智能芯片原理与应用

实验指导书

实验二：MindSpore 编程框架使用

华为智能基座课程

广东工业大学

计算机学院

2024 年 11 月

1. 实验介绍

1.1 实验介绍

1.1.1 关于本实验

实验前理论课应该完成深度学习编程框架的讲解，学生已了解、基于 Tensorflow 或其他深度学习编程框架的基本原理。本实验介绍了 Window/Linux 系统通过 Miniconda 安装 MindSpore、搭建 python 虚拟环境，类比 MindSpore、PyTorch 和 Tensorflow 的编程框架的语法区别。

1.1.2 实验目标

1. 熟悉 MindSpore 基本使用，包括 tensor、数据加载及数据处理（选做）、模型搭建、模型训练与测试、模型保存与加载等，了解 MindSpore 命令式编程，并完成相关代码。
2. 通过具体操作、熟悉相关类、方法等，并通过核心代码段、实现结果截图等记录实验结果。

1.1.3 软件版本介绍

类别	版本	获取方式	说明
Windows	**Windows11、10**	**/**	**需要是 64 位系统，CPU 支持 AVX2 指令集**
Ubuntu	**Ubuntu 22.04 / 18.04.4**	**https://ubuntu.com/download/desktop**	**需要是 64 位系统，CPU 支持 AVX2 指令集**
PyCharm	**2020.1.4 Community Edition**	**https://www.jetbrains.com/**	**/ vscode 二选一**
VScode	**1.77**	**https://code.visualstudio.com/download**	**/ pycharm 二选一**

Miniconda	**Python3.x**	**官方下载地址：** **https://docs.conda.io/en/latest/miniconda.html** **清华镜像源地址：** **https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/**	**Miniconda 可在线安装不同的 Python 版本，无需刻意下载特定版本，但需要下载 64 位，Python3.x 版本**

其中 Pycharm 和 VScode 任选其一。

1.2 软件介绍

1.2.1 MindSpore 深度学习编程框架

昇思 MindSpore 是华为公司开发的一个全场景开源深度学习框架，旨在实现易开发、高效执行、全场景覆盖三大目标。用于支持深度学习、自然语言处理、图像处理等多个领域的应用开发，图 1 展示了 MindSpore 深度学习框架的总体结构。其特点如下：

1. 全场景覆盖：MindSpore 支持多种设备、多种场景下的深度学习模型开发和发布，包括 CPU、GPU、Ascend 等处理器，同时也支持分布式和端到端联合训练。
2. 非透明自动微分：MindSpore 支持动态图，它可以自动微分，而且支持自动微分和数字信号处理等各方面技术。
3. 安全保障机制：MindSpore 针对最近出现的深度学习模型安全问题，加入了多种安全保障机制。
4. 速度快、易用性高：MindSpore 使用静态图作为开发模式，能够让用户更方便地进行模型优化，在保证速度的同时还能提高易用性。

MindSpore 的使用场景包括：

自然语言处理：MindSpore 能够帮助开发者快速构建和训练基于自然语言处理的模型，例如机器翻译、语音识别等。

图像处理：MindSpore 提供了丰富的算法支持，能够帮助开发者轻

松构建和训练图像识别、图像分类等模型。

数据分析和预测：MindSpore 能够处理多维向量、矩阵、张量等数据，能够支持数据分析和预测等工作。

MindSpore 支持的语言包括 Python、C++等多种编程语言，在 Python 中几乎可以实现全部功能，极大程度提高了开发效率。

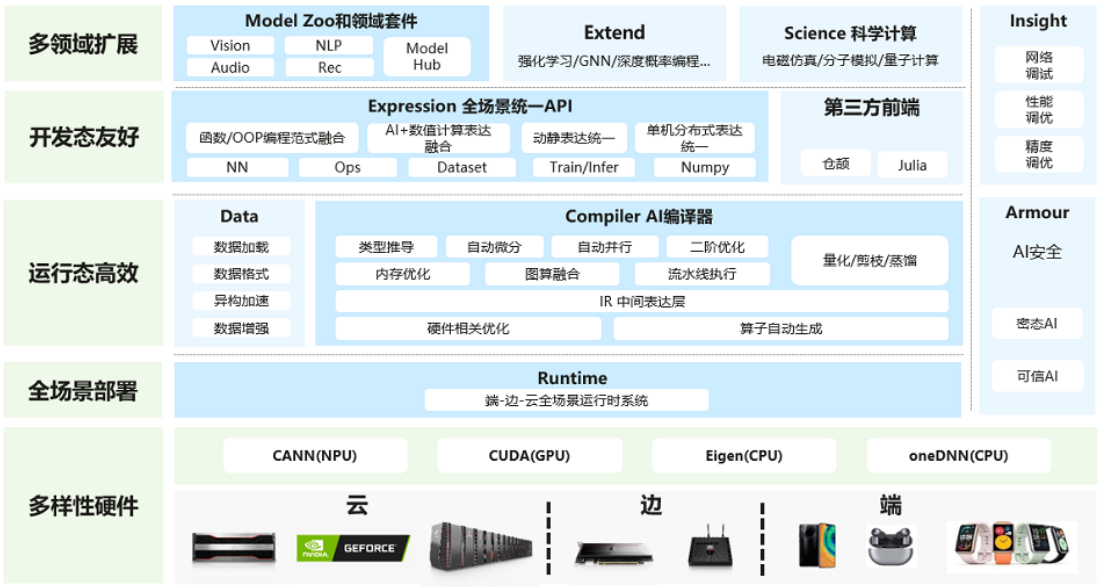


图 1 昇思 MindSpore 总体架构[1]

