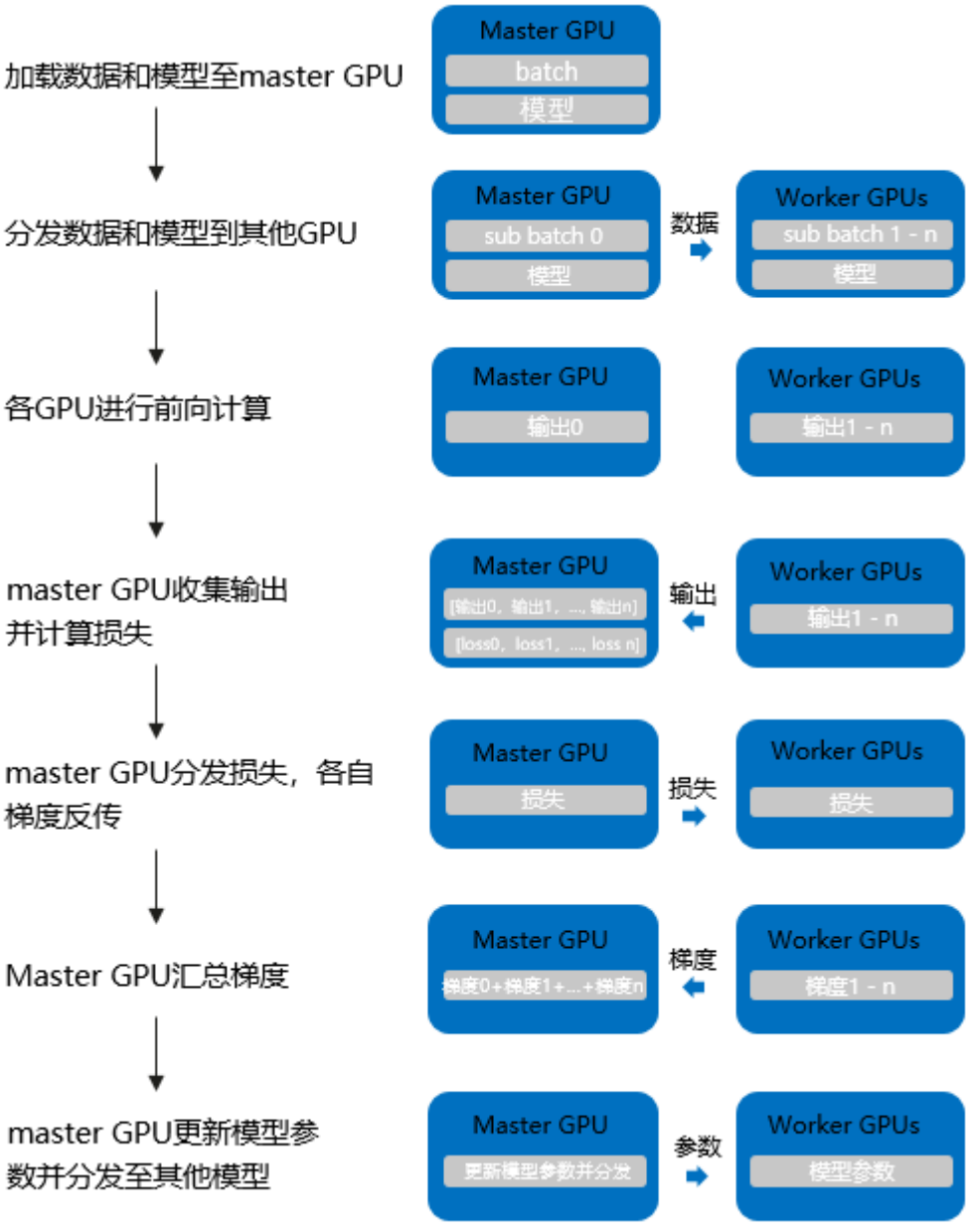
训练流程简述

单机多卡数据并行训练流程介绍如下：

1. 将模型拷贝到多个GPU上
2. 将一个Batch的数据均分到每一个GPU上
3. 各GPU上的模型进行前向传播，得到输出
4. 主GPU（逻辑序号为0）收集各GPU的输出，汇总后计算损失
5. 分发损失，各GPU各自反向传播梯度
6. 主GPU收集梯度并更新参数，将更新后的模型参数分发到各GPU

具体流程图如下：

图 7-1 单机多卡数据并行训练



版权所有 © 华为技术有限公司

DataParallel 进行单机多卡训练的优缺点

- 代码简单：仅需修改一行代码
- 通信瓶颈 ： 负责reducer的GPU更新模型参数后分发到不同的GPU，因此有较大的通信开销。
- GPU负载不均衡：负责reducer的GPU需要负责汇总输出、计算损失和更新权重，因此显存和使用率相比其他GPU都会更高。

代码改造点

模型分发：DataParallel(model)

完整代码 由于代码变动较少，此处进行简略介绍。

```
import torch
class Net(torch.nn.Module):
    pass

model = Net().cuda()

### DataParallel Begin ###
model = torch.nn.DataParallel(Net().cuda())
### DataParallel End ###
```

7.4 多机多卡数据并行-DistributedDataParallel(DDP)

本章节介绍基于Pytorch引擎的多机多卡数据并行训练。

训练流程简述

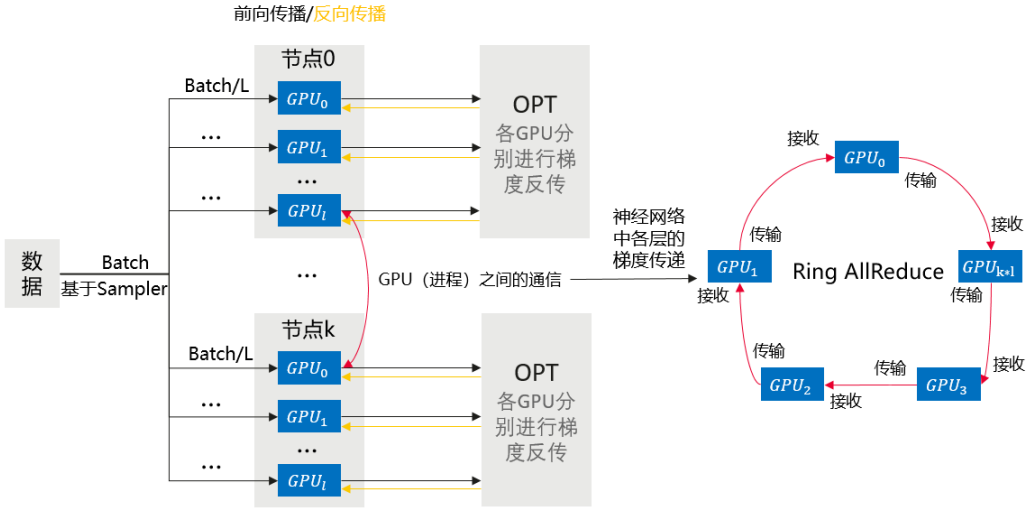
相比于DP，DDP能够启动多进程进行运算，从而大幅度提升计算资源的利用率。可以基于torch.distributed实现真正的分布式计算，具体的原理此处不再赘述。大致的流程如下：

1. 初始化进程组。
2. 创建分布式并行模型，每个进程都会有相同的模型和参数。
3. 创建数据分发Sampler，使每个进程加载一个min Batch中不同部分的数据
4. 网络中相邻参数分桶。
5. 每个进程前向传播并各自计算梯度。
6. 一个桶中的参数都得到梯度后会马上进行通讯，进行梯度平均
7. 各GPU更新模型参数

具体流程图如下：

 版权所有 © 华为技术有限公司

图 7-2 多机多卡数据并行训练



DistributedDataParallel 进行多机多卡训练的优缺点

- **通信更快**：相比于DP，通信速度更快
- **负载相对均衡**：相比于DP，GPU负载相对更均衡
- **运行速度快**：因为通信时间更短，效率更高，能更快速的完成训练任务

代码改造点

- 引入多进程启动机制：初始化进程
- 引入几个变量：tcp协议，rank 进程序号，worldsize 开启的进程数量
- 分发数据：DataLoader中多了一个Sampler参数，避免不同进程数据重复
- 模型分发：DistributedDataParallel(model)
- 模型保存：在序号为0的进程下保存模型

```
import torch
class Net(torch.nn.Module):
   pass

model = Net().cuda()

### DataParallel Begin ###
model = torch.nn.DataParallel(Net().cuda())
### DataParallel End ###
```

7.5 分布式调测适配及代码示例

在DistributedDataParallel中，不同进程分别从原始数据中加载batch的数据，最终将各个进程的梯度进行平均作为最终梯度，由于样本量更大，因此计算出的梯度更加可靠，可以适当增大学习率。

以下对resnet18在cifar10数据集上的分类任务，给出了单机训练和分布式训练改造(DDP)的代码。直接执行代码为多节点分布式训练且支持CPU分布式和GPU分布式，将代码中的分布式改造点注释掉后即可进行单节点单卡训练。

