昇腾创新实践课

MindSpore基础操作

实验手册



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2021。 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  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png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： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 注意事项 3](#_Toc101542903)

[1.1 关键流程 3](#_Toc101542904)

[1.2 ModelArts创建MindSpore实验环境注意事项 3](#_Toc101542905)

[2 实验环境介绍 6](#_Toc101542906)

[2.1 实验介绍 6](#_Toc101542907)

[2.1.1 关于本实验 6](#_Toc101542908)

[2.1.2 实验环境介绍 6](#_Toc101542909)

[3 MindSpore基础操作 7](#_Toc101542910)

[3.1 实验介绍 7](#_Toc101542911)

[3.1.1 关于本实验 7](#_Toc101542912)

[3.1.2 实验目的 7](#_Toc101542913)

[3.1.3 背景知识 7](#_Toc101542914)

[3.1.4 实验设计 7](#_Toc101542915)

[3.2 实验过程 8](#_Toc101542916)

[3.2.1 张量和数据类型 8](#_Toc101542917)

[3.2.2 数据集加载 11](#_Toc101542918)

[3.2.3 全连接网络搭建 12](#_Toc101542919)

[3.2.4 模型训练与评估 14](#_Toc101542920)

[3.3 实验总结 16](#_Toc101542921)

[4 手写数字识别 17](#_Toc101542922)

[4.1 手写数字识别实践 17](#_Toc101542923)

[4.1.1 关于本实践 17](#_Toc101542924)

[4.1.2 配置运行信息 17](#_Toc101542925)

[4.1.3 下载数据集 17](#_Toc101542926)

[4.1.4 数据处理 19](#_Toc101542927)

[4.1.5 创建模型 20](#_Toc101542928)

[4.1.6 优化模型参数 21](#_Toc101542929)

[4.1.7 训练及保存模型 22](#_Toc101542930)

[4.1.8 加载模型 23](#_Toc101542931)

[4.1.9 验证模型 24](#_Toc101542932)

# 注意事项

## 关键流程

1. 华为云官网( https://www.huaweicloud.com/）注册华为云账号并登录，完成实名认证

2．代金券查看，具体参考华为云使用手册.pdf中3.2部分



在图上右上边的菜单里的费用中心，可以查看自己的优惠券额度，注意查看自己的优惠券余额。



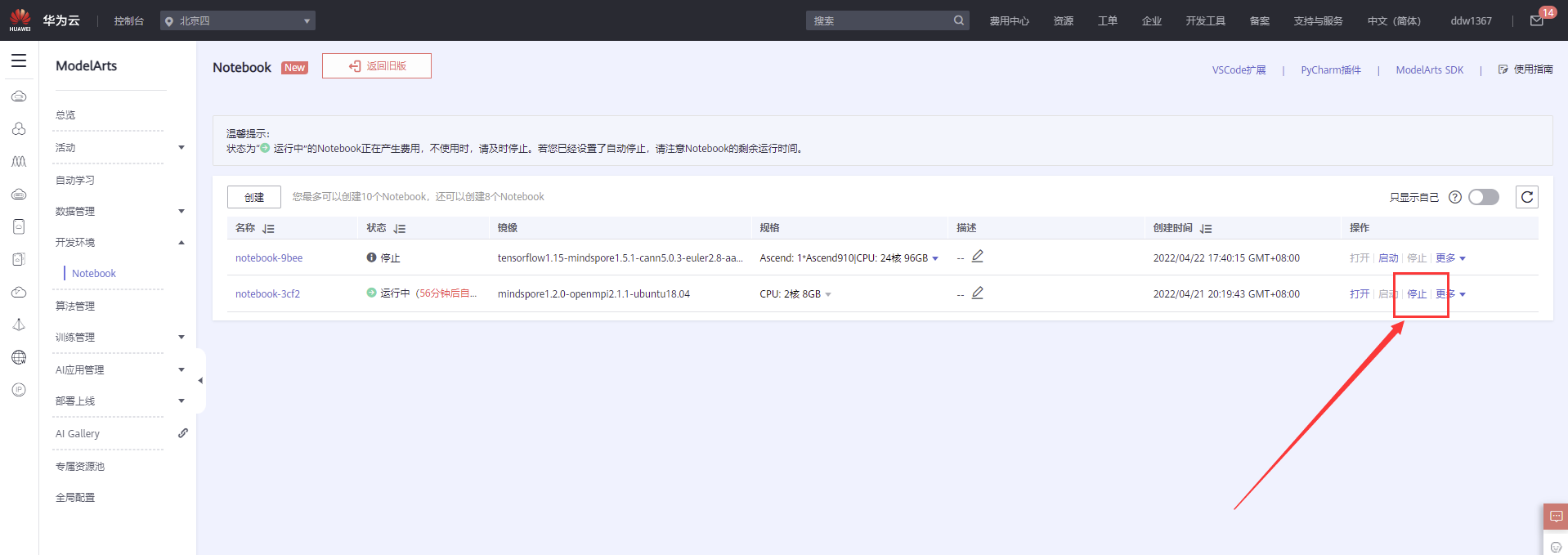
3．配置访问授权（ https://console.huaweicloud.com/modelarts/?region=cn-north-4#/dashboard )（使用委托），具体参考华为云使用手册.pdf中1.3.1部分