1.2.2 Miniconda 介绍

Miniconda[2]是一个轻量级的 Anaconda 版本，它包括了 Python 解释器和 conda 包管理器，但只包含少量的基本包和库。它的主要目的是为了让用户能够更加灵活地配置自己的 Python 环境，同时避免 Anaconda 安装过多的软件包和库，造成磁盘空间和性能的浪费。

Conda 是一个跨平台的包管理器和环境管理器，它可以用于安装、升级和删除软件包，并且可以创建和管理多个独立的 Python 环境。

[1] https://www.mindspore.cn/tutorials/zh-CN/r2.0/beginner/introduction.html

[2] https://docs.conda.io/en/latest/miniconda.html

Conda 可以在 Linux、Windows 和 MacOS 等操作系统上运行，并且可以管理不同版本的 Python 和各种依赖库。Conda 还可以用于创建和共享自定义软件包，方便其他用户进行安装和使用。

2 环境搭建

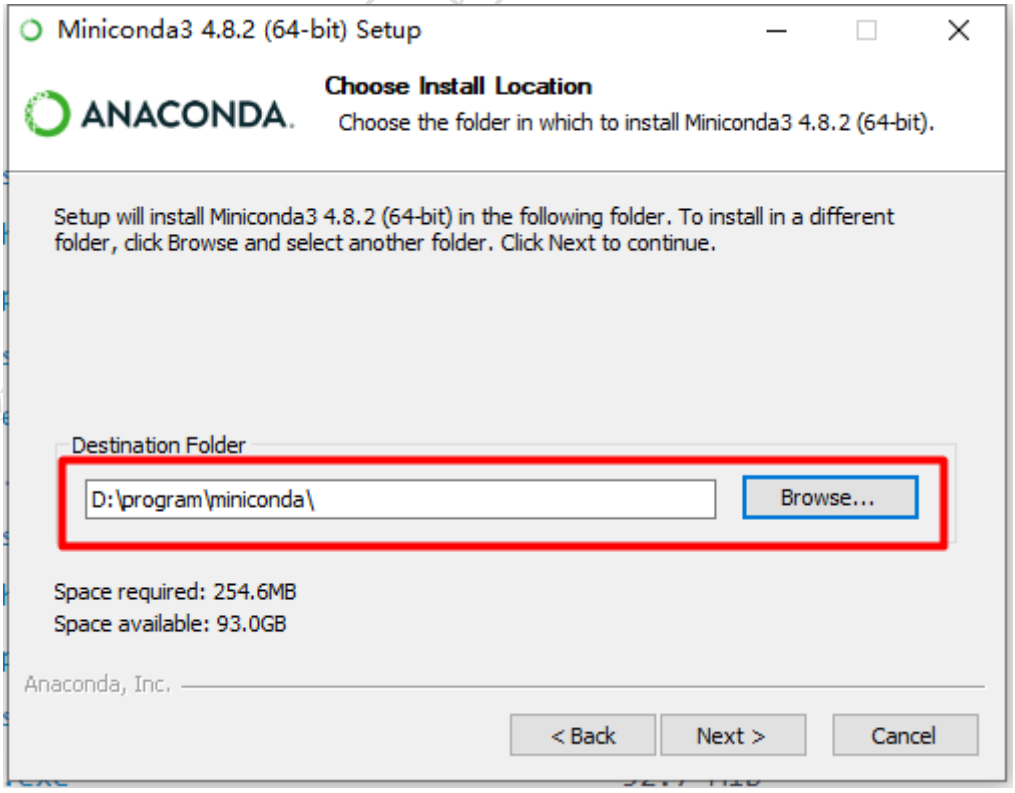
2.1 以本地 Windows 环境搭建为例（本次实验推荐利用 Virtual Box 安装 Ubuntu18.04/20.04/22.04 Linux 系统）

