山东大学 计算机科学与技术 学院

人工智能导论 课程实验报告

学号：202300130183	姓名： 宋浩宇	邮 箱 ： 202300130183 @mail.sdu.edu.cn

实验题目：八、手写数字识别与垃圾分类

实验过程：

（记录实验过程、遇到的问题和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

1. 首先完成数据集的下载，在 http://yann.lecun.com/exdb/mnist/网站下载标准 MNIST 数据集及标签。

2. 将 Ubyte 形式的数据转化为可视化的图片，此处使用了 pytorch 框架和 matplotlib 来实现图像的转化

```
mnist_train = datasets.MNIST(root=data_path, train=True,download=False)
mnist_test = datasets.MNIST(root=data_path, train=False,download=False)

# 遍历并保存训练集中的图像
for i, (image, label) in enumerate(mnist_train):
    # 创建一个新的图像并绘制MNIST图像
    plt.figure()
    plt.imshow(image, cmap='gray')
    plt.title('Label: {}'.format(label))
    plt.axis('off')
    time.sleep(0.00001)
    # 保存图像
    plt.savefig('F:\\DATASET\\MNIST\\image\\train_image\\mnist_train_{}.png'.format(i))
    plt.close()

# 遍历并保存测试集中的图像
for i, (image, label) in enumerate(mnist_test):
    # 创建一个新的图像并绘制MNIST图像
    plt.figure()
    plt.imshow(image, cmap='gray')
    plt.title('Label: {}'.format(label))
    plt.axis('off')
    time.sleep(0.00001)
    # 保存图像
    plt.savefig('F:\\DATASET\\MNIST\\image\\test_image\\mnist_test_{}.png'.format(i))
    plt.close()
```

此处因为 matplotlib 库中的 savefig 使用了 HTTP 协议通过本地回环网络 ip127.0.0.1 来把图片输出到本地，因此使用了 time 库的 sleep 函数来防止单位时间请求发送过多导致报错。

3. 为了实现神经网络实现 MNIST 手写数字识别，此处构建多层感知机的模型。

```
# 参数设置
n_epochs = 3                        #训练次数
batch_size_train = 64               #训练时单次抓取样本数
batch_size_test = 1000              #测试时单次抓取样本数
learning_rate = 0.01                #学习率
momentum = 0.5                      #梯度下降动量
log_interval = 10                   #多少个 batch 打印一次训练状态
lay1_node_cnt = 512                 #中间层1节点个数
lay2_node_cnt = 512                 #中间层2节点个数
```

4. 声明全局变量作为模型的超参数

```
# 初始化数据集
#加载
path="F:\\DATASET\\MNIST\\mnist_dataset_dir"
train_data = MnistDataset(path,shuffle=True)
test_data = MnistDataset(path,shuffle=True)
```

5. 对数据进行 transform 操作（由于使用的是 Windows 系统的 Mindspore，不支持 CUDA，所以此处没有将数据转成 tensor 对象）

```
# transforms部分
#更改数据类型(与实现transform对象等价)
composed = transforms.Compose(
    [
        vision.Rescale(1.0 / 255.0, shift: 0),
        vision.Normalize(mean=(0.1307,), std=(0.3081,)),
        vision.HWC2CHW()
    ]
)
train_data = train_data.map(operations=composed, input_columns="image")
# train_data = train_data.map(vision.ToTensor())
test_data = test_data.map(operations=composed,input_columns="image")
# test_data = test_data.map(vision.ToTensor())
print(type(train_data))
print(type(test_data))

image, label = next(train_data.create_tuple_iterator())
print(image.shape, image.dtype)
```

6. 定义损失函数

```python
# 损失函数
1 个用法
def loss_fn(logits, label):
    labels=[0 for i in range(10)]
    labels[label]=1
    logits = Net(logits)
    pred_probab = nn.Softmax(axis=1)(logits)
    print(pred_probab)
    ans = pred_probab.argmax(1)
    print(f"Predicted class: {ans}")
    loss=0
    print(logits[0])
    for i in range(len(logits[0])):
        loss+=(labels[i]-logits[0][i])**2
    print("loss=",loss)
    return loss
```