4．使用ModelArts创建MindSpore实验环境，具体参考华为云使用手册.pdf中3部分，涉及到的ModelArts请查看华为云使用手册.pdf中2部分

5．不再使用ModelArts,一定要停止收费，否则会一直计费

( https://support.huaweicloud.com/modelarts..faq/modelarts...05...0.109.html )请检查在ModelArts所创建运行中的作业，并停止或删除相关作业（重点查看

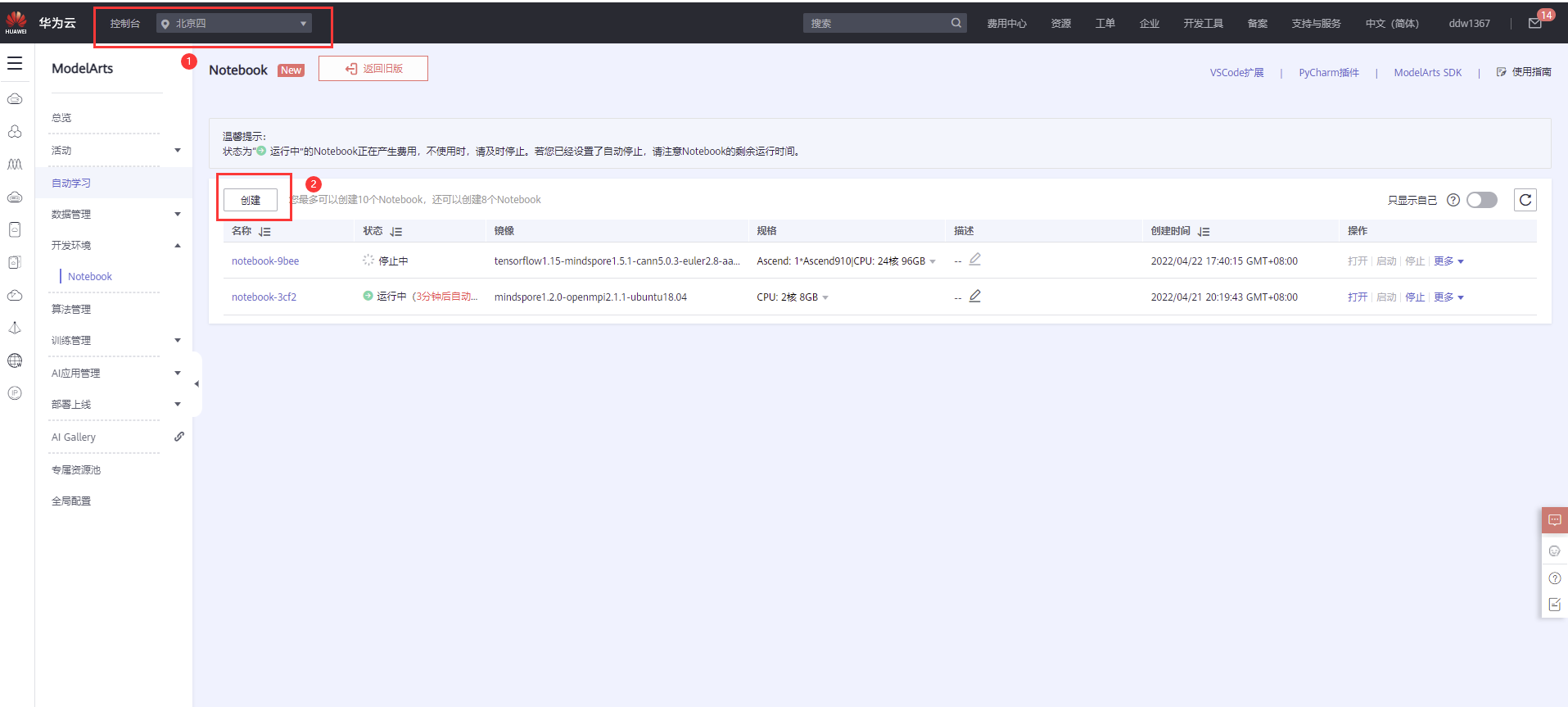
Notebook等），即可停止计费。由于ModelArts的数据存储在OBS中，请前往OBS服务删除对应数据和目录，停止计费。



PS：注意要在使用完毕后，关闭资源，点击页面上的停止按钮，关闭资源使用。如果不关闭，最后导致优惠券使用完毕，会出现欠费的情况，造成损失。且欠费状态下是无法使用优惠券的。