2.1.1 Miniconda 安装

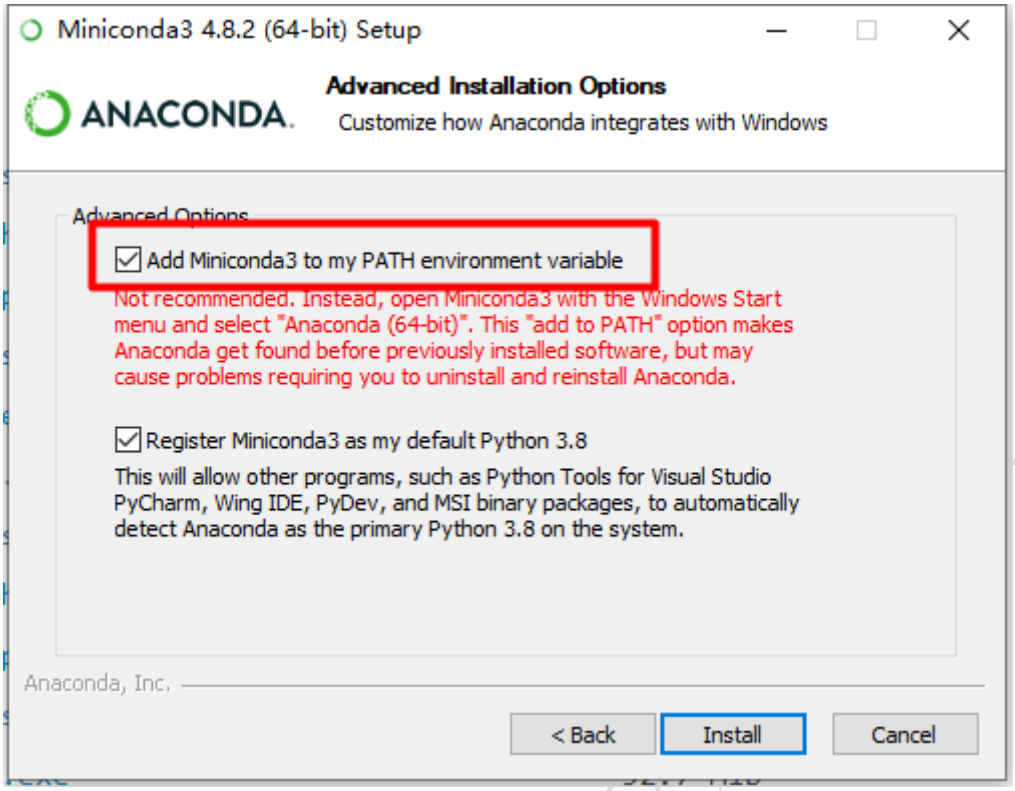
从 1.1.2 提供的链接下载 Miniconda 的 Windows 版本对应的 64 位安装包，由于官方源下载速度慢，实验所用安装包为清华源下载，带有 x86_64 的为 64 位安装包。