训练代码中包涵三部分入参，分别为训练基础参数、分布式参数和数据相关参数。其中分布式参数由平台自动入参，无需自行定义。数据相关参数中的custom_data表示

版权所有 © 华为技术有限公司

是否使用自定义数据进行训练，该参数为“true”时使用基于torch自定义的随机数据进行训练和验证。

数据集

cifar10数据集

在Notebook中，无法直接使用默认版本的torchvision获取数据集，因此示例代码中提供了三种训练数据加载方式。

cifar-10数据集下载链接**http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz**。

- 尝试基于torchvision获取cifar10数据集。
- 基于数据链接下载数据并解压，放置在指定目录下，训练集和测试集的大小分别为(50000, 3, 32, 32)和(10000, 3, 32, 32)。
- 考虑到下载cifar10数据集较慢，基于torch生成类似cifar10的随机数据集，训练集和测试集的大小分别为(5000, 3, 32, 32)和(1000, 3, 32, 32)，标签仍为10类，指定custom_data = 'true'后可直接进行训练任务，无需加载数据。

训练代码

以下代码中以 ### 分布式改造，... ### 注释的代码即为多节点分布式训练需要适配的代码改造点。

不对示例代码进行任何修改，适配数据路径后即可在ModelArts上完成多节点分布式训练。

注释掉分布式代码改造点，即可完成单节点单卡训练。完整代码见**7.6 完整代码示例**。

- **导入依赖包**

```
import datetime
import inspect
import os
import pickle
import random

import argparse
import numpy as np
import torch
import torch.distributed as dist
from torch import nn, optim
from torch.utils.data import TensorDataset, DataLoader
from torch.utils.data.distributed import DistributedSampler
from sklearn.metrics import accuracy_score
```

- **定义加载数据的方法和随机数**，由于加载数据部分代码较多，此处省略

```
def setup_seed(seed):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)
    torch.backends.cudnn.deterministic = True

def get_data(path):
    pass
```

- **定义网络结构**

```
class Block(nn.Module):

    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
```

 版权所有 © 华为技术有限公司

```
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels)
        )

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.residual_function(x) + self.shortcut(x)
        return nn.ReLU(inplace=True)(out)


class ResNet(nn.Module):

    def __init__(self, block, num_classes=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2 = self.make_layer(block, 64, 64, 2, 1)
        self.conv3 = self.make_layer(block, 64, 128, 2, 2)
        self.conv4 = self.make_layer(block, 128, 256, 2, 2)
        self.conv5 = self.make_layer(block, 256, 512, 2, 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.dense_layer = nn.Linear(512, num_classes)

    def make_layer(self, block, in_channels, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(in_channels, out_channels, stride))
            in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.conv5(out)
        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.dense_layer(out)
        return out
```

- **进行训练和验证**

```
def main():
    file_dir = os.path.dirname(inspect.getframeinfo(inspect.currentframe()).filename)

    seed = datetime.datetime.now().year
    setup_seed(seed)

    parser = argparse.ArgumentParser(description='Pytorch distribute training',
                                     formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--enable_gpu', default='true')
    parser.add_argument('--lr', default='0.01', help='learning rate')
    parser.add_argument('--epochs', default='100', help='training iteration')

    parser.add_argument('--init_method', default=None, help='tcp_port')
    parser.add_argument('--rank', type=int, default=0, help='index of current task')
    parser.add_argument('--world_size', type=int, default=1, help='total number of tasks')

    parser.add_argument('--custom_data', default='false')
```

 版权所有 © 华为技术有限公司

```
    parser.add_argument('--data_url', type=str, default=os.path.join(file_dir, 'input_dir'))
    parser.add_argument('--output_dir', type=str, default=os.path.join(file_dir, 'output_dir'))
    args, unknown = parser.parse_known_args()

    args.enable_gpu = args.enable_gpu == 'true'
    args.custom_data = args.custom_data == 'true'
    args.lr = float(args.lr)
    args.epochs = int(args.epochs)

    if args.custom_data:
        print('[warning] you are training on custom random dataset, '
              'validation accuracy may range from 0.4 to 0.6.')

    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参 ###
    dist.init_process_group(init_method=args.init_method, backend="nccl", world_size=args.world_size,
rank=args.rank)
    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参 ###

    tr_set, val_set = get_data(args.data_url, custom_data=args.custom_data)

    batch_per_gpu = 128
    gpus_per_node = torch.cuda.device_count() if args.enable_gpu else 1
    batch = batch_per_gpu * gpus_per_node

    tr_loader = DataLoader(tr_set, batch_size=batch_per_gpu, shuffle=False)

    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###
    tr_sampler = DistributedSampler(tr_set, num_replicas=args.world_size, rank=args.rank)
    tr_loader = DataLoader(tr_set, batch_size=batch, sampler=tr_sampler, shuffle=False, drop_last=True)
    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###

    val_loader = DataLoader(val_set, batch_size=batch_per_gpu, shuffle=False)

    lr = args.lr * gpus_per_node
    max_epoch = args.epochs
    model = ResNet(Block).cuda() if args.enable_gpu else ResNet(Block)

    ### 分布式改造，构建DDP分布式模型 ###
    model = nn.parallel.DistributedDataParallel(model)
    ### 分布式改造，构建DDP分布式模型 ###

    optimizer = optim.Adam(model.parameters(), lr=lr)
    loss_func = torch.nn.CrossEntropyLoss()

    os.makedirs(args.output_dir, exist_ok=True)

    for epoch in range(1, max_epoch + 1):
        model.train()
        train_loss = 0

        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据 ###
        tr_sampler.set_epoch(epoch)
        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据 ###

        for step, (tr_x, tr_y) in enumerate(tr_loader):
            if args.enable_gpu:
                tr_x, tr_y = tr_x.cuda(), tr_y.cuda()
            out = model(tr_x)
            loss = loss_func(out, tr_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
        print('train | epoch: %d | loss: %.4f' % (epoch, train_loss / len(tr_loader)))

        val_loss = 0
        pred_record = []
        real_record = []
        model.eval()
```

 版权所有 © 华为技术有限公司

```
        with torch.no_grad():
            for step, (val_x, val_y) in enumerate(val_loader):
                if args.enable_gpu:
                    val_x, val_y = val_x.cuda(), val_y.cuda()
                out = model(val_x)
                pred_record += list(np.argmax(out.cpu().numpy(), axis=1))
                real_record += list(val_y.cpu().numpy())
                val_loss += loss_func(out, val_y).item()
        val_accu = accuracy_score(real_record, pred_record)
        print('val | epoch: %d | loss: %.4f | accuracy: %.4f' % (epoch, val_loss / len(val_loader), val_accu),
'\n')

        if args.rank == 0:
            # save ckpt every epoch
            torch.save(model.state_dict(), os.path.join(args.output_dir, f'epoch_{epoch}.pth'))


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

- **结果对比**

  分别以单机单卡和两节点16卡两种资源类型完成100epoch的cifar-10数据集训练，训练时长和测试集准确率如下。

  表 7-1 训练结果对比

资源类型	单机单卡	两节点16卡
耗时	60分钟	20分钟
准确率	80+	80+

7.6 完整代码示例

以下对resnet18在cifar10数据集上的分类任务，给出了分布式训练改造(DDP)的完整代码示例。

```
import datetime
import inspect
import os
import pickle
import random
import logging

import argparse
import numpy as np
from sklearn.metrics import accuracy_score
import torch
from torch import nn, optim
import torch.distributed as dist
from torch.utils.data import TensorDataset, DataLoader
from torch.utils.data.distributed import DistributedSampler

file_dir = os.path.dirname(inspect.getframeinfo(inspect.currentframe()).filename)


def load_pickle_data(path):
    with open(path, 'rb') as file:
        data = pickle.load(file, encoding='bytes')
    return data
```

 版权所有 © 华为技术有限公司

```
def _load_data(file_path):
    raw_data = load_pickle_data(file_path)
    labels = raw_data[b'labels']
    data = raw_data[b'data']
    filenames = raw_data[b'filenames']

    data = data.reshape(10000, 3, 32, 32) / 255
    return data, labels, filenames


def load_cifar_data(root_path):
    train_root_path = os.path.join(root_path, 'cifar-10-batches-py/data_batch_')
    train_data_record = []
    train_labels = []
    train_filenames = []
    for i in range(1, 6):
        train_file_path = train_root_path + str(i)
        data, labels, filenames = _load_data(train_file_path)
        train_data_record.append(data)
        train_labels += labels
        train_filenames += filenames
    train_data = np.concatenate(train_data_record, axis=0)
    train_labels = np.array(train_labels)

    val_file_path = os.path.join(root_path, 'cifar-10-batches-py/test_batch')
    val_data, val_labels, val_filenames = _load_data(val_file_path)
    val_labels = np.array(val_labels)

    tr_data = torch.from_numpy(train_data).float()
    tr_labels = torch.from_numpy(train_labels).long()
    val_data = torch.from_numpy(val_data).float()
    val_labels = torch.from_numpy(val_labels).long()
    return tr_data, tr_labels, val_data, val_labels


def get_data(root_path, custom_data=False):
    if custom_data:
        train_samples, test_samples, img_size = 5000, 1000, 32
        tr_label = [1] * int(train_samples / 2) + [0] * int(train_samples / 2)
        val_label = [1] * int(test_samples / 2) + [0] * int(test_samples / 2)
        random.seed(2021)
        random.shuffle(tr_label)
        random.shuffle(val_label)
        tr_data, tr_labels = torch.randn((train_samples, 3, img_size, img_size)).float(),
torch.tensor(tr_label).long()
        val_data, val_labels = torch.randn((test_samples, 3, img_size, img_size)).float(),
torch.tensor(
            val_label).long()
        tr_set = TensorDataset(tr_data, tr_labels)
        val_set = TensorDataset(val_data, val_labels)
        return tr_set, val_set
    elif os.path.exists(os.path.join(root_path, 'cifar-10-batches-py')):
        tr_data, tr_labels, val_data, val_labels = load_cifar_data(root_path)
        tr_set = TensorDataset(tr_data, tr_labels)
        val_set = TensorDataset(val_data, val_labels)
        return tr_set, val_set
    else:
        try:
            import torchvision
            from torchvision import transforms
```

 版权所有 © 华为技术有限公司

