重庆邮电大学基于 MindSpore 框架的
手写数字识别实验报告

实验（实习）名称 基于 MindSpore 框架的手写数字识别实验

日期 2022.05.06 得分 指导教师 冯潇

系 计算机 专业 计算机科学与技术 年级 2021

班次 04912101 姓名 苏卓萱 学号 2021211818

一、实验目的

本实验以 MindSpore 框架为基础，实现深度学习中的常见任务，完整地学习深度学习任务流程，掌握配置运行信息，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。

二、实验任务

使用华为云平台资源完成《Python 程序设计》MindSpore 最终操作手册_20220423.docx 中的 2 个任务

1、MindSpore 基本操作

2、基于 MindSpore 的手写数字识别实验

三、实验步骤（参考 MindSpore 最终操作手册_20220423.docx，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，验证模型等步骤描述，要有运行截图）

任务一 MindSpore 基础操作

1.1.1 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。

步骤 1　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
```

步骤 2　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)
```

步骤 3 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
```

步骤 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]])
```

1.1.2 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。

步骤 1　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。

步骤 2　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)}
```

1.1.3 全连接网络搭建

步骤 1　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.]]
```

步骤 2 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.]
```

步骤 3 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
```

1.1.4 模型训练与评估

步骤 1 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]
```

步骤 2 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)
```

步骤 3　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)
```

步骤 4　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
```

步骤 5　模型评估

model.eval

valid_dataset：测试集

```
# model.eval(valid_dataset=test_dataset)    # test_dataset 是传入参数
```

任务二　手写数字识别

1.2 手写数字识别实践

1.2.1 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。

1.2.2 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

1.2.3 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
```

1.2.4 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。

1.2.5 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()
```

1.2.6 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)
```

1.2.7 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。

1.2.8 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)
```

1.2.9 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"

四、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。