Miniconda3-py38_4.8.2-Windows-x86_64.exe 52.7 MiB 2020-03-12 00:09

双击安装包进行安装，点击 next，然后选择安装位置。



环境变量打勾，这样可以直接在命令行中启动 Miniconda。

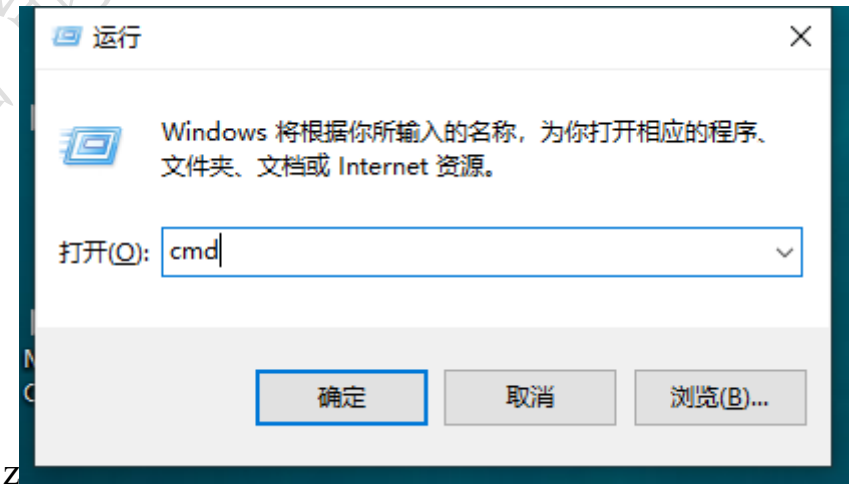


等待安装成功，然后点击 Finish。

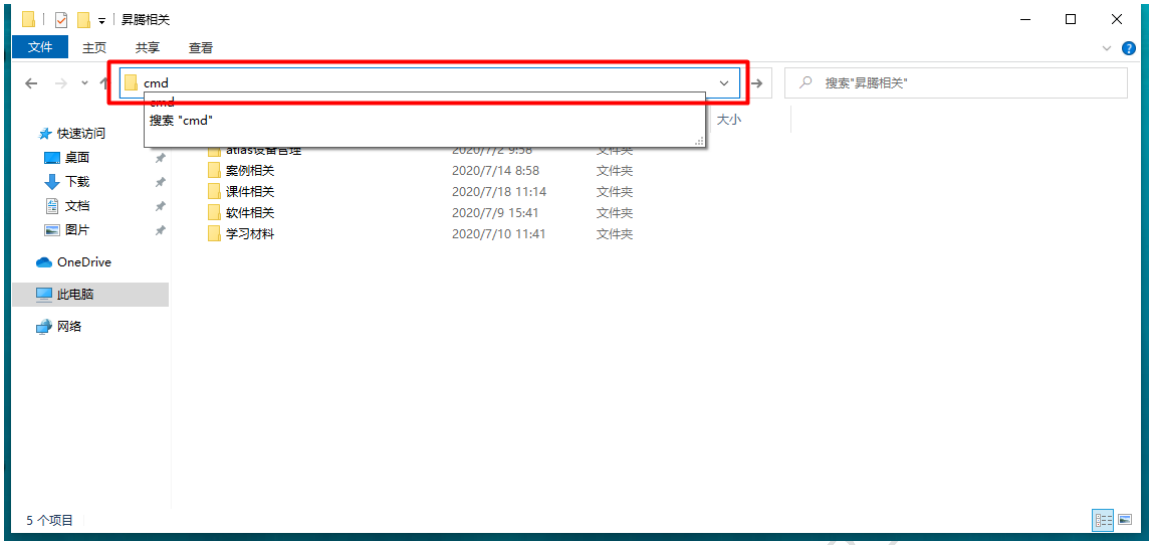
2.1.2 创建虚拟环境

在 Window 中有多种方式开启命令行窗口，这里介绍两种，按下 win+R 键，然后输入 cmd 点击确定，或者任意打开一个文件夹，在上方地址栏输入 cmd，然后按回车键。

运行打开命令行界面如下图所示：



地址栏打开命令行界面如下图所示：



打开命令行窗口之后，输入以下命令创建虚拟环境，Python 版本为 3.7.5，创建过程需要输入 y 确认。

>> conda create -n hwms python==3.8

虚拟环境创建成功后输入对应名称即可进入对应虚拟环境

>> activate hwms

2.1.3 Pip 换源（非必做步骤）

Python 可以通过 pip 和 conda 两种方式来安装包，但是两者所安装的包并不完全兼容，在实际使用过程中建议只选择一种方式来安装包，本实验使用的是 pip，但是由于 pip 的官方源在国外，直连速度较慢，因此需要换为国内的镜像源。