```
            tr_set = torchvision.datasets.CIFAR10(root='./data', train=True,
                                                  download=True, transform=transforms)
            val_set = torchvision.datasets.CIFAR10(root='./data', train=False,
                                                   download=True, transform=transforms)
            return tr_set, val_set
        except Exception as e:
            raise Exception(
                f"{e}, you can download and unzip cifar-10 dataset manually, "
                "the data url is http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz")


class Block(nn.Module):

    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1,
bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels)
        )

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.residual_function(x) + self.shortcut(x)
        return nn.ReLU(inplace=True)(out)


class ResNet(nn.Module):

    def __init__(self, block, num_classes=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2 = self.make_layer(block, 64, 64, 2, 1)
        self.conv3 = self.make_layer(block, 64, 128, 2, 2)
        self.conv4 = self.make_layer(block, 128, 256, 2, 2)
        self.conv5 = self.make_layer(block, 256, 512, 2, 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.dense_layer = nn.Linear(512, num_classes)

    def make_layer(self, block, in_channels, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(in_channels, out_channels, stride))
            in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
```

```
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.conv5(out)
        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.dense_layer(out)
        return out


def setup_seed(seed):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)
    torch.backends.cudnn.deterministic = True


def obs_transfer(src_path, dst_path):
    import moxing as mox
    mox.file.copy_parallel(src_path, dst_path)
    logging.info(f"end copy data from {src_path} to {dst_path}")


def main():
    seed = datetime.datetime.now().year
    setup_seed(seed)

    parser = argparse.ArgumentParser(description='Pytorch distribute training',
                                     formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--enable_gpu', default='true')
    parser.add_argument('--lr', default='0.01', help='learning rate')
    parser.add_argument('--epochs', default='100', help='training iteration')

    parser.add_argument('--init_method', default=None, help='tcp_port')
    parser.add_argument('--rank', type=int, default=0, help='index of current task')
    parser.add_argument('--world_size', type=int, default=1, help='total number of tasks')

    parser.add_argument('--custom_data', default='false')
    parser.add_argument('--data_url', type=str, default=os.path.join(file_dir, 'input_dir'))
    parser.add_argument('--output_dir', type=str, default=os.path.join(file_dir, 'output_dir'))
    args, unknown = parser.parse_known_args()

    args.enable_gpu = args.enable_gpu == 'true'
    args.custom_data = args.custom_data == 'true'
    args.lr = float(args.lr)
    args.epochs = int(args.epochs)

    if args.custom_data:
        logging.warning('you are training on custom random dataset, '
            'validation accuracy may range from 0.4 to 0.6.')

    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参
###
    dist.init_process_group(init_method=args.init_method, backend="nccl",
world_size=args.world_size, rank=args.rank)
    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参
###

    tr_set, val_set = get_data(args.data_url, custom_data=args.custom_data)

    batch_per_gpu = 128
    gpus_per_node = torch.cuda.device_count() if args.enable_gpu else 1
```

 版权所有 © 华为技术有限公司

```
    batch = batch_per_gpu * gpus_per_node

    tr_loader = DataLoader(tr_set, batch_size=batch_per_gpu, shuffle=False)

    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###
    tr_sampler = DistributedSampler(tr_set, num_replicas=args.world_size, rank=args.rank)
    tr_loader = DataLoader(tr_set, batch_size=batch, sampler=tr_sampler, shuffle=False,
drop_last=True)
    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###

    val_loader = DataLoader(val_set, batch_size=batch_per_gpu, shuffle=False)

    lr = args.lr * gpus_per_node * args.world_size
    max_epoch = args.epochs
    model = ResNet(Block).cuda() if args.enable_gpu else ResNet(Block)

    ### 分布式改造，构建DDP分布式模型 ###
    model = nn.parallel.DistributedDataParallel(model)
    ### 分布式改造，构建DDP分布式模型 ###

    optimizer = optim.Adam(model.parameters(), lr=lr)
    loss_func = torch.nn.CrossEntropyLoss()

    os.makedirs(args.output_dir, exist_ok=True)

    for epoch in range(1, max_epoch + 1):
        model.train()
        train_loss = 0

        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据
###
        tr_sampler.set_epoch(epoch)
        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据
###

        for step, (tr_x, tr_y) in enumerate(tr_loader):
            if args.enable_gpu:
                tr_x, tr_y = tr_x.cuda(), tr_y.cuda()
            out = model(tr_x)
            loss = loss_func(out, tr_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
        print('train | epoch: %d | loss: %.4f' % (epoch, train_loss / len(tr_loader)))

        val_loss = 0
        pred_record = []
        real_record = []
        model.eval()
        with torch.no_grad():
            for step, (val_x, val_y) in enumerate(val_loader):
                if args.enable_gpu:
                    val_x, val_y = val_x.cuda(), val_y.cuda()
                out = model(val_x)
                pred_record += list(np.argmax(out.cpu().numpy(), axis=1))
                real_record += list(val_y.cpu().numpy())
                val_loss += loss_func(out, val_y).item()
        val_accu = accuracy_score(real_record, pred_record)
        print('val | epoch: %d | loss: %.4f | accuracy: %.4f' % (epoch, val_loss / len(val_loader),
val_accu), '\n')
```

 版权所有 © 华为技术有限公司

```
        if args.rank == 0:
            # save ckpt every epoch
            torch.save(model.state_dict(), os.path.join(args.output_dir, f'epoch_{epoch}.pth'))


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

 版权所有 © 华为技术有限公司

8 自动模型优化（AutoSearch）

8.1 超参搜索简介

ModelArts新版训练中新增了超参搜索功能，自动实现模型超参搜索，为您的模型匹配最优的超参。

在模型训练过程中，有很多超参需要根据任务进行调整，比如learning_rate、weight_decay等等，这一工作往往需要一个有经验的算法工程师花费一定精力和大量时间进行手动调优。ModelArts支持的超参搜索功能，在无需算法工程师介入的情况下，即可自动进行超参的调优，在速度和精度上超过人工调优。

ModelArts支持以下三种超参搜索算法：

- **8.2.1 贝叶斯优化(SMAC)**
- **8.2.2 TPE算法**
- **8.2.3 模拟退火算法(Anneal)**

8.2 搜索算法

8.2.1 贝叶斯优化(SMAC)

贝叶斯优化假设超参和目标函数存在一个函数关系。基于已搜索超参的评估值，通过高斯过程回归来估计其他搜索点处目标函数值的均值和方差。根据均值和方差构造采集函数（Acquisition Function），下一个搜索点为采集函数的极大值点。相比网格搜索，贝叶斯优化会利用之前的评估结果，从而降低迭代次数、缩短搜索时间；缺点是不容易找到全局最优解。

 版权所有 © 华为技术有限公司

表 8-1 贝叶斯优化的参数说明

参数	说明	取值参考
num_samples	搜索尝试的超参组数	int，一般在10-20之间，值越大，搜索时间越长，效果越好
kind	采集函数类型	string，默认为'ucb'，可能取值还有'ei'、'poi'，一般不建议用户修改
kappa	采集函数ucb的调节参数，可理解为上置信边界	float，一般不建议用户修改
xi	采集函数poi和ei的调节参数	float，一般不建议用户修改

8.2.2 TPE 算法

TPE算法全称Tree-structured Parzen Estimator，是一种利用高斯混合模型来学习超参模型的算法。在每次试验中，对于每个超参，TPE为与最佳目标值相关的超参维护一个高斯混合模型l(x)，为剩余的超参维护另一个高斯混合模型g(x)，选择l(x)/g(x)最大化时对应的超参作为下一组搜索值。

表 8-2 TPE 算法的参数说明

参数	说明	取值参考
num_samples	搜索尝试的超参组数	int，一般在10-20之间，值越大，搜索时间越长，效果越好
n_initial_points	采用TPE接近目标函数之前，对目标函数的随机评估数	int，一般不建议用户修改
gamma	TPE算法的一定分位数，用于划分l(x)和g(x)	float，范围(0,1)，一般不建议用户修改

8.2.3 模拟退火算法(Anneal)

