1. **FashionMnist分类任务正则化对比实验**

## 实验介绍

本实验使用Fashion-MNIST数据集，它是一个替代MNIST手写数字集的图像数据集，由Zalando（一家德国的时尚科技公司）旗下的研究部门提供。其涵盖了来自10种类别的共7万个不同商品的正面图片。Fashion-MNIST的大小、格式和训练集/测试集划分与原始的MNIST完全一致。60000/10000的训练测试数据划分，28x28的灰度图片。通过上述实验我们对比不同正则化技术效果，此实验我们应用正则化技术做案例分析，并对比有无正则化的训练结果。

* 1. 实验环境要求
* ModelArts平台：Mindspore-1.3，也可在本地搭建python3.7.5和MindSpore1.3环境完成实验

若选择在ModelArts平台快速搭建(创建实验环境的过程见实验一：鸢尾花分类任务对比实验)

* 1. 实验总体设计
  2. 实验过程

本节将详细介绍实验的设计与实现。

数据准备；

导入实验环境；

数据读取和预处理；

训练模型；

观察总结。

* + 1. 数据准备

下载和解压数据集

本实验需要用到的是Fashion-MNIST数据集，由于华为云下载该数据集会花费较多时间，因此可以选择从OBS上下载并解压到notebook的环境中。只下载一次就可。

#从OBS桶下载数据集

!wget https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/fashion-mnist.zip

#解压文件

!unzip fashion-mnist.zip

Fashion Mnist有10个标签，如下表所示：

|  |  |
| --- | --- |
| 标签 | 描述 |
| 0 | T-shirt/top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boot |

* + 1. 导入实验环境

导入库

import os

import struct

import sys

from easydict import EasyDict as edict

import matplotlib.pyplot as plt

import numpy as np

import mindspore

import mindspore.dataset as ds

import mindspore.nn as nn

from mindspore import context

from mindspore.nn.metrics import Accuracy

from mindspore.train import Model

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor

from mindspore import Tensor

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

定义常量

cfg = edict({

'train\_size': 60000, # 训练集大小

'test\_size': 10000, # 测试集大小

'channel': 1, # 图片通道数

'image\_height': 28, # 图片高度

'image\_width': 28, # 图片宽度

'batch\_size': 64,

'num\_classes': 10, # 分类类别

'lr': 0.001, # 学习率

'epoch\_size': 20, # 训练次数

'data\_dir\_train': os.path.join('fashion-mnist', 'train'),

'data\_dir\_test': os.path.join('fashion-mnist', 'test'),

})

### 数据读取和预处理

定义函数用于读取数据

def read\_image(file\_name):

'''

:param file\_name: 文件路径

:return: 训练或者测试数据

如下是训练的图片的二进制格式

[offset] [type] [value] [description]

0000 32 bit integer 0x00000803(2051) magic number

0004 32 bit integer 60000 number of images

0008 32 bit integer 28 number of rows

0012 32 bit integer 28 number of columns

0016 unsigned byte ?? pixel

0017 unsigned byte ?? pixel

........

xxxx unsigned byte ?? pixel

'''

file\_handle = open(file\_name, "rb") # 以二进制打开文档

file\_content = file\_handle.read() # 读取到缓冲区中

head = struct.unpack\_from('>IIII', file\_content, 0) # 取前4个整数，返回一个元组

offset = struct.calcsize('>IIII')

imgNum = head[1] # 图片数

width = head[2] # 宽度

height = head[3] # 高度

bits = imgNum \* width \* height # data一共有60000\*28\*28个像素值

bitsString = '>' + str(bits) + 'B' # fmt格式：'>47040000B'

imgs = struct.unpack\_from(bitsString, file\_content, offset) # 取data数据，返回一个元组

imgs\_array = np.array(imgs, np.float32).reshape((imgNum, width \* height)) # 最后将读取的数据reshape成 【图片数，图片像素】二维数组

return imgs\_array

def read\_label(file\_name):

'''

:param file\_name:

:return:

标签的格式如下：

[offset] [type] [value] [description]

0000 32 bit integer 0x00000801(2049) magic number (MSB first)

0004 32 bit integer 60000 number of items

0008 unsigned byte ?? label

0009 unsigned byte ?? label

........

xxxx unsigned byte ?? label

The labels values are 0 to 9.