方法一：更换清华源

打开命令行，激活虚拟环境，利用如下命令进行设置。

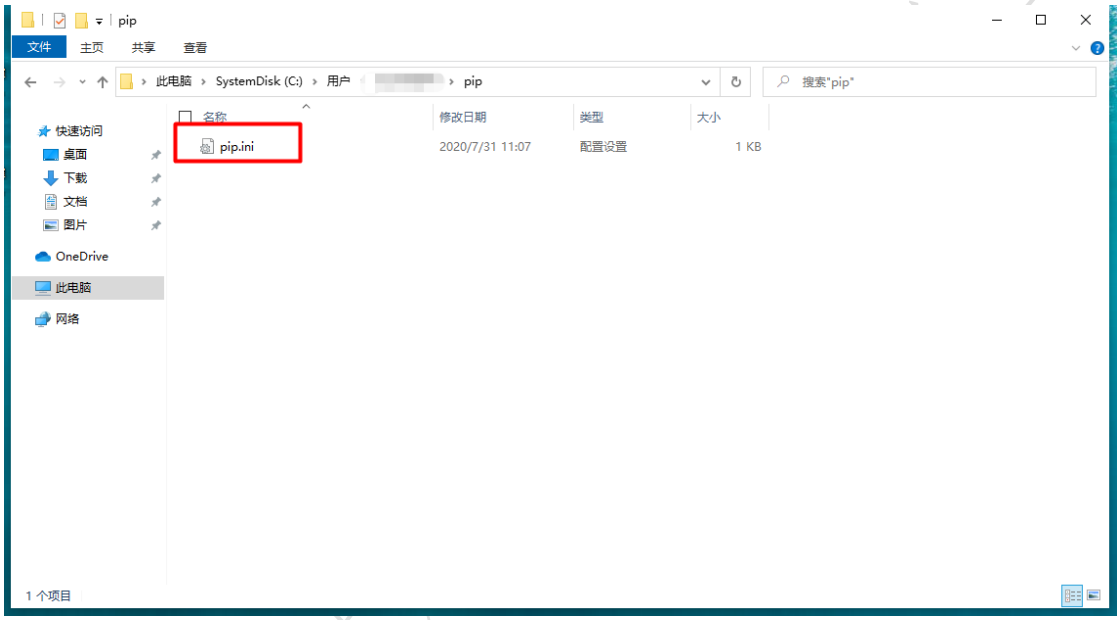
```
# 首先更新 pip 版本
python -m pip install --upgrade pip
# 设定 pip 源全局设定，注意下面命令没有换行
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

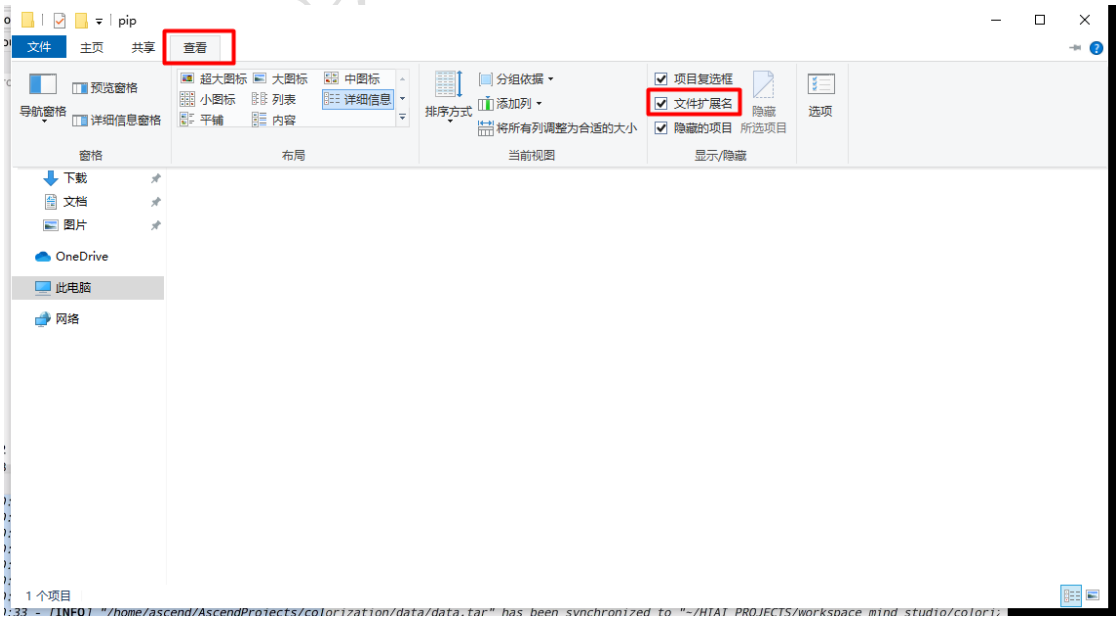
执行成功后显示如下结果：

Writing to C:\Users\<UserName>\AppData\Roaming\pip\pip.ini

方法二：手动设置清华源

打开此电脑在 C:\Users\<UserName>\AppData\Roaming\下，新建一个 pip 文件夹。新建一个文本文件，然后改名 pip.ini，该文件就是 pip 的配置文件，如果改完之后图标没变化，说明没有显示文件扩展名，点击查看，随后勾选显示文件扩展名。





打开 pip.ini 文件，将以下内容粘贴进去并保存。

[global]

index-url = https://pypi.tuna.tsinghua.edu.cn/simple

2.1.4 安装 MindSpore（详细步骤参见 1.MindSpore 环境搭建实验手册.docx）

首先在终端中，输入以下命令激活 hwms 安装虚拟环境。

(Ubuntu/Linux)

>> conda activate hwms

(Windows)

>> activate hwms

成功激活环境后，利用 conda 命令安装最新版本的 Mindspore 库，通过查阅官网，根据所需设备查阅安装方式，例如：

图 2 选择不同版本安装不同的 Mindspore 库

具体可以查阅网站 https://www.mindspore.cn/install

>> conda install mindspore=2.0.0a0 -c mindspore -c conda-forge

安装完毕后，可在命令行验证安装是否成功

>> python

>>> import mindspore

安装正常一般会不出现任何显示

注意：若同学有 GPU 设备，可查阅，mindspore-gpu 库的安装方式，但要注意 CUDA develop toolkit 库的兼容版本。Mindspore2.0 版本针对新型号显卡，需要安装 cudatoolkit 10.1, 11.1 或 11.6 版本，例如：

>> conda install cudatoolkit==11.1

3 MindSpore 的基本使用

3.1 张量的定义与使用（熟悉 jupyter notebook 的同学，可用 notebook 操作相关命令）

实验要求：结合示例代码，可尝试改变示例代码相关内容与配置，运行结果，并截图记录。

1. 首先激活 hwms 虚拟环境，确保成功激活，命令行窗口前会显示‘(hwms)’字样。之后，执行 python 命令，进入 python 命令行界面。

2. 引入 mindspore 库。具体代码参看 mindspore_tensor.py 文件

```
>>>import numpy as np
>>>import mindspore
>>>from mindspore import ops
>>>from mindspore import Tensor, CSRTensor, COOTensor
```

若成功引入，通常无任何显示。

张量(Tensor)的概念对于理解深度学习框架中的数据处理和构成是非常重要的。通常，对于计算机视觉任务来说，接触最多的是图像类型数据，通常会出现四维张量，即为(N，H，W，C)维度的 tensor。利用上述张量。再 Mindspore 中张量的使用与 Tensorflow 类似，主要分以下几个方面：

A. 创建张量：张量的创建方式有多种，构造张量时，支持传入`Tensor`、`float`、`int`、`bool`、`tuple`、`list`和`numpy.ndarray`类型。

1）由数据直接生成：