模拟退火算法即Anneal算法，是随机搜索中一个简单但有效的变体，它利用了响应曲面中的平滑度。退火速率不自适应。Anneal算法从先前采样的一个试验点作为起点，然后从与先验分布相似的分布中采样每组超参数，但其密度更集中在我们选择的试验点周围。随着时间推移，算法会倾向于从越来越接近最佳点处采样。在采样过程中，算法可能绘制一个次佳试验作为最佳试验，以一定概率跳出局部最优解。

版权所有 © 华为技术有限公司

表 8-3 模拟退火算法的参数说明

参数	说明	取值参考
num_samples	搜索尝试的超参组数	int，一般在10-20之间，值越大，搜索时间越长，效果越好
avg_best_idx	要探索试验的几何分布平均，从按照分数排序的试验中选择	float，一般不建议用户修改
shrink_coef	随着更多的点被探索，邻域采样大小的减少率	float，一般不建议用户修改

8.3 创建超参搜索作业

背景信息

对于用户希望优化的超参，需在“超参”设置中定义，可以给定名称、类型、默认值、约束等，具体设置方法可以参考**定义超参**。

如果用户使用的AI引擎为“PyTorch-1.4.0-python3.6-v2”或“TF-1.13.1-python3.6-v2”，并且优化的超参类型为float类型，ModelArts支持用户使用超参搜索功能。

在0代码修改的基础下，实现算法模型的超参搜索。需要完成以下步骤：

1. **准备工作**
2. **创建算法**
3. **创建训练作业（New）**
4. **查看超参搜索作业详情**

准备工作

- 数据已完成准备：已在ModelArts中创建可用的数据集，或者您已将用于训练的数据集上传至OBS目录。
- 请准备好训练脚本，并上传至OBS目录。训练脚本开发指导参见**3.3.2 开发自定义脚本**。
- 在训练代码中，用户需打印搜索指标参数。
- 已在OBS创建至少1个空的文件夹，用于存储训练输出的内容。
- 由于训练作业运行需消耗资源，确保帐户未欠费。
- 确保您使用的OBS目录与ModelArts在同一区域。

创建算法

进入ModelArts控制台，参考**3.3.3 创建算法**操作指导，创建自定义算法。在配置自定义算法参数时，需关注“超参”和“支持的策略”参数的设置。

版权所有 © 华为技术有限公司

图 8-1 超参搜索作业参数

对于用户希望优化的超参，需在“超参”设置中定义，可以给定名称、类型、默认值、约束等，具体设置方法可以参考**定义超参**。

单击勾选自动搜索，用户为算法设置算法搜索功能。自动搜索作业运行过程中，ModelArts后台通过指标正则表达式获取搜索指标参数，朝指定的优化方向进行超参优化。用户需要在代码中打印搜索参数并在控制台配置以下参数：

- 搜索指标

  搜索指标为目标函数的值，通常可以设置为loss、accuracy等。通过优化搜索指标的目标值超优化方向收敛，找到最契合的超参，提高模型精度和收敛速度。

  表 8-4 搜索指标参数

参数	说明
名称	搜索指标的名称。需要与您在代码中打印的搜索指标参数保持一致。
优化方向	可选“最大化”或者“最小化”。
指标正则	填入正则表达式。您可以单击智能生成功能自动获取正则表达式。

图 8-2 搜索指标参数

 版权所有 © 华为技术有限公司

- 设置自动化搜索参数

  从已设置的“超参”中选择可用于搜索优化的超参。优化的超参仅支持float类型，选中自动化搜索参数后，需设置取值范围。

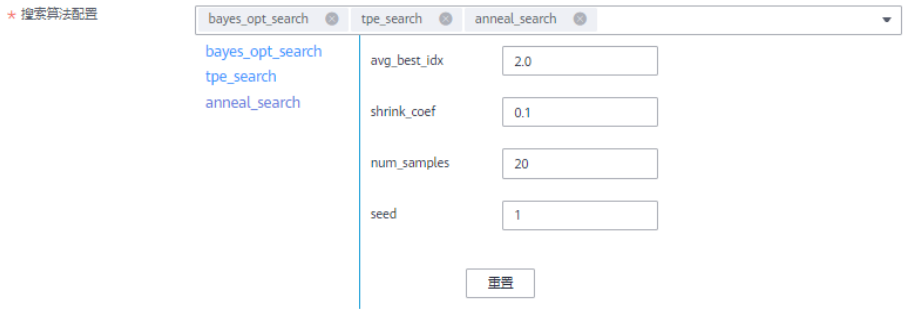
  图 8-3 自动化参数设置

- 搜索算法配置

  ModelArts内置三种超参搜索算法，用户可以根据实际情况选择对应的算法，支持多选。对应的算法和参数解析请参考以下：

  - bayes_opt_search:**8.2.1 贝叶斯优化(SMAC)**
  - tpe_search:**8.2.2 TPE算法**
  - anneal_search:**8.2.3 模拟退火算法(Anneal)**

  图 8-4 搜索算法配置



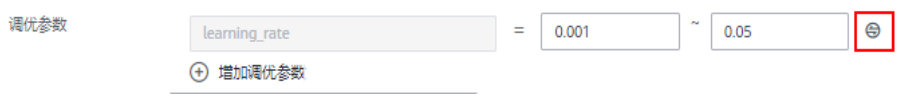
创建算法完成后，在“算法管理”页面，等待算法就绪。当新创建的算法状态变更为“就绪”时，即可执行其他操作。

创建训练作业（New）

登录ModelArts控制台，参考**4.1 创建训练作业**操作指导，创建训练作业。用户需关注以下操作才能开启超参搜索。

当您选择支持超参搜索的算法，需单击超级参数的范围设置按钮才能开启超参搜索功能。

图 8-5 范围设置



开启超参搜索功能后，用户可以设置搜索指标、搜索算法和搜索算法参数。三个参数显示的支持值与**算法管理模块的超参设置**一一对应。

版权所有 © 华为技术有限公司

图 8-6 超参搜索参数设置

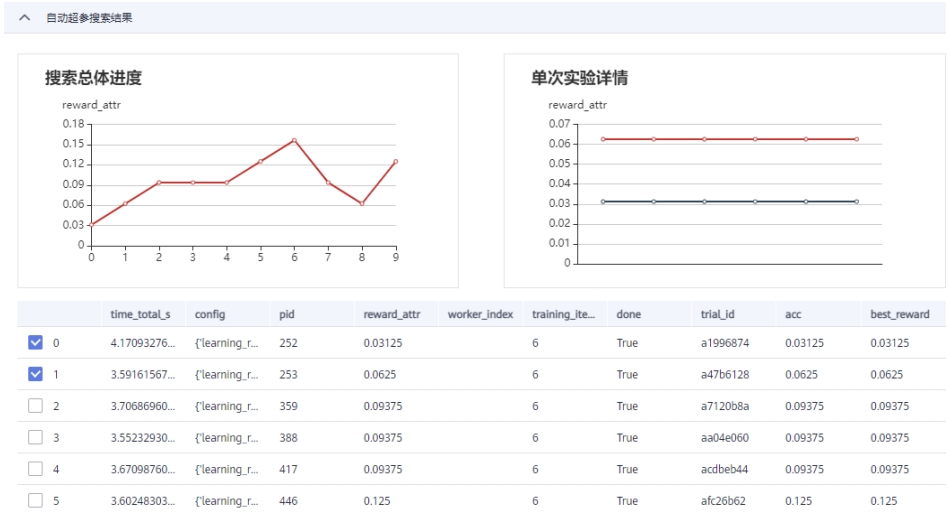
完成超参搜索作业的创建后，训练作业需要运行一段时间。

查看超参搜索作业详情

训练作业运行结束后，您可以通过查看训练作业详情判断此训练作业是否满意。

进入ModelArts控制台，您可以参考**4.2 查看作业详情**操作指导。如果训练作业是超参搜索作业，用户可以查看自动超参搜索结果。

图 8-7 超参搜索结果



	time_total_s	config	pid	reward_attr	worker_index	training_ite...	done	trial_id	acc	best_reward
0	4.17093276...	{'learning_r...	252	0.03125		6	True	a1996874	0.03125	0.03125
1	3.59161567...	{'learning_r...	253	0.0625		6	True	a47b6128	0.0625	0.0625
2	3.70686960...	{'learning_r...	359	0.09375		6	True	a7120b8a	0.09375	0.09375
3	3.55232930...	{'learning_r...	388	0.09375		6	True	aa04e060	0.09375	0.09375
4	3.67098760...	{'learning_r...	417	0.09375		6	True	acdbeb44	0.09375	0.09375
5	3.60248303...	{'learning_r...	446	0.125		6	True	afc26b62	0.125	0.125

版权所有 © 华为技术有限公司

9 模型转换

9.1 模型转换操作

针对您在ModelArts或者本地构建的模型，为获得更高的算力，希望将模型应用于Ascend芯片、ARM或GPU上，此时，您需要将已有模型压缩/转换成相应的格式后，再应用至不同的芯片类型。