'''

file\_handle = open(file\_name, "rb") # 以二进制打开文档

file\_content = file\_handle.read() # 读取到缓冲区中

head = struct.unpack\_from('>II', file\_content, 0) # 取前2个整数，返回一个元组

offset = struct.calcsize('>II')

labelNum = head[1] # label数

bitsString = '>' + str(labelNum) + 'B' # fmt格式：'>47040000B'

label = struct.unpack\_from(bitsString, file\_content, offset) # 取data数据，返回一个元组

return np.array(label, np.int32)

def get\_data():

# 文件获取

train\_image = os.path.join(cfg.data\_dir\_train, 'train-images-idx3-ubyte')

test\_image = os.path.join(cfg.data\_dir\_test, "t10k-images-idx3-ubyte")

train\_label = os.path.join(cfg.data\_dir\_train, "train-labels-idx1-ubyte")

test\_label = os.path.join(cfg.data\_dir\_test, "t10k-labels-idx1-ubyte")

# 读取数据

train\_x = read\_image(train\_image)

test\_x = read\_image(test\_image)

train\_y = read\_label(train\_label)

test\_y = read\_label(test\_label)

return train\_x, train\_y, test\_x, test\_y

数据预处理

图像最后一个维度，即通道（颜色），使用reshape()函数将其添加到train\_images和test\_images的维度中。在这种情况下，它是单一颜色，因此通道为1，即“灰度”。为了减少计算量，还需要把图片的像素值进行归一化，八位图像的像素值范围在0-255之间，将所有像素值除以255，使得像素值范围控制在0-1之间。并打印数据集形状和一张图片作为例子。

train\_x, train\_y, test\_x, test\_y = get\_data()

train\_x = train\_x.reshape(-1, 1, cfg.image\_height, cfg.image\_width)

test\_x = test\_x.reshape(-1, 1, cfg.image\_height, cfg.image\_width)

train\_x = train\_x / 255.0

test\_x = test\_x / 255.0

print('训练数据集样本数：', train\_x.shape[0])

print('测试数据集样本数：', test\_y.shape[0])

print('通道数/图像长/宽：', train\_x.shape[1:])

print('一张图像的标签样式：', train\_y[0]) # 一共10类，用0-9的数字表达类别。

plt.figure()

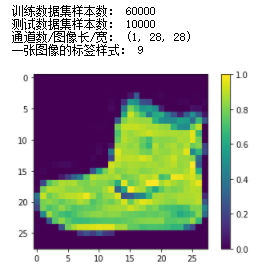
plt.imshow(train\_x[0,0,...])

plt.colorbar()

plt.grid(False)

plt.show()

输出结果：



数据集预处理

在训练之前，需要先对数据集中的数据进行“洗牌”，打乱数据集的顺序。

# 转换数据类型为Dataset

def create\_dataset():

XY\_train = list(zip(train\_x, train\_y))

ds\_train = ds.GeneratorDataset(XY\_train, ['x', 'y'])

ds\_train = ds\_train.shuffle(buffer\_size=1000).batch(cfg.batch\_size, drop\_remainder=True)

XY\_test = list(zip(test\_x, test\_y))

ds\_test = ds.GeneratorDataset(XY\_test, ['x', 'y'])

ds\_test = ds\_test.shuffle(buffer\_size=1000).batch(cfg.batch\_size, drop\_remainder=True)

return ds\_train, ds\_test

### 训练模型

创建卷积神经网络

该实验中，可以选择使用不含正则化的卷积神经网络或者选择加入正则化的卷积神经网络，用于对比结果。

下面这部分用于创建不加入正则化的卷积神经网络，网络结构为：卷积层1🡪卷积层2🡪卷积层3🡪最大池化层🡪全连接层1🡪全连接层2。

# 定义卷积神经网络，无正则化

class ForwardFashion(nn.Cell):

def \_\_init\_\_(self, num\_class=10): # 一共分十类，图片通道数是1

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

self.num\_class = num\_class

self.conv1 = nn.Conv2d(1, 32,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.conv2 = nn.Conv2d(32, 64,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.conv3 = nn.Conv2d(64, 128,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.maxpool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

self.relu = nn.ReLU()

self.flatten = nn.Flatten()

self.fc1 = nn.Dense(128 \* 11 \* 11, 128)

self.fc2 = nn.Dense(128, self.num\_class)

def construct(self, x):

x = self.conv1(x)

x = self.relu(x)

x = self.conv2(x)

x = self.relu(x)

x = self.conv3(x)

x = self.relu(x)

x = self.maxpool2d(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.relu(x)

x = self.fc2(x)

return x

下面这部分用于创建加入正则化的卷积神经网络，网络结构为卷积层1🡪dropout层1🡪卷积层2🡪dropout层1🡪卷积层3🡪dropout层1🡪最大池化层🡪 dropout层2🡪全连接层1🡪 dropout层2🡪全连接层2：