```
>>>data = [1, 0, 1, 0]
>>>x_data = Tensor(data)
```

2）由 Numpy 数字生成：

```
>>>np_array = np.array(data)
>>>x_np = Tensor(np_array)
```

3）使用 init 初始化器构造张量

当使用 init 初始化器对张量进行初始化时，支持传入的参数有 init、shape、dtype，其中：

init: 支持传入 initializer 的子类。

shape: 支持传入 list、tuple、 int。

dtype: 支持传入 mindspore.dtype。

```
from mindspore.common.initializer import One, Normal
# 初始化一个全 1 的 tensor
tensor1 = mindspore.Tensor(shape=(2, 2), dtype=mindspore.float32,
init=One())
```

```
# 初始化一个符合标准正态分布的 Tensor
tensor2 = mindspore.Tensor(shape=(2, 2), dtype=mindspore.float32,
init=Normal())
# 打印查看结果(print 函数 python 的 debug 神器)
print("tensor1:\n", tensor1)
print("tensor2:\n", tensor2)
```

4）继承另外一个 Tensor 的大小、类型，例如申请全零和全 1 张量时

```
from mindspore import ops
x_ones = ops.ones_like(x_data)
print(f"Ones Tensor: \n {x_ones} \n")
x_zeros = ops.zeros_like(x_data)
print(f"Zeros Tensor: \n {x_zeros} \n")
```

B. 查看张量属性

张量的属性包括形状、数据类型、转置张量、单个元素大小、占用字节数量、维数、元素个数和每一维步长。可通过查看如下属性，显示具体类型，例如：

- 形状（shape）：Tensor 的 shape，是一个 tuple。
- 数据类型（dtype）：Tensor 的 dtype，是 MindSpore 的一个数据类型。
- 单个元素大小（itemsize）： Tensor 中每一个元素占用字节数，是一个整数。
- 占用字节数量（nbytes）： Tensor 占用的总字节数，是一个整数。
- 维数（ndim）： Tensor 的秩，也就是 len(tensor.shape)，是一个整数。
- 元素个数（size）： Tensor 中所有元素的个数，是一个整数。
- 每一维步长（strides）： Tensor 每一维所需要的字节数，是一个 tuple。

```
# 可尝试其他数据类型和大小,查看返回的维度等信息
x = Tensor(np.array([[1, 2], [3, 4]]), mindspore.int32)
print("x_shape:", x.shape)
print("x_dtype:", x.dtype)
print("x_itemsize:", x.itemsize)
print("x_nbytes:", x.nbytes)
print("x_ndim:", x.ndim)
print("x_size:", x.size)
print("x_strides:", x.strides)
```

C. 张量如何索引

即如何获得具位置、范围 Tensor 中的数值。Tensor 索引与 Numpy 索引类似，索引从 0 开始编制，负索引表示按倒序编制，冒号:和 ... 用于对数据进行切片。

```
# 尝试使用其他非示例代码,熟悉检索具体行、列、范围内的数值
tensor = Tensor(np.array([[0, 1], [2, 3]]).astype(np.float32))
print("First row: {}".format(tensor[0]))
print("value of bottom right corner: {}".format(tensor[1, 1]))
print("Last column: {}".format(tensor[:, -1]))
print("First column: {}".format(tensor[..., 0]))
```

D. 张量运算

张量之间有很多运算，包括算术、线性代数、矩阵处理（转置、标引、切片）、采样等，张量运算和 NumPy 的使用方式类似，下面介绍其中几种操作：

```
x = Tensor(np.array([1, 2, 3]), mindspore.float32)
y = Tensor(np.array([4, 5, 6]), mindspore.float32)
output_add = x + y
output_sub = x - y
output_mul = x * y
output_div = y / x
output_mod = y % x
output_floordiv = y // x

print("add:", output_add)
print("sub:", output_sub)
print("mul:", output_mul)
print("div:", output_div)
print("mod:", output_mod)
print("floordiv:", output_floordiv)
```

试查阅相关材料，实验完成转置、矩阵内积、外积运算等操作。

E. 其他张量操作

1）Tensor 级联 concat：通过指定具体的维度，进行级联

```
data1 = Tensor(np.array([[0, 1], [2, 3]]).astype(np.float32))
data2 = Tensor(np.array([[4, 5], [6, 7]]).astype(np.float32))
output = ops.concat((data1, data2), axis=0)
print(output)
print("shape:\n", output.shape)
```

2）Tensor 的堆叠：

```
data1 = Tensor(np.array([[0, 1], [2, 3]]).astype(np.float32))
data2 = Tensor(np.array([[4, 5], [6, 7]]).astype(np.float32))
output = ops.stack([data1, data2])
print(output)
print("shape:\n", output.shape)
```

3.2 MindSpore 网络构建

实验要求：结合示例代码，可尝试改变示例代码相关内容与配置，运行结果，并截图记录。

与 Tensorflow 类似，mindspore.nn 中提供了常见神经网络层的实现，在 MindSpore 中，Cell 类是构建所有网络的基类，也是网络的基本单元。一个神经网络模型表示为一个 Cell，它由不同的子 Cell 构成。使用这样的嵌套结构，可以简单地使用面向对象编程的思维，对神经网络结构进行构建和管理。构建前，需要引入 mindspore 的 nn。

1）定义模型类：

当定义神经网络时，可以继承 nn.Cell 类，在__init__方法中进行子 Cell 的实例化和状态管理，在 construct 方法中实现 Tensor 操作。