ModelArts提供了模型转换功能，即将已有的模型转换成所需格式，以便应用于算力和性能更高的芯片上。

模型转换主要应用场景如下所示：

- 使用Caffe（.caffemodel格式）或者Tensorflow框架（“frozen_graph”或“saved_model”格式）训练的模型，使用转换功能可转换成om格式，转换后的模型可在昇腾（Ascend）芯片上部署运行。
- 使用Tensorflow框架训练模型（frozen_graph或“saved_model”格式）， 使用转换功能可以将模型转换量化成tflite格式，转换后的模型可以在ARM上部署运行。
- 使用Tensorflow框架训练模型（frozen_graph或“saved_model”格式）， 使用转换功能可以将模型转换量化成tensorRT格式， 转换后的模型可以在nvidia p4 GPU上部署运行。

约束限制

- 模型转换当前只支持三种芯片类型，分别为：Ascend、ARM、GPU。
- 模型转换当前仅支持原始框架类型为Caffe和Tensorflow的模型转换。当原始框架类型为Caffe时，输入数据类型为FLOAT；当原始框架类型为Tensorflow时，输入数据类型为INT32、BOOL、UINT8、FLOAT。
- ModelArts提供了转换模板供用户选择，只能选择对应模板进行转换，支持的模板描述，请参见**9.4 转换模板**。

 版权所有 © 华为技术有限公司

- 现阶段由于tflite和tensorRT支持的算子和量化算子有限，可能存在部分模型转换失败的情况。如果出现转换失败，可以将页面的日志框滑到最后或者去转换输出目录下查看错误日志。
- 针对用于Ascend芯片的模型转换，其转换限制说明可参见“昇腾开发者社区”的**约束及参数说明**。
- 压缩/转换任务指定的OBS路径，需确保OBS目录与ModelArts在同一区域。
- 转换后的模型，再导入ModelArts时，需**从模板中选择元模型**。
- 支持模型文件类型为onnx的模型转换，转换时会先将其转换为Tensorflow框架的FrozenGraphDef格式，然后再转换为om格式。转换工具要求onnx版本为1.6.0，opset为9+。
- 当原始框架类型为Caffe时，模型文件（.prototxt）和权重文件（.caffemodel）的op name、op type必须保持名称一致（包括大小写）。
- 当原始框架类型为Caffe时，除了top与bottom相同的layer以外（例如BatchNorm，Scale，ReLU等），其他layer的top名称需要与其name名称保持一致。
- 当原始框架类型为TensorFlow时，支持FrozenGraphDef格式和SavedModel 格式。如果是SavedModel格式，转换时会先将其转换为FrozenGraphDef格式，然后再转换为om格式。
- 不支持动态shape的输入，例如：NHWC输入为[?,?,?,3]多个维度可任意指定数值。模型转换时需指定固定数值。
- 输入数据最大支持四维，转维算子（reshape、expanddim等）不能输出五维。
- 模型中的所有层算子除const算子外，输入和输出需要满足“dim!=0”。
- 模型转换不支持含有训练算子的模型。
- 量化（uint8）后的模型不支持模型转换。
- 模型中的算子只支持2D卷积，暂不支持3D卷积。暂不支持多批量转换batch_normalization_1算子和FusedBatchNorm算子。
- 只支持**Caffe算子清单**和**Tensorflow算子**清单中的算子，并需满足算子限制条件。

创建模型转换任务

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI应用管理 > 模型转换”，进入模型转换列表页面。
2. 单击左上角的“创建任务”，进入任务创建任务页面。
3. 在“创建任务”页面，参考以下说明，填写转换任务的详细参数。
   a. 填写转换任务的“名称”和“描述”信息。
   b. 填写转换任务的具体参数，详细描述请参见**表9-1**。

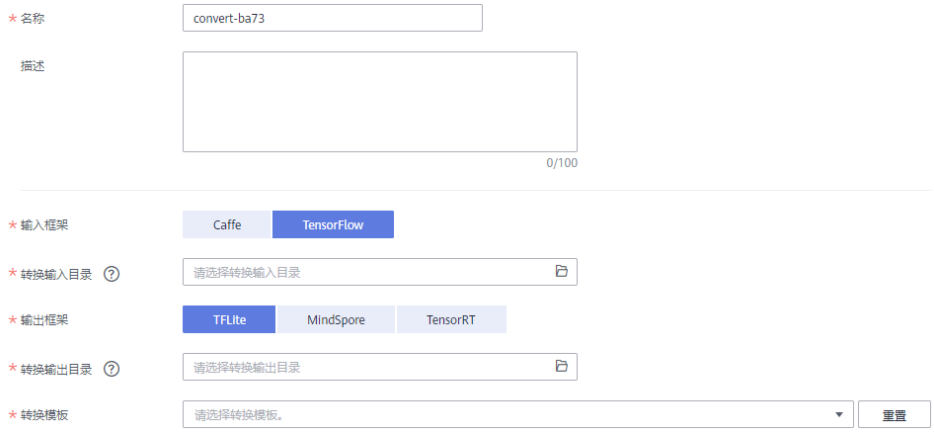
表 9-1 参数说明

参数	说明
输入框架	目前支持转换的框架有“Caffe”和“TensorFlow”。

 版权所有 © 华为技术有限公司

参数	说明
转换输入目录	用于转换的模型所在目录，此目录必须为OBS目录，且模型文件的目录需符合ModelArts规范，详情请参见**9.2 模型输入目录规范**。
输出框架	当“输入框架”选择“Caffe”时，“输出框架”支持“MindSpore”。 当“输入框架”选择“TensorFlow”时，“输出框架”支持“TFLite”、“MindSpore”、“TensorRT”。
转换输出目录	模型转换完成后，根据此参数设置的目录存储模型。输出目录需符合ModelArts规范要求，详情请参见**9.3 模型输出目录说明**。
转换模板	ModelArts提供了一系列的模板，定义转换功能以及转换过程中所需的参数。当前支持的转换模板详细描述请参见**9.4 转换模板**。转换模板右侧下拉框，将根据您选择的“输入框架”和“输出框架”，展现匹配的模板。 在下拉框选择模板后，下方将呈现此模板对应的高级参数。例如量化精度，方便您可以对模型转换任务进行更高阶的设置。 不同的转换模板，其对应的高级选项支持的参数不同，每个模板支持的详细参数，请参见**9.4 转换模板**。

图 9-1 创建模型转换任务



4. 任务信息填写完成后，单击右下角“立即创建”。

   创建完成后，系统自动跳转至“模型转换”列表中。刚创建的转换任务将呈现在界面中，其“任务状态”为“初始化”。任务执行过程预计需要几分钟到十几分钟不等，请耐心等待，当“任务状态”变为“成功”时，表示任务运行完成并且模型转换成功。

   如果“任务状态”变为“失败”，建议单击任务名称进入详情页面，查看日志信息，根据日志信息调整任务的相关参数并创建新的转换任务。

   模型转换成功后，有如下几个使用场景：

   - 可以在HiLens管理控制台，导入转换后的模型，导入后可将模型安装部署至HiLens Kits设备上。

版权所有 © 华为技术有限公司

- 登录“转换输出目录”对应的OBS路径，下载目录中的模型（.om格式），部署至您的设备中。

删除模型转换任务

针对运行结束的任务，如果不需要再使用，您可以删除转换任务。其中，“运行中”或“初始化”状态中的任务不支持删除操作。

说明

任务删除后，将无法恢复，请谨慎操作。

- **删除单个**

  在“模型转换列表”中，针对需要删除的单个任务，您可以在此任务所在行，单击操作列的“删除”，完成删除操作。

- **批量删除**

  在“模型转换列表”中，勾选多个待删除的任务，然后单击左上角“删除”，完成批量任务的删除操作。

9.2 模型输入目录规范

模型转换后，应用于不同的芯片，针对不同的芯片，其模型输入目录的要求不同。ModelArts当前对模型输入目录的要求为**Ascend芯片**和**ARM或GPU**两种。

Ascend 芯片

用于Ascend芯片的模型，其转换要求如下所示：

- 针对基于Caffe框架的模型，执行模型转换时，其输入目录需符合如下规范。

```
|
|---xxxx.caffemodel        模型参数文件，输入目录下有且只能有一个，必填。
|---xxxx.prototxt          模型网络文件，输入目录下有且只能有一个，必填。
|---insert_op_conf.cfg     插入算子配置文件，输入目录下有且只有一个，可选。
|---plugin                 自定义算子目录，输入目录下有且只能有一个plugin文件夹，可选。仅支持基于TE（Tensor Engine）开发的自定义算子。
```

- 针对基于TensorFlow框架的模型（“frozen_graph”或“saved_model”格式），执行模型转换时，其输入目录需符合如下规范。

  “frozen_graph”格式

```
|
|---xxxx.pb                模型网络文件，输入目录下有且只能有一个，必填。支持以frozen_graph或saved_model格式保存的模型。
|---insert_op_conf.cfg     插入算子配置文件，输入目录下有且只有一个，可选。
|---plugin                 自定义算子目录，输入目录下有且只能有一个plugin文件夹，可选。仅支持基于TE（Tensor Engine）开发的自定义算子。
```