# 定义卷积神经网络，有正则化

class ForwardFashionRegularization(nn.Cell):

def \_\_init\_\_(self, num\_class=10): # 一共分十类，图片通道数是1

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

self.num\_class = num\_class

self.conv1 = nn.Conv2d(1, 32,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.conv2 = nn.Conv2d(32, 64,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.conv3 = nn.Conv2d(64, 128,kernel\_size=3, stride=1, padding=0, has\_bias=False, pad\_mode="valid")

self.maxpool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

self.relu = nn.ReLU()

self.dropout = nn.Dropout()

self.flatten = nn.Flatten()

self.fc1 = nn.Dense(3200, 128)

self.bn = nn.BatchNorm1d(128)

self.fc2 = nn.Dense(128, self.num\_class)

def construct(self, x):

x = self.conv1(x)

x = self.relu(x)

x = self.conv2(x)

x = self.relu(x)

x = self.maxpool2d(x)

x = self.dropout(x)

x = self.conv3(x)

x = self.relu(x)

x = self.maxpool2d(x)

x = self.dropout(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.relu(x)

x = self.bn(x)

x = self.dropout(x)

x = self.fc2(x)

return x

启动训练

为这个模型指定优化器（adam）、损失函数（crossentropy）和度量(accuracy)，然后启动训练，最后进行验证。

def train(Net):

ds\_train, ds\_test = create\_dataset()

# 构建网络

network = Net(cfg.num\_classes)

# 定义模型的损失函数，优化器

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

net\_opt = nn.Adam(network.trainable\_params(), cfg.lr)

# 训练模型

model = Model(network, loss\_fn=net\_loss, optimizer=net\_opt, metrics={'acc': Accuracy()})

loss\_cb = LossMonitor()

print("============== Starting Training ==============")

model.train(30, ds\_train, callbacks=[loss\_cb], dataset\_sink\_mode=True)

# 验证

metric = model.eval(ds\_test)

print(metric)

return model

训练并验证无正则化的网络。

# 训练无正则化的网络

model = train(ForwardFashion)

输出结果：

============== Starting Training ==============

epoch: 1 step: 937, loss is 0.36209568

epoch: 2 step: 937, loss is 0.11113132

epoch: 3 step: 937, loss is 0.107788906

epoch: 4 step: 937, loss is 0.12908919

epoch: 5 step: 937, loss is 0.078461185

epoch: 6 step: 937, loss is 0.18977618

epoch: 7 step: 937, loss is 0.15168177

epoch: 8 step: 937, loss is 0.06739945

epoch: 9 step: 937, loss is 0.14379226

epoch: 10 step: 937, loss is 0.076876596

epoch: 11 step: 937, loss is 0.055951692

epoch: 12 step: 937, loss is 0.022819532

epoch: 13 step: 937, loss is 0.10054826

epoch: 14 step: 937, loss is 0.012528818

epoch: 15 step: 937, loss is 0.0076259132

epoch: 16 step: 937, loss is 0.07877082

epoch: 17 step: 937, loss is 0.031406786

epoch: 18 step: 937, loss is 0.009203883

epoch: 19 step: 937, loss is 0.005287296

epoch: 20 step: 937, loss is 0.0929834

epoch: 21 step: 937, loss is 0.0015465739

epoch: 22 step: 937, loss is 0.03491202

epoch: 23 step: 937, loss is 0.0005662525

epoch: 24 step: 937, loss is 0.010102608

epoch: 25 step: 937, loss is 0.003999765

epoch: 26 step: 937, loss is 0.011775437

epoch: 27 step: 937, loss is 0.080310896

epoch: 28 step: 937, loss is 0.0018242185

epoch: 29 step: 937, loss is 0.007360851

epoch: 30 step: 937, loss is 0.003999797

{'acc': 0.9147636217948718}

训练并验证有正则化的网络。

# 训练有正则化的网络

model = train(ForwardFashionRegularization)

