华为昇腾-昇思MindSpore《机器学习原理》精品课程

机器学习原理

精品课程

实验指导手册

版本:1.0



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

[17. 基于MindSpore构建Dropout层 4](#_Toc134174664)

[17.1. 实验介绍 4](#_Toc134174665)

[17.1.1. Dropout知识点的介绍 4](#_Toc134174666)

[17.1.2. 数据集介绍 5](#_Toc134174667)

[17.2. 实验环境要求 5](#_Toc134174668)

[17.3. 实验详细设计与实现 6](#_Toc134174669)

[17.3.1. 数据准备 6](#_Toc134174670)

[17.3.2. 实验步骤 6](#_Toc134174671)

[17.4. 实验总结 13](#_Toc134174672)

[17.5. 实验任务与参考解答任务 13](#_Toc134174673)

[17.5.1. 实验任务 13](#_Toc134174674)

[17.5.2. 参考答案 13](#_Toc134174675)

# 基于MindSpore构建Dropout层

## 实验介绍

本实验主要介绍使用MindSpore深度学习框架中使用Dropout层，并对比Dropout层对深度学习模型性能的影响,实验流程如下：

* 处理数据符合要求；
* 构建加载Dropout层的模型
* 构建除Dropout层外完全相同的模型
* 训练两个模型并对比效果
* 查看两个模型的预测结果。

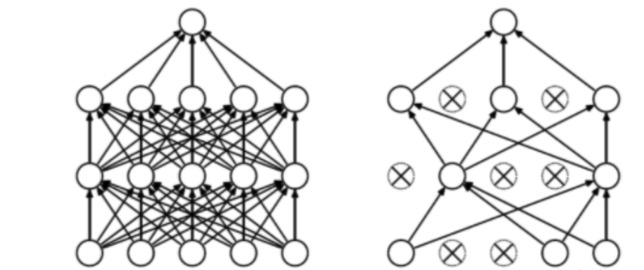
### Dropout知识点的介绍

背景知识：

2012年，Hinton在论文《[Improving neural networks by preventing co-adaptation of feature detectors](https://arxiv.org/pdf/1207.0580.pdf)》中提出Dropout。当一个复杂的前馈神经网络被训练在小数据集时，容易造成过拟合。为了防止过拟合，可以通过阻止特征检测器的共同作用来提高神经网络的性能。同年，Alex、Hinton在论文《[ImageNet Classification with Deep Convolutional Neural Networks](https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=1166C42C86BD3A3C6C75B53E2CD2E14F?doi=10.1.1.299.205&rep=rep1&type=pdf)》]中使用了Dropout算法，用于防止过拟合。该论文赢得了2012年图像识别大赛冠军。随后，关于Dropout的论文《[Dropout:A Simple Way to Prevent Neural Networks from Overfitting](http://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf)》、《[Improving Neural Networks with Dropout](http://www.cs.toronto.edu/~nitish/msc_thesis.pdf)》、《[Dropout as data augmentation](https://arxiv.org/pdf/1506.08700.pdf)》等不断提高Dropout层的应用。

Dropout原理介绍

Dropout作为训练深度神经网络的trick供开发者选择。其原理是在每个训练批次中，通过忽略一半的特征检测器（让一半的隐层节点值为0），从而地减少过拟合现象。这种方式可以减少特征检测器（隐层节点）间的相互作用（检测器相互作用是指某些检测器依赖其他检测器才能发挥作用）。具体的说：在前向传播时，让某个神经元的激活值以一定的概率p停止工作，从而使模型泛化性更强，减少对某些局部的特征的依赖。



### 数据集介绍

MNIST数据集是机器学习领域中经典的数据集，由60,000个训练样本和10,000个测试样本组成，每张图片都是28x28像素大小的灰度图像，数字从0到9。MNIST数据集主要用于图像识别领域的研究，特别是用于机器学习算法的测试和比较。由于它的简单性和易于使用，MNIST已经成为机器学习社区中的标准数据集之一。以下是几个示例图像：