  “saved_model”格式

```
|
|---saved_model.pb         模型网络文件，输入目录下有且只能有一个，必填。支持以frozen_graph或saved_model格式保存的模型。
|---variables              固定子目录名称，包含模型的权重偏差等信息，必选
   |---variables.index     必选
   |---variables.data-00000-of-00001 必选
|---insert_op_conf.cfg     插入算子配置文件，输入目录下有且只有一个，可选。
|---plugin                 自定义算子目录，输入目录下有且只能有一个plugin文件夹，可选。仅支持基于TE（Tensor Engine）开发的自定义算子。
```


版权所有 © 华为技术有限公司

ARM 或 GPU

用于ARM或GPU的模型，当前只支持TensorFlow框架的模型，包含两种格式“frozen_graph”和“saved_model”。

“frozen_graph”格式如下所示：

```
|
|---model                  模型存放目录，必须以model命名，有且只能有一个， 目录下只能放一个模型相关文件。
    |----xxx.pb            模型文件。必须是tensorflow的frozen_graph格式的文件。
|---calibration_data        校准数据集存放目录，必须以calibration_data命名，8bit和32bit转换都需要。输入目录下有且只能有一个。
32bit转换时可以直接使用python工具生成的test.npy文件。
    |---xx.npy             校准数据集。可以是多个npy格式文件，需要确保npy是在预处理后直接输入模型的数据，其输入的tensor需要与模型输入保持一致。
```

“saved_model”格式如下所示：

```
|
|---model                  模型存放目录，必须以model命名，有且只能有一个， 目录下只能放一个模型相关文件。
    |----saved_model.pb      模型文件。必须是tensorflow的saved_model格式的文件。
    |----variables          变量存储文件夹。
        |----variables.data-******-of-*****     saved_model格式文件需要的数据。
        |----variables.index                    saved_model格式文件需要的索引。
|---calibration_data        校准数据集存放目录，必须以calibration_data命名，8bit和32bit转换都需要。输入目录下有且只能有一个。32bit转换时可以直接使用python工具生成的test.npy文件。
    |---xx.npy             校准数据集。可以是多个npy格式文件，需要确保npy是在预处理后直接输入模型的数据，其输入的tensor需要与模型输入保持一致。
```

说明

32bit转换时，可以直接使用python工具生成的test.npy文件，并将此文件放置在上述两种格式的calibration_data目录下。

```
import numpy as np
a = np.arange(4)
np.save("test.npy", a)
```

9.3 模型输出目录说明

转换模型任务执行完成后，ModelArts将转换后的模型输出至指定的OBS路径。针对不同的转换任务，基于不同的芯片，其对应的目录有所区别，分为**Ascend芯片**和**ARM或GPU**两种。

Ascend 芯片

用于Ascend芯片的模型，其转换后输出目录说明如下所示：

- 针对基于Caffe框架的模型，执行模型转换时，其输出目录说明如下所示。

  ```
  |
  |---xxxx.om          转换输出的模型，可用于Ascend芯片，模型文件后缀统一为“.om”。
  |---job_log.txt      转换过程的日志文件。
  ```

- 针对基于TensorFlow框架的模型，执行模型转换时，其输出目录说明如下所示。

  ```
  |
  |---xxxx.om          转换输出的模型，可用于Ascend芯片，模型文件后缀统一为“.om”。
  |---job_log.txt      转换过程的日志文件。
  ```

ARM 或 GPU

用于ARM或GPU的模型，其转换后输出目录说明如下所示：

版权所有 © 华为技术有限公司

GPU格式如下所示：

```
|
|---model
        |---xxx.pb          GPU转换后模型后缀为“.pb”。
|---job_log.txt             转换过程的日志文件。
```

ARM格式如下所示：

```
|
|---model
        |---xxx.tflite          ARM转换后模型后缀为“.tflite”。
        |---config.json         8bit转换后，用户需要使用tflite时需要的参数。
|---job_log.txt         转换过程的日志文件。
```

9.4 转换模板

基于不同的AI框架，ModelArts提供的转换模板如下所示：

- **Caffe转Ascend**
- **Tensorflow frozen_graph转TFLite**
- **Tensorflow saved_model转TFLite**
- **Tensorflow frozen_graph转TensorRT**
- **Tensorflow saved_model转TensorRT**
- **Tensorflow frozen graph 转 Ascend**
- **TF-FrozenGraph-To-Ascend**
- **TF-SavedModel-To-Ascend**
- **Onnx-To-Ascend-TBE**
- **TF-FrozenGraph-To-Ascend-C32**
- **TF-SavedModel-To-Ascend-C32**
- **TF-FrozenGraph-To-Ascend-HiLens**
- **TF-SavedModel-To-Ascend-HiLens**

Caffe 转 Ascend

转换Caffe框架训练出来的模型， 转换后模型可在Ascend芯片上运行。

此模板无高级选项。

Tensorflow frozen_graph 转 TFLite

转换Tensorflow框架训练并以“frozen_graph”格式保存的模型，转换后模型可在ARM上运行。

表 9-2 Tensorflow frozen_graph 转 TFLite 的高级选项

参数名称	参数解释
“模型输入tensor名称”	以字符串形式输入模型输入张量名称，以“input1:input2”形式表示。

 版权所有 © 华为技术有限公司

参数名称	参数解释
“模型输出tensor名称”	以字符串形式输入模型输出张量名称，以“output1:output2”形式表示。
“量化精度”	可选择8bit或32bit。32bit表示直接转换模型，8bit表示模型进行量化。
“量化批大小”	以数值形式输入量化批大小。必须为正整数。

Tensorflow saved_model 转 TFLite

转换Tensorflow框架训练并以“saved_model”格式保存的模型，转换后模型可在ARM上运行。

表 9-3 Tensorflow saved_model 转 TFLite 的高级选项

参数名称	参数解释
“模型签名”	以字符串形式输入模型输入tensor签名，默认会选择第一个签名。
“传入模型标签”	以字符串形式输入模型输出标签，默认会选择第一个标签。
“量化精度”	可选择8bit或32bit。32bit表示直接转换模型，8bit表示模型进行量化。
“量化批大小”	以数值形式输入量化批大小。必须为正整数。

Tensorflow frozen_graph 转 TensorRT

转换Tensorflow框架训练并以“frozen_graph”格式保存的模型，转换后模型可在GPU上运行。

表 9-4 Tensorflow frozen_graph 转 TensorRT 的高级选项

参数名称	参数解释
“模型输入tensor名称”	以字符串形式输入模型输入张量名称，以“input1:input2”形式表示。
“模型输出tensor名称”	以字符串形式输入模型输出张量名称，以“output1:output2”形式表示。
“量化精度”	可选择8bit或32bit。32bit表示直接转换模型，8bit表示模型进行量化。
“量化批大小”	以数值形式输入量化批大小。必须为正整数。

 版权所有 © 华为技术有限公司

Tensorflow saved_model 转 TensorRT

转换Tensorflow框架训练并以“saved_model”格式保存的模型，转换后模型可在GPU上运行。

表 9-5 Tensorflow saved_model 转 TensorRT 的高级选项

参数名称	参数解释
“模型签名”	以字符串形式输入模型输入tensor签名，默认会选择第一个签名。
“传入模型标签”	以字符串形式输入模型输出标签，默认会选择第一个标签。
“量化精度”	可选择8bit或32bit。32bit表示直接转换模型，8bit表示模型进行量化。
“量化批大小”	以数值形式输入量化批大小。必须为正整数。

Tensorflow frozen graph 转 Ascend

转换Tensorflow框架训练并以“frozen_graph”格式保存的模型，转换后模型可在Ascend上运行。

表 9-6 Tensorflow frozen graph 转 Ascend 的高级选项

参数名称	参数解释
“输入张量形状”	模型输入数据的shape，输入数据格式为NHWC，如“input_name:1,224,224,3”，必填项。“input_name”必须是转换前的网络模型中的节点名称。当模型存在动态shape输入时必须提供。例如“input_name1:?,h,w,c”，该参数必填，其中“?”为batch数，表示1次处理的图片数量，需要根据实际情况填写，用于将动态shape的原始模型转换为固定shape的离线模型。目前不支持批量特性，转换输入张量形状batch只能为1。

TF-FrozenGraph-To-Ascend

转换Tensorflow框架训练并以“frozen_graph”格式保存的模型，转换后模型可在Ascend上运行。转换时支持使用基于TE（Tensor Engine）开发的自定义算子(TE算子)。