损失函数使用了目标值与预测值的差的平方的和作为标准，此处还涉及到概率的放缩，即将预测向量的和放缩到 1

7. 构建网络

```python
class Network(nn.Cell):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        #相邻两层的节点之间均为全链接
        self.dense_relu_sequential = nn.SequentialCell(
            *args: nn.Dense(28*28, lay1_node_cnt, weight_init="normal", bias_init="zeros"),
            nn.ReLU(),
            nn.Dense(lay1_node_cnt, lay2_node_cnt, weight_init="normal", bias_init="zeros"),
            nn.ReLU(),
            nn.Dense(lay2_node_cnt, out_channels: 10, weight_init="normal", bias_init="zeros")
        )

    def construct(self, x):
        x = self.flatten(x)
        logits = self.dense_relu_sequential(x)
        return logits
Net = Network()
Net.set_train(train_data)
```

该网络一共有四层，输入层接受一个 784 维的向量，即图像的点阵，隐含层有两层，分别由 lay1_node_cnt 和 lay2_node_cnt 设置，输出层输出一个十维的向量，代表网络的预测概率

8. 定义梯度下降函数

```python
# 梯度函数
def accuracy(function,x,y):
    grad_fn = mindspore.value_and_grad(function, grad_position: None, weights=Net.trainable_params())
    loss, grads = grad_fn(x,y)
    return loss.asnumpy()
```

该函数返回一个 numpy 数组，里边存储的是损失函数对于网络中每一个参数的偏导数，function 参数接受损失函数，x，y 分别是训练数据和网络的输出结果，此处利用到了 mindspore 的自动微分功能。

9. 训练模型

```python
def train_step(data, label):
    (loss, _), grads = grad_fn(data, label)
    optimizer(grads)
    return loss

1 个用法
def train_loop(model, dataset):
    size = dataset.get_dataset_size()
    model.set_train()
    for batch, (data, label) in enumerate(dataset.create_tuple_iterator()):
        loss = train_step(data, label)

        if batch % 100 == 0:
            loss, current = loss.asnumpy(), batch
            print(f"loss: {loss:>7f}  [{current:>3d}/{size:>3d}]")
1 个用法
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():
        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")
loss_fn = nn.CrossEntropyLoss()
optimizer = nn.SGD(Net.trainable_params(), learning_rate=learning_rate)

for i in range(n_epochs):
    print(f"Epoch {i+1}\n-------------------------------")
    train_loop(Net, train_data)
    test_loop(Net, test_data, loss_fn)
```

此处用到了 mindspore 官方网站帮助档里给的样例代码，简要来说就是对该数据集进行 n_epochs 次学习，每次挑出 batch_size_train 个数据来进行学习，参数的调整由梯度下降函数返回的 numpy 数组和学习率决定，在学习完一次后在按照 batch_size_test 设置的一次测试的数据来计算损失函数的平均值。完成学习循环之后模型即为训练完毕。

10. 储存模型

```python
mindspore.save_checkpoint(Net, ckpt_file_name: "model.ckpt")
```

11. 在另一段程序中加载模型

```python
class Network(nn.Cell):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        #相邻两层的节点之间均为全链接
        self.dense_relu_sequential = nn.SequentialCell(
            *args: nn.Dense(28*28, lay1_node_cnt, weight_init="normal", bias_init="zeros"),
            nn.ReLU(),
            nn.Dense(lay1_node_cnt, lay2_node_cnt, weight_init="normal", bias_init="zeros"),
            nn.ReLU(),
            nn.Dense(lay2_node_cnt, out_channels: 10, weight_init="normal", bias_init="zeros")
        )

    def construct(self, x):
        x = self.flatten(x)
        logits = self.dense_relu_sequential(x)
        return logits

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

结果分析与体会：

华为的 mindspore 框架为人工智能应用的开发提供了许多方便的工具，而且还提供了许多云计算服务。本身昇思框架也有着开源的开发者社区，有着丰富的简体中文学习资源。但是框架的性能相对于 tensorflow 和 pytorch 等存在更久的框架还有些距离，而且对于 windows 系统的 cuda 开发支持不够，导致 windows 上跑模型会用不上 nvidia 显卡的加速，只能靠 CPU，希望华为能早日将该框架的 windows 系统的 cuda 版发布出来。顺便一提这个网络只是一个多层感知机模型，它的中间层是完全的黑箱，另外 mindspore 也提供了卷积层较为简单的实现方式，也可以使用卷积神经网络来优化这个学习过程。