```
# 首先需要继承 nn.Cell 类
class Network(nn.Cell):
    def __init__(self):
        super().__init__()
        # 构造 784->512->512->10 的 MLP，思考能不能控制每层的参数量
        self.flatten = nn.Flatten()
```

```
        self.dense_relu_sequential = nn.SequentialCell(
            nn.Dense(28*28, 512),
            nn.ReLU(),
            nn.Dense(512, 512),
            nn.ReLU(),
            nn.Dense(512, 10)
        )
    # construct 函数类似 forward 函数,等价构建计算图
    def construct(self, x):
        x = self.flatten(x)
        logits = self.dense_relu_sequential(x)
        return logits
```

构建完成后，实例化 Network 对象，并查看其结构。

```
model = Network()
print(model)
```

默认动态图机制，随便输入一个输入结果，测试下模型输出：

```
X = ops.ones((1, 28, 28), mindspore.float32)
logits = model(X)
print(logits)
```

多类分类，通过一个 nn.Softmax 层实例来获得预测概率。

```
pred_probab = nn.Softmax(axis=1)(logits)
y_pred = pred_probab.argmax(1)
print(f"Predicted class: {y_pred}")
```

查看输出结果

2）模型层 Layers、或 Op：

对于每层的属性及其计算结果，通过命令式编程，可以测试其模型层的计算和设置方式。分解上节构造的神经网络模型中的每一层。首先我们构造一个 shape 为(3, 28, 28)的随机数据（3 个 28x28 的图像），依次通过每一个神经网络层来观察其效果。

```
input_image = ops.ones((3, 28, 28), mindspore.float32)
print(input_image.shape)
```

nn.Flatten: 实例化 nn.Flatten 层，将 28x28 的 2D 张量转换为 784 大小

的连续数组。

```
flatten = nn.Flatten()
flat_image = flatten(input_image)
print(flat_image.shape)
```

nn.Dense: 全连接层，其使用权重和偏差对输入进行线性变换。

```
layer1 = nn.Dense(in_channels=28*28, out_channels=20)
hidden1 = layer1(flat_image)
print(hidden1.shape)
```

nn.ReLU: 非线性的激活函数，帮助神经网络学习各种复杂的特征。

```
print(f"Before ReLU: {hidden1}\n\n")
hidden1 = nn.ReLU()(hidden1)
print(f"After ReLU: {hidden1}")
```

nn.SequentialCell: 是一个有序的 Cell 容器。输入 Tensor 将按照定义的顺序通过所有 Cell。我们可以使用 SequentialCell 来快速组合构造一个神经网络模型。(一种搭积木的形式)。

```
seq_modules = nn.SequentialCell(
    flatten,
    layer1,
    nn.ReLU(),
    nn.Dense(20, 10)
)

logits = seq_modules(input_image)
print(logits.shape)
```

nn.Softmax: 最后使用 nn.Softmax 将神经网络最后一个全连接层返回的 logits 的值缩放为[0, 1]，表示每个类别的预测概率。axis 指定的维度数值和为 1。

```
softmax = nn.Softmax(axis=1)
```

```
pred_probab = softmax(logits)
print(pred_probab)
```

3）模型参数

网络内部神经网络层具有权重参数和偏置参数（如 nn.Dense），这些参数会在训练过程中不断进行优化，可通过 model.parameters_and_names() 来获取参数名及对应的参数详情。

```
print(f"Model structure: {model}\n\n")

for name, param in model.parameters_and_names():
    print(f"Layer: {name}\nSize: {param.shape}\nValues : {param[:2]} \n")
```

等多 nn 库的 API，参见链接[3]，以及 mindspore 与 Pytorch API 的对应关系[4]。

4 模型训练

实验要求：结合示例代码，可尝试改变示例代码相关内容与配置，运行结果，并截图记录。

4.1 构造 MLP 进行模型训练和测试

模型训练一般分为四个步骤：

1. 构建数据集。
2. 定义神经网络模型。
3. 定义超参、损失函数及优化器。
4. 输入数据集进行训练与评估。

依然以 MLP 再 MNIST 数据集上训练为例，运行以下代码（mindspore_train.py）：

[3] https://www.mindspore.cn/docs/zh-CN/r2.0/api_python/mindspore.nn.html

[4] https://www.mindspore.cn/docs/zh-CN/r1.7/note/api_mapping/pytorch_api_mapping.html

```
#################### 0. 引入相关包 ########################
import mindspore
from mindspore import nn
from mindspore.dataset import vision, transforms
from mindspore.dataset import MnistDataset