## 实验环境要求

在动手进行实践之前，需要注意以下几点：

（1）确保实验环境正确安装，包括安装MindSpore。安装过程：首先登录[MindSpore官网安装界面](https://www.mindspore.cn/install)，根据安装指南下载安装包及查询相关文档。同时，官网环境安装也可以按下表说明找到对应环境搭建文档链接，根据环境搭建手册配置对应的实验环境。

（2）推荐使用交互式的计算环境Jupyter Notebook，其交互性强，易于可视化，适合频繁修改的数据分析实验环境。

（3）实验也可以在华为云一站式的AI开发平台ModelArts上完成。

（4）推荐实验环境：MindSpore版本=2.4；Python环境=3.11。

|  |  |  |
| --- | --- | --- |
| **硬件平台** | **操作系统** | **环境搭建链接** |
| CPU | Windows-x64 | [MindSpore环境搭建实验手册第二章2.1节和第三章3.1节](MindSpore环境搭建实验手册.docx) |
| GPU CUDA 10.1 | Linux-x86\_64 | [MindSpore环境搭建实验手册第二章2.2节和第三章3.1节](MindSpore环境搭建实验手册.docx) |
| Ascend 910 | Linux-x86\_64 | [MindSpore环境搭建实验手册第四章](MindSpore环境搭建实验手册.docx) |

## 实验详细设计与实现

### 数据准备

MNIST数据集是机器学习领域中经典的数据集，由60,000个训练样本和10,000个测试样本组成，每个样本是28\\*28像素的灰度手写数字图片，共10类（0-9）。整个数据集约为50M。

### 实验步骤

* + - 1. 下载数据集：

从开放数据集中下载MNIST数据集的压缩包,并解压储存在项目的根目录下。

# 从开放数据集中下载MNIST数据集

from download import download

url = "https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/" \

"notebook/datasets/MNIST\_Data.zip"

path = download(url, "./", kind="zip", replace=True)

* + - 1. 导入所需的库和函数

在使用前，导入需要的Python库。

详细的MindSpore的模块说明，可以在MindSpore API页面中搜索查询。

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

# MindSpore库

import mindspore

# 神经网络模块

from mindspore import nn

# 常见算子操作

from mindspore import ops

# 图像增强模块

from mindspore.dataset import vision

# 通用数据增强

from mindspore.dataset import transforms

# 读取和解析Manifest数据文件构建数据集

from mindspore.dataset import MnistDataset

* + - 1. 定义参数变量：

集中设置在模型训练过程中需要使用到的超参数的具体数值，方便在后续调试代码时修改超参数。

BATCH\_SIZE= 64 # batch的大小

LEARNING\_RATE = 1e-2 # 学习率

EPOCH = 3 # 迭代次数

* + - 1. 数据的读取和处理

利用MindSpore中的dataset相关的函数读取数据集中的数据。同时，MindSpore的dataset使用数据处理流水线（Data Processing Pipeline），需指定map、batch、shuffle等操作。这里我们使用map对图像数据及标签进行变换处理，然后将处理好的数据集打包为大小为64的batch。

from mindspore.dataset import transforms

# 读取和解析Manifest数据文件构建数据集

from mindspore.dataset import MnistDataset

train\_dataset = MnistDataset('MNIST\_Data/train')

test\_dataset = MnistDataset('MNIST\_Data/test')

def datapipe(dataset, batch\_size):

image\_transforms = [

# 基于给定的缩放和平移因子调整图像的像素大小。输出图像的像素大小为：output = image \* rescale + shift。

# 此处rescale取1.0 / 255.0，shift取0

vision.Rescale(1.0 / 255.0, 0),

# 正则化 均值为0.1307，标准差为0.3081（查自官网）

vision.Normalize(mean=(0.1307,), std=(0.3081,)),

# 将输入图像的shape从 <H, W, C> 转换为 <C, H, W>

vision.HWC2CHW()

]

# 将输入的Tensor转换为指定的数据类型。

label\_transform = transforms.TypeCast(mindspore.int32)

# map给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。

dataset = dataset.map(image\_transforms, 'image')

dataset = dataset.map(label\_transform, 'label')