版权所有 © 华为技术有限公司

表 9-7 支持自定义算子转换模板的高级选项

参数名称	参数解释
“输入张量形状”	模型输入数据的shape，如“input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2”。“input_name”必须是转换前的网络模型中的节点名称。当模型存在动态shape输入时必须提供。例如“input_name1:?,h,w,c”，该参数必填，其中“?”为batch数，表示1次处理的图片数量，需要根据实际情况填写，用于将动态shape的原始模型转换为固定shape的离线模型。目前不支持批量特性，转换输入张量形状batch只能为1。转换时系统会解析输入模型获取输入张量并打印在日志中，如果不了解所使用模型的输入张量，可参考日志中的解析结果。
“输入数据格式”	支持NCHW和NHWC，默认是NHWC。当原始框架是TensorFlow时，默认是NHWC。如果实际是NCHW的话，需要通过此参数指定NCHW。原始框架为Caffe时，只支持NCHW格式。
“转换输出节点”	指定输出节点，例如“node_name1:0;node_name1:1;node_name2:0”，其中“node_name”必须是模型转换前的网络模型中的节点名称，冒号后的数字表示第几个输出，例如“node_name1:0”，表示节点名称为“node_name1”的第0个输出。如果不指定输出节点，则模型的输出默认为最后一层的算子信息，某些情况下，用户想要查看某层算子参数是否合适，则需要将该层算子的参数输出，即可以在模型转换时通过该参数指定输出某层算子。转换时系统会解析输入模型获取输出节点并打印在日志中，如果不了解所使用模型的输入张量，可以参考日志中的解析结果。
“优选数据格式”	指定网络算子优先选用的数据格式，“ND（N<=4）”和“5D”。仅在网络中算子的输入数据同时支持“ND”和“5D”两种格式时，指定该参数才会生效。“ND”表示模型中算子按NCHW转换成通用格式，“5D”表示模型中算子按华为自研的5维转换成华为格式。“5D”为默认值。
“生成高精度模型”	指定是否生成高精度FP16 Davinci模型。“0”为默认值，表示生成普通FP16 Davinci模型，推理性能更好。“1”表示生成高精度FP16 Davinci模型，推理精度更好。高精度当前仅支持Caffe算子（Convolution、Pooling、FullConnection）和TensorFlow算子（tf.nn.conv2d、tf.nn.max_poo）。
“网络输出数据类型”	FP32为默认值，推荐分类网络、检测网络使用。图像超分辨率网络，推荐使用UINT8，推理性能更好。

TF-SavedModel-To-Ascend

转换Tensorflow框架训练并以“saved_model”格式保存的模型，转换后模型可在Ascend上运行。转换时支持使用基于TE（Tensor Engine）开发的自定义算子(TE算子)。

 版权所有 © 华为技术有限公司

表 9-8 支持自定义算子转换模板的高级选项

参数名称	参数解释
“输入张量形状”	模型输入数据的shape，如“input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2”。“input_name”必须是转换前的网络模型中的节点名称。当模型存在动态shape输入时必须提供。例如“input_name1:?,h,w,c”，该参数必填，其中“?”为batch数，表示1次处理的图片数量，需要根据实际情况填写，用于将动态shape的原始模型转换为固定shape的离线模型。目前不支持批量特性，转换输入张量形状batch只能为1。转换时系统会解析输入模型获取输入张量并打印在日志中，如果不了解所使用模型的输入张量，可参考日志中的解析结果。
“输入数据格式”	支持NCHW和NHWC，默认是NHWC。当原始框架是TensorFlow时，默认是NHWC。如果实际是NCHW的话，需要通过此参数指定NCHW。原始框架为Caffe时，只支持NCHW格式。
“转换输出节点”	指定输出节点，例如“node_name1:0;node_name1:1;node_name2:0”，其中“node_name”必须是模型转换前的网络模型中的节点名称，冒号后的数字表示第几个输出，例如“node_name1:0”，表示节点名称为“node_name1”的第0个输出。如果不指定输出节点，则模型的输出默认为最后一层的算子信息，某些情况下，用户想要查看某层算子参数是否合适，则需要将该层算子的参数输出，即可以在模型转换时通过该参数指定输出某层算子。转换时系统会解析输入模型获取输出节点并打印在日志中，如果不了解所使用模型的输入张量，可以参考日志中的解析结果。
“优选数据格式”	指定网络算子优先选用的数据格式，“ND（N<=4）”和“5D”。仅在网络中算子的输入数据同时支持“ND”和“5D”两种格式时，指定该参数才会生效。“ND”表示模型中算子按NCHW转换成通用格式，“5D”表示模型中算子按华为自研的5维转换成华为格式。“5D”为默认值。
“生成高精度模型”	指定是否生成高精度FP16 Davinci模型。“0”为默认值，表示生成普通FP16 Davinci模型，推理性能更好。“1”表示生成高精度FP16 Davinci模型，推理精度更好。高精度当前仅支持Caffe算子（Convolution、Pooling、FullConnection）和TensorFlow算子（tf.nn.conv2d、tf.nn.max_poo）。
“网络输出数据类型”	FP32为默认值，推荐分类网络、检测网络使用。图像超分辨率网络，推荐使用UINT8，推理性能更好。

Onnx-To-Ascend-TBE

转换onnx格式保存的模型，模型首先被转换为frozen_graph格式，然后再转换为可在Ascend上运行的模型。转换时支持使用基于TBE（Tensor Based Engine）开发的自定义算子(TBE算子)。

 版权所有 © 华为技术有限公司

表 9-9 支持自定义算子转换模板的高级选项

参数名称	参数解释
“输入张量形状”	模型输入数据的shape，如“input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2”。“input_name”必须是转换前的网络模型中的节点名称。当模型存在动态shape输入时必须提供。例如“input_name1:?,h,w,c”，该参数必填，其中“?”为batch数，表示1次处理的图片数量，需要根据实际情况填写，用于将动态shape的原始模型转换为固定shape的离线模型。目前不支持批量特性，转换输入张量形状batch只能为1。转换时系统会解析输入模型获取输入张量并打印在日志中，如果不了解所使用模型的输入张量，可参考日志中的解析结果。
“输入数据格式”	支持NCHW和NHWC，默认是NHWC。当原始框架是TensorFlow时，默认是NHWC。如果实际是NCHW的话，需要通过此参数指定NCHW。原始框架为Caffe时，只支持NCHW格式。
“转换输出节点”	指定输出节点，例如“node_name1:0;node_name1:1;node_name2:0”，其中“node_name”必须是模型转换前的网络模型中的节点名称，冒号后的数字表示第几个输出，例如“node_name1:0”，表示节点名称为“node_name1”的第0个输出。如果不指定输出节点，则模型的输出默认为最后一层的算子信息，某些情况下，用户想要查看某层算子参数是否合适，则需要将该层算子的参数输出，即可以在模型转换时通过该参数指定输出某层算子。转换时系统会解析输入模型获取输出节点并打印在日志中，如果不了解所使用模型的输入张量，可以参考日志中的解析结果。
“优选数据格式”	指定网络算子优先选用的数据格式，“ND（N<=4）”和“5D”。仅在网络中算子的输入数据同时支持“ND”和“5D”两种格式时，指定该参数才会生效。“ND”表示模型中算子按NCHW转换成通用格式，“5D”表示模型中算子按华为自研的5维转换成华为格式。“5D”为默认值。
“生成高精度模型”	指定是否生成高精度FP16 Davinci模型。“0”为默认值，表示生成普通FP16 Davinci模型，推理性能更好。“1”表示生成高精度FP16 Davinci模型，推理精度更好。高精度当前仅支持Caffe算子（Convolution、Pooling、FullConnection）和TensorFlow算子（tf.nn.conv2d、tf.nn.max_poo）。
“网络输出数据类型”	FP32为默认值，推荐分类网络、检测网络使用。图像超分辨率网络，推荐使用UINT8，推理性能更好。

TF-FrozenGraph-To-Ascend-C32

转换Tensorflow框架训练并以“frozen_graph”格式保存的模型，转换后模型可在Ascend上运行。转换时支持使用基于TBE（Tensor Based Engine）开发的自定义算子(TBE算子)。

 版权所有 © 华为技术有限公司

表 9-10 支持自定义算子转换模板的高级选项

参数名称	参数解释
“输入张量形状”	模型输入数据的shape，如“input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2”。“input_name”必须是转换前的网络模型中的节点名称。当模型存在动态shape输入时必须提供。例如“input_name1:?,h,w,c”，该参数必填，其中“?”为batch数，表示1次处理的图片数量，需要根据实际情况填写，用于将动态shape的原始模型转换为固定shape的离线模型。目前不支持批量特性，转换输入张量形状batch只能为1。转换时系统会解析输入模型获取输入张量并打印在日志中，如果不了解所使用模型的输入张量，可参考日志中的解析结果。
“输入数据格式”	支持NCHW和NHWC，默认是NHWC。当原始框架是TensorFlow时，默认是NHWC。如果实际是NCHW的话，需要通过此参数指定NCHW。原始框架为Caffe时，只支持NCHW格式。
“转换输出节点”	指定输出节点，例如“node_name1:0;node_name1:1;node_name2:0”，其中“node_name”必须是模型转换前的网络模型中的节点名称，冒号后的数字表示第几个输出，例如“node_name1:0”，表示节点名称为“node_name1”的第0个输出。如果不指定输出节点，则模型的输出默认为最后一层的算子信息，某些情况下，用户想要查看某层算子参数是否合适，则需要将该层算子的参数输出，即可以在模型转换时通过该参数指定输出某层算子。转换时系统会解析输入模型获取输出节点并打印在日志中，如果不了解所使用模型的输入张量，可以参考日志中的解析结果。
“优选数据格式”	指定网络算子优先选用的数据格式，“ND（N<=4）”和“5D”。仅在网络中算子的输入数据同时支持“ND”和“5D”两种格式时，指定该参数才会生效。“ND”表示模型中算子按NCHW转换成通用格式，“5D”表示模型中算子按华为自研的5维转换成华为格式。“5D”为默认值。
“生成高精度模型”	指定是否生成高精度FP16 Davinci模型。“0”为默认值，表示生成普通FP16 Davinci模型，推理性能更好。“1”表示生成高精度FP16 Davinci模型，推理精度更好。高精度当前仅支持Caffe算子（Convolution、Pooling、FullConnection）和TensorFlow算子（tf.nn.conv2d、tf.nn.max_poo）。
“网络输出数据类型”	FP32为默认值，推荐分类网络、检测网络使用。图像超分辨率网络，推荐使用UINT8，推理性能更好。