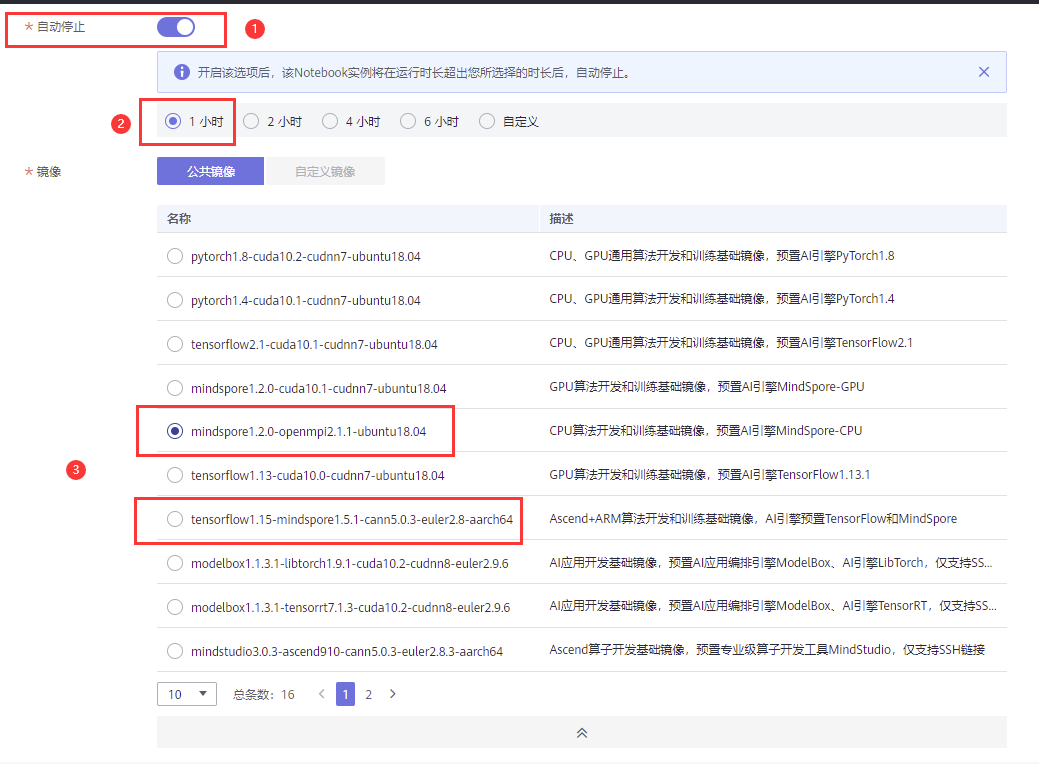
## ModelArts创建MindSpore实验环境注意事项

其中对于我们要创建的 MindSpore环境，这里需要特别注意

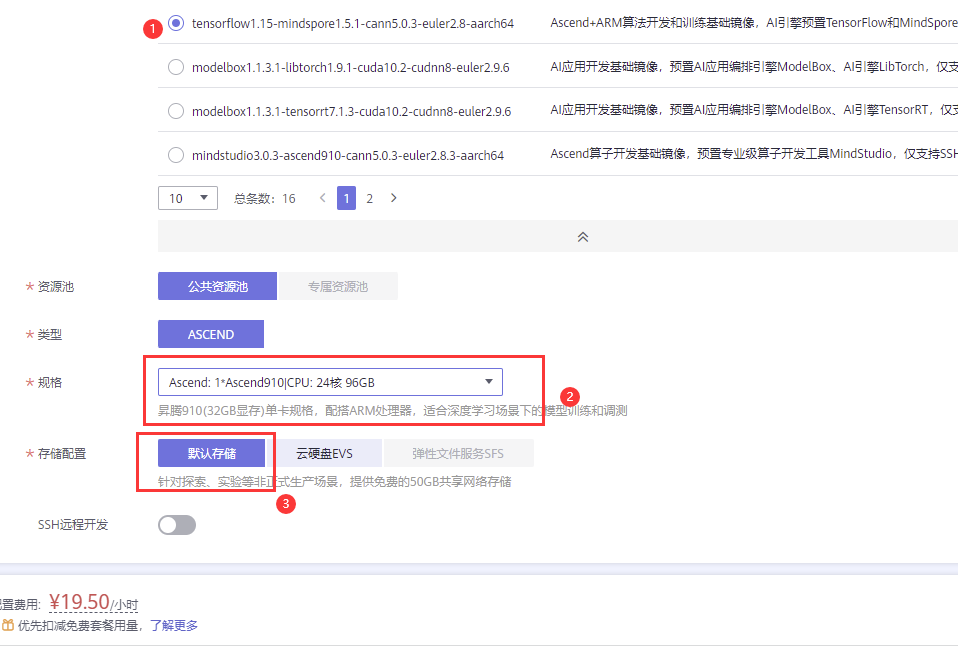


1.首先在创建的页面，左上角，确保是北京四。

2.点击创建进入页面后



1. 页面上1处的自动停止一定要勾选上。
2. 2处的时间建议选1-2小时
3. 重点：
4. 对于3处，也就是我们要选什么配置的服务器。这里框选了2个MindSpore的环境，上面一个只提供cpu，没有提供GPU或者华为自己的显卡，但是价格非常便宜，0.9/h，这里建议大家都先选第上面那个CPU版本的。等实验流程学完，代码无报错的时候再去创建含有显卡的下面的环境。