# 将数据集中连续 batch\_size 条数据组合为一个批数据

dataset = dataset.batch(batch\_size)

return dataset

# 对数据集进行transfrom和batch

train\_dataset = datapipe(train\_dataset, BATCH\_SIZE)

test\_dataset = datapipe(test\_dataset, BATCH\_SIZE)

* + - 1. 模型构建

使用MindSpore中提供的相关函数构建训练使用的神经网络模型，包括平图层，致密连接层等，并在construct中完成神经网络的前向构造。为了对比dropout层对深度学习的影响，做如下设置：

1. 构建两个除了dropout层完全相同的网络模型。

2. 采用完全相同的超参数及相关设置。

data\_number=1600# 定义模型

# MindSpore 中提供用户通过继承 nn.Cell 来方便用户创建和执行自己的网络

class Network(nn.Cell):

# 自定义的网络中，需要在\_\_init\_\_构造函数中申明各个层的定义

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

# 继承父类nn.cell的\_\_init\_\_方法

super().\_\_init\_\_()

# nn.Flatten为输入展成平图层，即去掉那些空的维度

self.flatten = nn.Flatten()

# 使用SequentialCell对网络进行管理

self.dense\_relu\_sequential = nn.SequentialCell(

# nn.Dense为致密连接层，它的第一个参数为输入层的维度，第二个参数为输出的维度，

# 第三个参数为神经网络可训练参数W权重矩阵的初始化方式，默认为normal

# nn.ReLU()非线性激活函数，它往往比论文中的sigmoid激活函数具有更好的效益

nn.Dense(28 \* 28, 512), # 致密连接层 输入28\*28 输出512

nn.ReLU(), # ReLU层

nn.Dense(512, 512), # 致密连接层 输入512 输出512

#请补全相关的代码,# Dropout层，舍弃率为0.7

nn.ReLU(), # ReLu层

nn.Dense(512, 10) # 致密连接层 输入512 输出10

)

# 在construct中实现层之间的连接关系，完成神经网络的前向构造

def construct(self, x):

#调用init中定义的self.flatten()方法

x = self.flatten(x)

#调用init中的self.dense\_relu\_sequential()方法

logits = self.dense\_relu\_sequential(x)

# 返回模型

return logits

model = Network()

print(model)

# 定义不附加dropout层的模型

class Network2(nn.Cell):

# 自定义的网络中，需要在\_\_init\_\_构造函数中申明各个层的定义

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

# 继承父类nn.cell的\_\_init\_\_方法

super().\_\_init\_\_()

# nn.Flatten为输入展成平图层，即去掉那些空的维度

self.flatten = nn.Flatten()

# 使用SequentialCell对网络进行管理

self.dense\_relu\_sequential = nn.SequentialCell(

# nn.Dense为致密连接层，它的第一个参数为输入层的维度，第二个参数为输出的维度，

# 第三个参数为神经网络可训练参数W权重矩阵的初始化方式，默认为normal

# nn.ReLU()非线性激活函数，它往往比论文中的sigmoid激活函数具有更好的效益

nn.Dense(28 \* 28, 512), # 致密连接层 输入28\*28 输出512

# nn.Dropout(keep\_prob=0.3),# Dropout层，舍弃率为0.3

nn.ReLU(), # ReLU层

nn.Dense(512, 512), # 致密连接层 输入512 输出512

nn.ReLU(), # ReLu层

nn.Dense(512, 10) # 致密连接层 输入512 输出10

)

# 在construct中实现层之间的连接关系，完成神经网络的前向构造

def construct(self, x):

#调用init中定义的self.flatten()方法

x = self.flatten(x)

#调用init中的self.dense\_relu\_sequential()方法

logits = self.dense\_relu\_sequential(x)

# 返回模型

return logits

model2 = Network2()

print(model2)