TF-SavedModel-To-Ascend-C32

转换Tensorflow框架训练并以“saved_model”格式保存的模型，转换后模型可在Ascend上运行。转换时支持使用基于TE（Tensor Engine）开发的自定义算子(TE算子)。

 版权所有 © 华为技术有限公司

表 9-11 支持自定义算子转换模板的高级选项

参数名称	参数解释
“输入张量形状”	模型输入数据的shape，如“input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2”。“input_name”必须是转换前的网络模型中的节点名称。当模型存在动态shape输入时必须提供。例如“input_name1:?,h,w,c”，该参数必填，其中“?”为batch数，表示1次处理的图片数量，需要根据实际情况填写，用于将动态shape的原始模型转换为固定shape的离线模型。目前不支持批量特性，转换输入张量形状batch只能为1。转换时系统会解析输入模型获取输入张量并打印在日志中，如果不了解所使用模型的输入张量，可参考日志中的解析结果。
“输入数据格式”	支持NCHW和NHWC，默认是NHWC。当原始框架是TensorFlow时，默认是NHWC。如果实际是NCHW的话，需要通过此参数指定NCHW。原始框架为Caffe时，只支持NCHW格式。
“转换输出节点”	指定输出节点，例如“node_name1:0;node_name1:1;node_name2:0”，其中“node_name”必须是模型转换前的网络模型中的节点名称，冒号后的数字表示第几个输出，例如“node_name1:0”，表示节点名称为“node_name1”的第0个输出。如果不指定输出节点，则模型的输出默认为最后一层的算子信息，某些情况下，用户想要查看某层算子参数是否合适，则需要将该层算子的参数输出，即可以在模型转换时通过该参数指定输出某层算子。转换时系统会解析输入模型获取输出节点并打印在日志中，如果不了解所使用模型的输入张量，可以参考日志中的解析结果。
“优选数据格式”	指定网络算子优先选用的数据格式，“ND（N<=4）”和“5D”。仅在网络中算子的输入数据同时支持“ND”和“5D”两种格式时，指定该参数才会生效。“ND”表示模型中算子按NCHW转换成通用格式，“5D”表示模型中算子按华为自研的5维转换成华为格式。“5D”为默认值。
“生成高精度模型”	指定是否生成高精度FP16 Davinci模型。“0”为默认值，表示生成普通FP16 Davinci模型，推理性能更好。“1”表示生成高精度FP16 Davinci模型，推理精度更好。高精度当前仅支持Caffe算子（Convolution、Pooling、FullConnection）和TensorFlow算子（tf.nn.conv2d、tf.nn.max_poo）。
“网络输出数据类型”	FP32为默认值，推荐分类网络、检测网络使用。图像超分辨率网络，推荐使用UINT8，推理性能更好。

TF-FrozenGraph-To-Ascend-HiLens

主要提供给华为HiLens使用，当HiLens Kit系统固件版本为2.2.200.011时建议使用此模板进行转换。支持将TensorFlow frozen_graph模型转换成可在ascend芯片上运行的模型，转换时支持使用基于TE(Tensor Engine)开发的自定义算子。

 版权所有 © 华为技术有限公司

表 9-12 支持自定义算子转换模板的高级选项

参数名称	参数解释
“输入张量形状”	模型输入数据的shape，如“input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2”。“input_name”必须是转换前的网络模型中的节点名称。当模型存在动态shape输入时必须提供。例如“input_name1:?,h,w,c”，该参数必填，其中“?”为batch数，表示1次处理的图片数量，需要根据实际情况填写，用于将动态shape的原始模型转换为固定shape的离线模型。目前不支持批量特性，转换输入张量形状batch只能为1。转换时系统会解析输入模型获取输入张量并打印在日志中，如果不了解所使用模型的输入张量，可参考日志中的解析结果。
“输入数据格式”	支持NCHW和NHWC，默认是NHWC。当原始框架是TensorFlow时，默认是NHWC。如果实际是NCHW的话，需要通过此参数指定NCHW。原始框架为Caffe时，只支持NCHW格式。
“转换输出节点”	指定输出节点，例如“node_name1:0;node_name1:1;node_name2:0”，其中“node_name”必须是模型转换前的网络模型中的节点名称，冒号后的数字表示第几个输出，例如“node_name1:0”，表示节点名称为“node_name1”的第0个输出。如果不指定输出节点，则模型的输出默认为最后一层的算子信息，某些情况下，用户想要查看某层算子参数是否合适，则需要将该层算子的参数输出，即可以在模型转换时通过该参数指定输出某层算子。转换时系统会解析输入模型获取输出节点并打印在日志中，如果不了解所使用模型的输入张量，可以参考日志中的解析结果。
“优选数据格式”	指定网络算子优先选用的数据格式，“ND（N<=4）”和“5D”。仅在网络中算子的输入数据同时支持“ND”和“5D”两种格式时，指定该参数才会生效。“ND”表示模型中算子按NCHW转换成通用格式，“5D”表示模型中算子按华为自研的5维转换成华为格式。“5D”为默认值。
“生成高精度模型”	指定是否生成高精度FP16 Davinci模型。“0”为默认值，表示生成普通FP16 Davinci模型，推理性能更好。“1”表示生成高精度FP16 Davinci模型，推理精度更好。高精度当前仅支持Caffe算子（Convolution、Pooling、FullConnection）和TensorFlow算子（tf.nn.conv2d、tf.nn.max_poo）。
“网络输出数据类型”	FP32为默认值，推荐分类网络、检测网络使用。图像超分辨率网络，推荐使用UINT8，推理性能更好。

TF-SavedModel-To-Ascend-HiLens

主要提供给华为HiLens使用，当HiLens Kit系统固件版本为2.2.200.011时建议使用此模板进行转换。支持将TensorFlow saved_model模型转换成可在ascend芯片上运行的模型，转换时支持使用基于TE(Tensor Engine)开发的自定义算子。

 版权所有 © 华为技术有限公司

表 9-13 支持自定义算子转换模板的高级选项

参数名称	参数解释
“输入张量形状”	模型输入数据的shape，如“input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2”。“input_name”必须是转换前的网络模型中的节点名称。当模型存在动态shape输入时必须提供。例如“input_name1:?,h,w,c”，该参数必填，其中“?”为batch数，表示1次处理的图片数量，需要根据实际情况填写，用于将动态shape的原始模型转换为固定shape的离线模型。目前不支持批量特性，转换输入张量形状batch只能为1。转换时系统会解析输入模型获取输入张量并打印在日志中，如果不了解所使用模型的输入张量，可参考日志中的解析结果。
“输入数据格式”	支持NCHW和NHWC，默认是NHWC。当原始框架是TensorFlow时，默认是NHWC。如果实际是NCHW的话，需要通过此参数指定NCHW。原始框架为Caffe时，只支持NCHW格式。
“转换输出节点”	指定输出节点，例如“node_name1:0;node_name1:1;node_name2:0”，其中“node_name”必须是模型转换前的网络模型中的节点名称，冒号后的数字表示第几个输出，例如“node_name1:0”，表示节点名称为“node_name1”的第0个输出。如果不指定输出节点，则模型的输出默认为最后一层的算子信息，某些情况下，用户想要查看某层算子参数是否合适，则需要将该层算子的参数输出，即可以在模型转换时通过该参数指定输出某层算子。转换时系统会解析输入模型获取输出节点并打印在日志中，如果不了解所使用模型的输入张量，可以参考日志中的解析结果。
“优选数据格式”	指定网络算子优先选用的数据格式，“ND（N<=4）”和“5D”。仅在网络中算子的输入数据同时支持“ND”和“5D”两种格式时，指定该参数才会生效。“ND”表示模型中算子按NCHW转换成通用格式，“5D”表示模型中算子按华为自研的5维转换成华为格式。“5D”为默认值。
“生成高精度模型”	指定是否生成高精度FP16 Davinci模型。“0”为默认值，表示生成普通FP16 Davinci模型，推理性能更好。“1”表示生成高精度FP16 Davinci模型，推理精度更好。高精度当前仅支持Caffe算子（Convolution、Pooling、FullConnection）和TensorFlow算子（tf.nn.conv2d、tf.nn.max_poo）。
“网络输出数据类型”	FP32为默认值，推荐分类网络、检测网络使用。图像超分辨率网络，推荐使用UINT8，推理性能更好。

 版权所有 © 华为技术有限公司