1. 当选择GPU的服务器的时候记得规格哪里选择96G的版本，
2. 存储当时选择默认存储。

# 实验环境介绍

## 实验介绍

### 关于本实验

MindSpore是一个全场景深度学习框架，旨在实现易开发、高效执行、全场景覆盖三大目标，其中易开发表现为API友好、调试难度低，高效执行包括计算效率、数据预处理效率和分布式训练效率，全场景则指框架同时支持云、边缘以及端侧场景。

本课程将介绍MindSpore的基础操作，通过学习本课程，学员将掌握MindSpore的基础模块功能，以及如何用MindSpore搭建简单神经网络的必备知识。

更多MindSpore基础操作练习，请查阅官网教程：<https://www.mindspore.cn/tutorials/zh-CN/r1.5/index.html>

### 实验环境介绍

实验、介绍、难度、软件环境、硬件环境：

实验环境介绍

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 实验介绍 | 难度 | 软件环境 | 开发环境 |
| MindSpore基础操作 | 介绍MindSpore的基础模块，搭建神经网络的必备知识 | 简单 | Python3.7 MindSpore1.5 | PC机 |

实验环境主要通过华为云的ModelArts平台使用华为研发的深度学习框架MindSpore。

如果有兴趣，且喜欢MindSpore这个框架，可以在自己电脑上安装。

# MindSpore基础操作

## 实验介绍

### 关于本实验

本实验通过介绍MindSpore的数据结构与数据类型，MindSpore搭建神经网络用到的基础模块，比如数据集加载，神经网络搭建，模型训练与评估等，让学员熟悉MindSpore的基础用法，掌握MindSpore开发的简单流程。

### 实验目的

理解MindSpore开发基本流程。

理解MindSpore基础模块的功能。

掌握MindSpore的简单操作。

### 背景知识

神经网络知识，感知机，多层感知机，前向传播，反向传播，激活函数，损失函数，优化器，评估方法。

### 实验设计

1.张量和数据类型

2.数据集加载

3.全连接网络搭建

4.模型训练

5.模型评估

## 实验过程

### 张量和数据类型

张量（Tensor）是MindSpore网络运算中的基本数据结构。张量中的数据类型可参考dtype。

不同维度的张量分别表示不同的数据，0维张量表示标量，1维张量表示向量，2维张量表示矩阵，3维张量可以表示彩色图像的RGB三通道等等。

MindSpore张量支持不同的数据类型，包含int8、int16、int32、int64、uint8、uint16、uint32、uint64、float16、float32、float64、bool\_，与NumPy的数据类型一一对应。

在MindSpore的运算处理流程中，Python中的int数会被转换为定义的int64类型，float数会被转换为定义的float32类型。

指定MindSpore数据类型

导入MindSpore，设置Jupyter notebook的cell同时输出多行。

# 导入MindSpore

import mindspore

from mindspore import dtype

from mindspore import Tensor

# cell同时输出多行

from IPython.core.interactiveshell import InteractiveShell

InteractiveShell.ast\_node\_interactivity = "all"

指定数据类型。

# 指定数据类型

a = 1

type(a)

b = Tensor(a, dtype.float64)

b.dtype

输出：

int

mindspore.float64

张量构造

构造张量时，支持传入Tensor、float、int、bool、tuple、list和NumPy.array类型，其中tuple和list里只能存放float、int、bool类型数据。

Tensor初始化时，可指定dtype。如果没有指定dtype，初始值int、float、bool分别生成数据类型为mindspore.int32、mindspore.float32、mindspore.bool\_的0维Tensor， 初始值tuple和list生成的1维Tensor数据类型与tuple和list里存放的数据类型相对应，如果包含多种不同类型的数据，则按照优先级：bool < int < float，选择相对优先级最高类型所对应的mindspore数据类型。 如果初始值是Tensor，则生成的Tensor数据类型与其一致；如果初始值是NumPy.array，则生成的Tensor数据类型与之对应。

用数组创建张量

代码：

import numpy as np

from mindspore import Tensor

# 用数组创建张量

x = Tensor(np.array([[1, 2], [3, 4]]), dtype.int32)

x

输出：

Tensor(shape=[2, 2], dtype=Int32, value=

[[1, 2],

[3, 4]])