* + - 1. 模型训练
* 在模型训练中，一个完整的训练过程（step）需要实现以下三步：

1. 正向计算：模型预测结果（logits），并与正确标签（label）求预测损失（loss）。

2. 反向传播：利用自动微分机制，自动求模型参数（parameters）对于loss的梯度（gradients）。

3. 参数优化：将梯度更新到参数上。

* MindSpore使用函数式自动微分机制，因此针对上述步骤需要实现：

1. 正向计算函数定义。

2. 通过函数变换获得梯度计算函数。

3. 训练函数定义，执行正向计算、反向传播和参数优化。

* 为对比增加dropout层和普通定义模型，进行如下设置：

1. 构建完全相同的梯度下降方法分别训练两个模型。

2. 采用完全相同的数据集以及相同的分割比例。

3. 采用相同的交叉熵损失函数。

# 实例化损失函数和优化器

# 计算预测值和目标值之间的交叉熵损失

loss\_fn = nn.CrossEntropyLoss()

#构建一个Optimizer对象，能够保持当前参数状态并基于计算得到的梯度进行参数更新 此处使用随机梯度下降算法

optimizer = nn.SGD(model.trainable\_params(), learning\_rate=LEARNING\_RATE)

optimizer2 = nn.SGD(model2.trainable\_params(), learning\_rate=LEARNING\_RATE)

def train(model, dataset, loss\_fn, optimizer):

# 定义 forward 函数

def forward\_fn(data, label):

# 将数据载入模型

logits = model(data)

# 根据模型训练获取损失函数值

loss = loss\_fn(logits, label)

return loss, logits

# 调用梯度函数，value\_and\_grad()为生成求导函数，用于计算给定函数的正向计算结果和梯度

grad\_fn = ops.value\_and\_grad(forward\_fn, None, optimizer.parameters, has\_aux=True)

# 定义一步训练的函数

def train\_step(data, label):

# 计算梯度，记录变量是怎么来的

(loss, \_), grads = grad\_fn(data, label)

# 获得损失 depend用来处理操作间的依赖关系

optimizer(grads)

return loss

size = dataset.get\_dataset\_size()

model.set\_train()

for batch, (data, label) in enumerate(dataset.create\_tuple\_iterator()):

# 批量训练获得损失值

loss = train\_step(data, label)

# 当完成所有数据样本的训练

if batch % 100 == 0:

loss, current = loss.asnumpy(), batch

print(f"loss: {loss:>7f} [{current:>3d}/{size:>3d}]")

def train(model2, dataset, loss\_fn, optimizer2):

# 定义 forward 函数

def forward\_fn(data, label):

# 将数据载入模型

logits = model2(data)

# 根据模型训练获取损失函数值

loss = loss\_fn(logits, label)

return loss, logits

# 调用梯度函数，value\_and\_grad()为生成求导函数，用于计算给定函数的正向计算结果和梯度

grad\_fn = ops.value\_and\_grad(forward\_fn, None, optimizer2.parameters, has\_aux=True)

# 定义一步训练的函数

def train\_step(data, label):

# 计算梯度，记录变量是怎么来的

(loss, \_), grads = grad\_fn(data, label)

# 获得损失 depend用来处理操作间的依赖关系

optimizer2(grads)

return loss

size = dataset.get\_dataset\_size()

model2.set\_train()

for batch, (data, label) in enumerate(dataset.create\_tuple\_iterator()):

# 批量训练获得损失值

loss = train\_step(data, label)

# 当完成所有数据样本的训练

if batch % 100 == 0:

loss, current = loss.asnumpy(), batch

print(f"loss: {loss:>7f} [{current:>3d}/{size:>3d}]")

* 测试函数及预测指标：

定义完全相同的两个测试函数，通过调用MindSpore中的相关函数，使用accuracy以及平均损失等指标评估预测模型的性能。

def test(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(): # 遍历所有测试样本数据

pred = model(data) # 根据已训练模型获取预测值

total += len(data) # 统计样本数

test\_loss += loss\_fn(pred, label).asnumpy() # 统计样本损失值

correct += (pred.argmax(1) == label).asnumpy().sum()# 统计预测正确的样本个数

test\_loss /= num\_batches # 求得平均损失

correct /= total # 计算accuracy

print(f"Test: \n Accuracy: {(100\*correct):>0.1f}%, Avg loss: {test\_loss:>8f} \n")

def test(model2, dataset, loss\_fn):

num\_batches = dataset.get\_dataset\_size()

model2.set\_train(False)