# 下载数据，注意缺失 download 库，通过 pip install download 安装
from download import download
#################### 1. 准备数据 ############################
url = "https://mindspore-website.obs.cn-north-
4.myhuaweicloud.com/" \
      "notebook/datasets/MNIST_Data.zip"
path = download(url, "./", kind="zip", replace=True)
# 初始化数据加载管线,数据加载器
def datapipe(path, batch_size):
    image_transforms = [
        vision.Rescale(1.0 / 255.0, 0),
        vision.Normalize(mean=(0.1307,), std=(0.3081,)),
        vision.HWC2CHW()
    ]
    label_transform = transforms.TypeCast(mindspore.int32)

    dataset = MnistDataset(path)
    dataset = dataset.map(image_transforms, 'image')
    dataset = dataset.map(label_transform, 'label')
    dataset = dataset.batch(batch_size)
    return dataset

train_dataset = datapipe('MNIST_Data/train', batch_size=64)
test_dataset = datapipe('MNIST_Data/test', batch_size=64)

#################### 3. 定义模型 MLP ########################
class Network(nn.Cell):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.dense_relu_sequential = nn.SequentialCell(
```

```
            nn.Dense(28*28, 512),
            nn.ReLU(),
            nn.Dense(512, 512),
            nn.ReLU(),
            nn.Dense(512, 10)
        )
# 自定义时注意 构造 construct 方法，x 为输入，return 为输出
    def construct(self, x):
        x = self.flatten(x)
        logits = self.dense_relu_sequential(x)
        return logits
# 实例化一个 MLP
model = Network()
# 设定训练模型的超参数，这类参数与模型本身参数无关
epochs = 3
batch_size = 64
learning_rate = 1e-2
# 定义 Loss op
loss_fn = nn.CrossEntropyLoss()
# 定义优化器
optimizer = nn.SGD(model.trainable_params(),
learning_rate=learning_rate)
# 定义模型整体的 forward op
def forward_fn(data, label):
    logits = model(data)
    loss = loss_fn(logits, label)
return loss, logits

# Get gradient function
grad_fn = mindspore.value_and_grad(forward_fn, None,
optimizer.parameters, has_aux=True)

# Define function of one-step training
def train_step(data, label):
    (loss, _), grads = grad_fn(data, label)
    optimizer(grads)
    return loss
```

```
# 定义训练主循环
def train_loop(model, dataset):
    size = dataset.get_dataset_size()
    model.set_train()
    for batch, (data, label) in
enumerate(dataset.create_tuple_iterator(num_epochs=1)):
        loss = train_step(data, label)

        if batch % 100 == 0:
            # 每 100 次迭代输出以下 loss
            loss, current = loss.asnumpy(), batch
            print(f"loss: {loss:>7f}  [{current:>3d}/{size:>3d}]")

# 定义测试循环
def test_loop(model, dataset, loss_fn):
    num_batches = dataset.get_dataset_size()
    model.set_train(False)
    total, test_loss, correct = 0, 0, 0
    for data, label in
dataset.create_tuple_iterator(num_epochs=1):
        pred = model(data)
        total += len(data)
        test_loss += loss_fn(pred, label).asnumpy()
        correct += (pred.argmax(1) == label).asnumpy().sum()
    test_loss /= num_batches
    correct /= total
    print(f"Test: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:
{test_loss:>8f} \n")



for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(model, train_dataset)
    test_loop(model, test_dataset, loss_fn)
print("Done!")
```

4.2 模型存储与加载

在训练网络模型的过程中，需要保存中间和最后的结果，用于微调（fine-tune）和后续的模型推理与部署，本章节我们将介绍如何保存与加载模型。保存模型使用 save_checkpoint 接口，传入网络和指定的保存路径：

```
import numpy as np
import mindspore
from mindspore import nn
from mindspore import Tensor
def network():
    model = nn.SequentialCell(
                nn.Flatten(),
                nn.Dense(28*28, 512),
                nn.ReLU(),
                nn.Dense(512, 512),
                nn.ReLU(),
                nn.Dense(512, 10))
    return model
model = network()
mindspore.save_checkpoint(model, "model.ckpt") # 设定名模型名字
```

要加载模型权重，需要先创建相同模型的实例，然后使用 load_checkpoint 和 load_param_into_net 方法加载参数。

```
model = network()
param_dict = mindspore.load_checkpoint("model.ckpt")
param_not_load, _ = mindspore.load_param_into_net(model,
param_dict)
print(param_not_load)
```

实验 1：结合训练测试代码，以及保存、加载模型的操作，在 mindspore_train.py 中实现每一个 epoch，存一下模型，并且文件名跟随 epoch 数量变化（查看 python 的字符串格式化代码，以及 if 判断语句等）。

实验 2：查阅相关材料（mindspore 的 nn.Conv2d 等模型层），在 mindspore_train.py 中实现一个 CNN 模型，并在 MNIST 上训练和测试，对比前后的精度结果。

实验 3（拓展可选做）：选做 mindspore_dataset.py 和 mindspore_transforms.py 及 notebook 中的内容，运行或改变示例代码，熟悉数据集加载方式和数据增强方法的使用，可加入实验报告中。（save_load 代码中 MindIR 模式在 PC 端不支持加载，仅限云和昇腾系列计算设备上执行）