用数值创建张量

代码：

# 用数值创建张量

y = Tensor(1.0, dtype.int32)

z = Tensor(2, dtype.int32)

y

z

输出：

Tensor(shape=[], dtype=Int32, value= 1)

Tensor(shape=[], dtype=Int32, value= 2)

用Bool创建张量

代码：

# 用Bool创建张量

m = Tensor(True, dtype.bool\_)

m

输出：

Tensor(shape=[], dtype=Bool, value= True)

用tuple创建张量

代码：

# 用tuple创建张量

n = Tensor((1, 2, 3), dtype.int16)

n

输出：

Tensor(shape=[3], dtype=Int16, value= [1, 2, 3])

用list创建张量

代码：

# 用list创建张量

p = Tensor([4.0, 5.0, 6.0], dtype.float64)

p

输出：

Tensor(shape=[3], dtype=Float64, value= [4.00000000e+000, 5.00000000e+000, 6.00000000e+000]

用常量创建张量

代码：

# 用常量创建张量

q = Tensor(1, dtype.float64)

q

输出：

Tensor(shape=[], dtype=Float64, value= 1)

张量的属性

张量的属性包括形状（shape）和数据类型（dtype）。

形状：Tensor的shape，是一个tuple。

数据类型：Tensor的dtype，是MindSpore的一个数据类型。

代码：

x = Tensor(np.array([[1, 2], [3, 4]]), dtype.int32)

x.shape # 形状

x.dtype # 数据类型

x.ndim # 维度

x.size # 大小

输出：

(2, 2)

mindspore.int32

2

4

张量的方法

asnumpy()：将Tensor转换为NumPy的array。

代码：

y = Tensor(np.array([[True, True], [False, False]]), dtype.bool\_)

# 将Tensor数据类型转换成NumPy

y\_array = y.asnumpy()

y

y\_array

输出：

Tensor(shape=[2, 2], dtype=Bool, value=

[[ True, True],

[False, False]])

array([[ True, True],

[False, False]])

### 数据集加载

MindSpore.dataset提供API来加载和处理各种常见的数据集，如MNIST, CIFAR-10, CIFAR-100, VOC, ImageNet, CelebA等。

加载MNIST数据集

下载MNIST数据集并在开发环境解压：<https://zhuanyejianshe.obs.cn-north-4.myhuaweicloud.com/chuangxinshijianke/cv-nlp/MNIST.zip>

mindspore.dataset.MnistDataset

代码：

import os

import mindspore.dataset as ds

import matplotlib.pyplot as plt

dataset\_dir = "./data/train" # 数据集路径

# 从mnist dataset读取3张图片

mnist\_dataset = ds.MnistDataset(dataset\_dir=dataset\_dir, num\_samples=3)

# 设置图像大小

plt.figure(figsize=(8,8))

i = 1

# 打印3张子图

for dic in mnist\_dataset.create\_dict\_iterator(output\_numpy=True):

plt.subplot(3,3,i)

plt.imshow(dic['image'][:,:,0])

plt.axis('off')

i +=1

plt.show()

输出：



MindSpore还支持加载多种数据存储格式下的数据集，用户可以直接使用mindspore.dataset中对应的类加载磁盘中的数据文件。

加载NumPy数据集

mindspore.dataset.NumpySlicesDataset

代码：

import mindspore.dataset as ds

data = ds.NumpySlicesDataset([1, 2, 3], column\_names=["col\_1"])

for x in data.create\_dict\_iterator():

print(x)

输出：

{'col\_1': Tensor(shape=[], dtype=Int32, value= 2)}

{'col\_1': Tensor(shape=[], dtype=Int32, value= 3)}

{'col\_1': Tensor(shape=[], dtype=Int32, value= 1)}

### 全连接网络搭建

全连接神经网络

全连接层

mindspore.nn.Dense

in\_channels：输入通道

out\_channels：输出通道

weight\_init：权重初始化，Default 'normal'.

代码：

import mindspore.nn as nn

from mindspore import Tensor

# 构造输入张量

input = Tensor(np.array([[1, 1, 1], [2, 2, 2]]), mindspore.float32)

print(input)

# 构造全连接网络，输入通道为3，输出通道为3

net = nn.Dense(in\_channels=3, out\_channels=3, weight\_init=1)

output = net(input)

print(output)

输出：

[[1. 1. 1.]

[2. 2. 2.]]

[[3. 3. 3.]

[6. 6. 6.]]

激活函数

矫正线性单元激活函数

mindspore.nn.ReLU

代码：

input\_x = Tensor(np.array([-1, 2, -3, 2, -1]), mindspore.float16)

relu = nn.ReLU()

output = relu(input\_x)

print(output)

输出：

[0. 2. 0. 2. 0.]