total, test\_loss, correct = 0, 0, 0

for data, label in dataset.create\_tuple\_iterator(): # 遍历所有测试样本数据

pred = model2(data) # 根据已训练模型获取预测值

total += len(data) # 统计样本数

test\_loss += loss\_fn(pred, label).asnumpy() # 统计样本损失值

correct += (pred.argmax(1) == label).asnumpy().sum()# 统计预测正确的样本个数

test\_loss /= num\_batches # 求得平均损失

correct /= total # 计算accuracy

print(f"Test: \n Accuracy: {(100\*correct):>0.1f}%, Avg loss: {test\_loss:>8f} \n")

* 执行训练：

训练过程需多次迭代数据集，一次完整的迭代称为一轮（epoch）。在每一轮，遍历训练集进行训练，结束后使用测试集进行预测。打印每一轮的loss值和预测准确率（Accuracy）。

对于两个模型，采用完全相同的设置进行训练，并查看训练结果，对比准确率 和损失函数。

for t in range(EPOCH):

print(f"Epoch {t+1}\n-------------------------------")

train(model, train\_dataset, loss\_fn, optimizer) # 训练模型

test(model, test\_dataset, loss\_fn) # 测试模型

print("finished!")

for i in range(EPOCH):

print(f"Epoch {i+1}\n-------------------------------")

train(model2, train\_dataset, loss\_fn, optimizer2) # 训练模型

test(model2, test\_dataset, loss\_fn) # 测试模型

print("finished!")

* + - 1. 模型预测

模模型训练完成后，将训练好的模型保存至指定的路径，方便后续的实验中在使用该模型的时候能够加载模型。

实例化一个随机初始化的模型，并将训练好的超参数加载到刚才初始化的模型中。

使用测试数据测试模型的识别能力。

# 保存checkpoint时的配置策略

mindspore.save\_checkpoint(model, "model.ckpt")

# 实例化随机初始化的模型

model = Network()

model2 = Network2()

# 加载检查点，加载参数到模型

param\_dict = mindspore.load\_checkpoint("model.ckpt")

param\_dict2 = mindspore.load\_checkpoint("model2.ckpt")

param\_not\_load = mindspore.load\_param\_into\_net(model, param\_dict)

param\_not\_load2 = mindspore.load\_param\_into\_net(model2, param\_dict2)

print(param\_not\_load)

调用加载dropout层的模型进行预测并显示预测结果和真实结果的对比：

model.set\_train(False)

for data, label in test\_dataset:

pred = model(data)

predicted = pred.argmax(1)

print(f'Predicted: "{predicted[:10]}", Actual: "{label[:10]}"')

break

调用不加载dropout层的模型进行预测并显示预测结果和真实结果的对比：

model2.set\_train(False)

for data, label in test\_dataset:

pred = model2(data)

predicted2 = pred.argmax(1)

print(f'Predicted: "{predicted2[:10]}", Actual: "{label[:10]}"')

break

## 实验总结

MNIST数据集是机器学习领域中经典的数据集，通过MNIST数据集来训练图片识别的模型，在整个机器学习训练的过程中构建除dropout完全相同的两个模型，通过两个模型的训练和预测的结果，观察Dropout层对深度学习模型训练以及预测的影响。

## 实验任务与参考解答任务

### 实验任务

在实验过程中补全Dropout层相关的代码。

### 参考答案

self.dense\_relu\_sequential = nn.SequentialCell(

# nn.Dense为致密连接层，它的第一个参数为输入层的维度，第二个参数为输出的维度，

# 第三个参数为神经网络可训练参数W权重矩阵的初始化方式，默认为normal

# nn.ReLU()非线性激活函数，它往往比论文中的sigmoid激活函数具有更好的效益

nn.Dense(28 \* 28, 512), # 致密连接层 输入28\*28 输出512

nn.ReLU(), # ReLU层

nn.Dense(512, 512), # 致密连接层 输入512 输出512

nn.Dropout(p=0.7),# Dropout层，舍弃率为0.7

nn.ReLU(), # ReLu层

nn.Dense(512, 10) # 致密连接层 输入512 输出10

)