搭建模型

所有神经网络的基类

mindspore.nn.Cell

代码：

import mindspore.nn as nn

class MyCell(nn.Cell):

# 定义算子

def \_\_init\_\_(self, ):

super(MyCell, self).\_\_init\_\_()

# 全连接层

self.fc = nn.Dense()

# 激活函数

self.relu = nn.ReLU()

# 建构网络

def construct(self, x):

x = self.fc(x)

x = self.relu(x)

return x

### 模型训练与评估

损失函数

交叉熵损失函数，用于分类模型。当标签数据不是one-hot编码形式时，需要输入参数sparse为True。

mindspore.nn.SoftmaxCrossEntropyWithLogits

代码：

import mindspore.nn as nn

# 交叉熵损失函数

loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)

np.random.seed(0)

logits = Tensor(np.random.randint(0, 9, [1, 10]), mindspore.float32)

print(logits)

labels\_np = np.ones([1,]).astype(np.int32)

labels = Tensor(labels\_np)

print(labels)

output = loss(logits, labels)

print(output)

输出：

[[5. 0. 3. 3. 7. 3. 5. 2. 4. 7.]]

[1]

[7.868383]

优化器

深度学习优化算法大概常用的有SGD、Adam、Ftrl、lazyadam、Momentum、RMSprop、Lars、Proximal\_ada\_grad和lamb这几种。

动量优化器

mindspore.nn.Momentum

代码：

# optim = nn.Momentum(params, learning\_rate=0.1, momentum=0.9, weight\_decay=0.0)

模型编译

mindspore.Model

network：神经网络

loss\_fn：损失函数

optimizer：优化器

metrics：评估指标

代码：

from mindspore import Model

# 定义神经网络

net = nn.Dense(in\_channels=3, out\_channels=3, weight\_init=1)

# 定义损失函数

loss = nn.SoftmaxCrossEntropyWithLogits()

# 定义优化器

optim = nn.Momentum(params=net.trainable\_params(), learning\_rate=0.1, momentum=0.9)

# 模型编译

model = Model(network = net, loss\_fn=loss, optimizer=optim, metrics=None)

模型训练

model.train

epoch：训练次数

train\_dataset ：训练集

代码：

# model.train(epoch=10, train\_dataset=train\_dataset) # train\_dataset 是传入参数

模型评估

model.eval

valid\_dataset：测试集

# model.eval(valid\_dataset=test\_dataset) # test\_dataset 是传入参数

## 实验总结

本实验介绍了MindSpore的数据结构与类型，以及MindSpore搭建神经网络用到的基础模块，让学员学会如何加载数据集，搭建神经网络，训练和评估模型等，从易到难，由浅入深，让学员熟悉MindSpore的基础用法，掌握MindSpore开发的简单流程。

# 手写数字识别

## 手写数字识别实践

### 关于本实践

本节贯穿第二节的MindSpore的基础功能，实现深度学习中的常见任务，完整的深度学习中的常见任务的流程。从配置运行信息，下载数据，数据处理，创建模型，优化模型参数，训练及保存模型，加载模型，验证模型等步骤。

### 配置运行信息

MindSpore通过context.set\_context来配置运行需要的信息，如运行模式、后端信息、硬件等信息。

代码：

import os

import argparse

from mindspore import context

parser = argparse.ArgumentParser(description='MindSpore LeNet Example')

parser.add\_argument('--device\_target', type=str, default="CPU", choices=['Ascend', 'GPU', 'CPU'])

args = parser.parse\_known\_args()[0]

context.set\_context(mode=context.GRAPH\_MODE, device\_target=args.device\_target)

在样例中，我们配置样例运行使用图模式。根据实际情况配置硬件信息，譬如代码运行在Ascend AI处理器上，则--device\_target选择Ascend，代码运行在CPU、GPU同理。详细参数说明，请参见https://www.mindspore.cn/docs/api/zh-CN/r1.6/api\_python/mindspore.context.html

### 下载数据集

我们示例中用到的MNIST数据集是由10类28∗28的灰度图片组成，训练数据集包含60000张图片，测试数据集包含10000张图片。

代码：

import os

import requests

requests.packages.urllib3.disable\_warnings()

def download\_dataset(dataset\_url, path):

filename = dataset\_url.split("/")[-1]

save\_path = os.path.join(path, filename)

if os.path.exists(save\_path):

return

if not os.path.exists(path):

os.makedirs(path)

res = requests.get(dataset\_url, stream=True, verify=False)

with open(save\_path, "wb") as f:

for chunk in res.iter\_content(chunk\_size=512):

if chunk:

f.write(chunk)

print("The {} file is downloaded and saved in the path {} after processing".format(os.path.basename(dataset\_url), path))

train\_path = "datasets/MNIST\_Data/train"

test\_path = "datasets/MNIST\_Data/test"

download\_dataset("https://mindspore-website.obs.myhuaweicloud.com/notebook/datasets/mnist/train-labels-idx1-ubyte", train\_path)

download\_dataset("https://mindspore-website.obs.myhuaweicloud.com/notebook/datasets/mnist/train-images-idx3-ubyte", train\_path)

download\_dataset("https://mindspore-website.obs.myhuaweicloud.com/notebook/datasets/mnist/t10k-labels-idx1-ubyte", test\_path)

download\_dataset("https://mindspore-website.obs.myhuaweicloud.com/notebook/datasets/mnist/t10k-images-idx3-ubyte", test\_path)

下载的数据集文件的目录结构如下：

./datasets/MNIST\_Data

├── test

│ ├── t10k-images-idx3-ubyte

│ └── t10k-labels-idx1-ubyte

└── train

├── train-images-idx3-ubyte

└── train-labels-idx1-ubyte

### 数据处理

数据集对于模型训练非常重要，好的数据集可以有效提高训练精度和效率。 MindSpore提供了用于数据处理的API模块 mindspore.dataset ，用于存储样本和标签。在加载数据集前，我们通常会对数据集进行一些处理，mindspore.dataset也集成了常见的数据处理方法。

首先导入MindSpore中mindspore.dataset和其他相应的模块

代码：

import mindspore.dataset as ds

import mindspore.dataset.transforms.c\_transforms as C

import mindspore.dataset.vision.c\_transforms as CV

from mindspore.dataset.vision import Inter

from mindspore import dtype as mstype

数据集处理主要分为四个步骤：

1.定义函数create\_dataset来创建数据集。

2.定义需要进行的数据增强和处理操作，为之后进行map映射做准备。

3.使用map映射函数，将数据操作应用到数据集。

4.进行数据shuffle、batch操作。

代码：

def create\_dataset(data\_path, batch\_size=32, repeat\_size=1,

num\_parallel\_workers=1):

# 定义数据集

mnist\_ds = ds.MnistDataset(data\_path)

resize\_height, resize\_width = 32, 32

rescale = 1.0 / 255.0

shift = 0.0

rescale\_nml = 1 / 0.3081

shift\_nml = -1 \* 0.1307 / 0.3081

# 定义所需要操作的map映射

resize\_op = CV.Resize((resize\_height, resize\_width), interpolation=Inter.LINEAR)

rescale\_nml\_op = CV.Rescale(rescale\_nml, shift\_nml)

rescale\_op = CV.Rescale(rescale, shift)

hwc2chw\_op = CV.HWC2CHW()

type\_cast\_op = C.TypeCast(mstype.int32)

# 使用map映射函数，将数据操作应用到数据集

mnist\_ds = mnist\_ds.map(operations=type\_cast\_op, input\_columns="label", num\_parallel\_workers=num\_parallel\_workers)

mnist\_ds = mnist\_ds.map(operations=[resize\_op, rescale\_op, rescale\_nml\_op, hwc2chw\_op], input\_columns="image", num\_parallel\_workers=num\_parallel\_workers)

# 进行shuffle、batch、repeat操作

buffer\_size = 10000

mnist\_ds = mnist\_ds.shuffle(buffer\_size=buffer\_size)

mnist\_ds = mnist\_ds.batch(batch\_size, drop\_remainder=True)

mnist\_ds = mnist\_ds.repeat(count=repeat\_size)

return mnist\_ds

其中，batch\_size为每组包含的数据个数，现设置每组包含32个数据。

### 创建模型

使用MindSpore定义神经网络需要继承mindspore.nn.Cell。Cell是所有神经网络（如Conv2d-relu-softmax等）的基类。

神经网络的各层需要预先在\_\_init\_\_方法中定义，然后通过定义construct方法来完成神经网络的前向构造。按照LeNet的网络结构，定义网络各层如下：

代码：

import mindspore.nn as nn

from mindspore.common.initializer import Normal

class LeNet5(nn.Cell):

"""

Lenet网络结构

"""

def \_\_init\_\_(self, num\_class=10, num\_channel=1):

super(LeNet5, self).\_\_init\_\_()

# 定义所需要的运算

self.conv1 = nn.Conv2d(num\_channel, 6, 5, pad\_mode='valid')

self.conv2 = nn.Conv2d(6, 16, 5, pad\_mode='valid')

self.fc1 = nn.Dense(16 \* 5 \* 5, 120, weight\_init=Normal(0.02))

self.fc2 = nn.Dense(120, 84, weight\_init=Normal(0.02))

self.fc3 = nn.Dense(84, num\_class, weight\_init=Normal(0.02))

self.relu = nn.ReLU()

self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

self.flatten = nn.Flatten()

def construct(self, x):

# 使用定义好的运算构建前向网络

x = self.conv1(x)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.conv2(x)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.relu(x)

x = self.fc2(x)

x = self.relu(x)

x = self.fc3(x)

return x

# 实例化网络

net = LeNet5()

### 优化模型参数

要训练神经网络模型，需要定义损失函数和优化器。

MindSpore支持的损失函数有SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss等。这里使用交叉熵损失函数SoftmaxCrossEntropyWithLogits。

代码：

# 定义损失函数

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

MindSpore支持的优化器有Adam、AdamWeightDecay、Momentum等。这里使用Momentum优化器为例。

代码：

# 定义优化器

net\_opt = nn.Momentum(net.trainable\_params(), learning\_rate=0.01, momentum=0.9)

### 训练及保存模型

MindSpore提供了回调Callback机制，可以在训练过程中执行自定义逻辑，这里以使用框架提供的ModelCheckpoint为例。 ModelCheckpoint可以保存网络模型和参数，以便进行后续的Fine-tuning（微调）操作。

代码：

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig

# 设置模型保存参数

config\_ck = CheckpointConfig(save\_checkpoint\_steps=1875, keep\_checkpoint\_max=10)

# 应用模型保存参数

ckpoint = ModelCheckpoint(prefix="checkpoint\_lenet", config=config\_ck)

通过MindSpore提供的model.train接口可以方便地进行网络的训练，LossMonitor可以监控训练过程中loss值的变化。

代码：

# 导入模型训练需要的库

from mindspore.nn import Accuracy

from mindspore.train.callback import LossMonitor

from mindspore import Model

def train\_net(model, epoch\_size, data\_path, repeat\_size, ckpoint\_cb, sink\_mode):

"""定义训练的方法"""

# 加载训练数据集

ds\_train = create\_dataset(os.path.join(data\_path, "train"), 32, repeat\_size)

model.train(epoch\_size, ds\_train, callbacks=[ckpoint\_cb, LossMonitor(125)], dataset\_sink\_mode=sink\_mode)

其中，dataset\_sink\_mode用于控制数据是否下沉，数据下沉是指数据通过通道直接传送到Device上，可以加快训练速度，dataset\_sink\_mode为True表示数据下沉，否则为非下沉。

通过模型运行测试数据集得到的结果，验证模型的泛化能力。

1.使用model.eval接口读入测试数据集。

2.使用保存后的模型参数进行推理。

代码：

def test\_net(model, data\_path):

"""定义验证的方法"""

ds\_eval = create\_dataset(os.path.join(data\_path, "test"))

acc = model.eval(ds\_eval, dataset\_sink\_mode=False)

print("{}".format(acc))

这里把train\_epoch设置为1，对数据集进行1个迭代的训练。在train\_net和 test\_net方法中，我们加载了之前下载的训练数据集，mnist\_path是MNIST数据集路径。

代码：

train\_epoch = 1

mnist\_path = "./datasets/MNIST\_Data"

dataset\_size = 1

model = Model(net, net\_loss, net\_opt, metrics={"Accuracy": Accuracy()})

train\_net(model, train\_epoch, mnist\_path, dataset\_size, ckpoint, False)

test\_net(model, mnist\_path)

训练过程中会打印loss值，类似下图。loss值会波动，但总体来说loss值会逐步减小，精度逐步提高。每个人运行的loss值有一定随机性，不一定完全相同。 训练过程中loss打印示例如下：

epoch: 1 step: 125, loss is 2.3083377

epoch: 1 step: 250, loss is 2.3019726

epoch: 1 step: 1500, loss is 0.028385757

epoch: 1 step: 1625, loss is 0.0857362

epoch: 1 step: 1750, loss is 0.05639569

epoch: 1 step: 1875, loss is 0.12366105

{'Accuracy': 0.9663477564102564}

可以在打印信息中看出模型精度数据，示例中精度数据达到96.6%，模型质量良好。随着网络迭代次数train\_epoch增加，模型精度会进一步提高。

### 加载模型

将训练好的模型加载，用于test

代码：

from mindspore import load\_checkpoint, load\_param\_into\_net

# 加载已经保存的用于测试的模型

param\_dict = load\_checkpoint("checkpoint\_lenet-1\_1875.ckpt")

# 加载参数到网络中

load\_param\_into\_net(net, param\_dict)

### 验证模型

我们使用生成的模型进行单个图片数据的分类预测，具体步骤如下：（被预测的图片会随机生成，每次运行结果可能会不一样。）

代码：

import numpy as np

from mindspore import Tensor

# 定义测试数据集，batch\_size设置为1，则取出一张图片

ds\_test = create\_dataset(os.path.join(mnist\_path, "test"), batch\_size=1).create\_dict\_iterator()

data = next(ds\_test)

# images为测试图片，labels为测试图片的实际分类

images = data["image"].asnumpy()

labels = data["label"].asnumpy()

# 使用函数model.predict预测image对应分类

output = model.predict(Tensor(data['image']))

predicted = np.argmax(output.asnumpy(), axis=1)

# 输出预测分类与实际分类

print(f'Predicted: "{predicted[0]}", Actual: "{labels[0]}"')

验证结果：

Predicted: "6", Actual: "6"