输出结果：

============== Starting Training ==============

epoch: 1 step: 937, loss is 0.29856867

epoch: 2 step: 937, loss is 0.28910726

epoch: 3 step: 937, loss is 0.18035105

epoch: 4 step: 937, loss is 0.2785972

epoch: 5 step: 937, loss is 0.21400028

epoch: 6 step: 937, loss is 0.27920294

epoch: 7 step: 937, loss is 0.17452516

epoch: 8 step: 937, loss is 0.309029

epoch: 9 step: 937, loss is 0.30411178

epoch: 10 step: 937, loss is 0.2842149

epoch: 11 step: 937, loss is 0.22666603

epoch: 12 step: 937, loss is 0.16507925

epoch: 13 step: 937, loss is 0.17004505

epoch: 14 step: 937, loss is 0.23396353

epoch: 15 step: 937, loss is 0.20207018

epoch: 16 step: 937, loss is 0.43118417

epoch: 17 step: 937, loss is 0.23762615

epoch: 18 step: 937, loss is 0.24660718

epoch: 19 step: 937, loss is 0.12197974

epoch: 20 step: 937, loss is 0.22719634

epoch: 21 step: 937, loss is 0.2809552

epoch: 22 step: 937, loss is 0.21124852

epoch: 23 step: 937, loss is 0.2100177

epoch: 24 step: 937, loss is 0.29766798

epoch: 25 step: 937, loss is 0.115716025

epoch: 26 step: 937, loss is 0.41360933

epoch: 27 step: 937, loss is 0.11700327

epoch: 28 step: 937, loss is 0.2552187

epoch: 29 step: 937, loss is 0.14747506

epoch: 30 step: 937, loss is 0.19088028

{'acc': 0.9234775641025641}

预测模型

使用上述训练好的模型对测试数据集进行预测。打印预测结果

# 预测

ds\_test, \_ = create\_dataset()

test\_ = ds\_test.create\_dict\_iterator(output\_numpy=True).\_\_next\_\_()

predictions = model.predict(Tensor(test\_['x']))

predictions = predictions.asnumpy()

for i in range(15):

p\_np = predictions[i, :]

p\_list = p\_np.tolist()

print('第' + str(i) + '个sample预测结果：', p\_list.index(max(p\_list)), ' 真实结果：', test\_['y'][i])

输出结果：

第0个sample预测结果： 3 真实结果： 3

第1个sample预测结果： 5 真实结果： 5

第2个sample预测结果： 6 真实结果： 2

第3个sample预测结果： 4 真实结果： 4

第4个sample预测结果： 3 真实结果： 3

第5个sample预测结果： 3 真实结果： 3

第6个sample预测结果： 7 真实结果： 7

第7个sample预测结果： 8 真实结果： 8

第8个sample预测结果： 3 真实结果： 3

第9个sample预测结果： 5 真实结果： 5

第10个sample预测结果： 9 真实结果： 9

第11个sample预测结果： 2 真实结果： 4

第12个sample预测结果： 6 真实结果： 6

第13个sample预测结果： 5 真实结果： 5

第14个sample预测结果： 0 真实结果： 0

可视化结果

定义可视化函数。

# -------------------定义可视化函数--------------------------------

# 输入预测结果序列，真实标签序列，以及图片序列

# 目标是根据预测值对错，让其标签显示为红色或者蓝色。对：标签为红色；错：标签为蓝色

def plot\_image(predictions\_array, true\_label, img):

plt.grid(False)

plt.xticks([])

plt.yticks([])

# 显示对应图片

plt.imshow(img, cmap=plt.cm.binary)

# 显示预测结果的颜色，如果对上了是蓝色，否则为红色

predicted\_label = np.argmax(predictions\_array)

if predicted\_label == true\_label:

color = 'blue'

else:

color = 'red'

# 显示对应标签的格式，样式

plt.xlabel('{},{:2.0f}% ({})'.format(class\_names[predicted\_label],

100 \* np.max(predictions\_array),

class\_names[true\_label]), color=color)

# 将预测的结果以柱状图形状显示蓝对红错

def plot\_value\_array(predictions\_array, true\_label):

plt.grid(False)

plt.xticks([])

plt.yticks([])

this\_plot = plt.bar(range(10), predictions\_array, color='#777777')

plt.ylim([0, 1])

predicted\_label = np.argmax(predictions\_array)

this\_plot[predicted\_label].set\_color('red')

this\_plot[true\_label].set\_color('blue')

import numpy as np

def softmax\_np(x):

x = x - np.max(x)

exp\_x = np.exp(x)

softmax\_x = exp\_x/np.sum(exp\_x)

return softmax\_x

预测结果可视化，输入预测结果序列，真实标签序列，以及图片序列。目标是根据预测值对错，让其标签显示为红色或者蓝色。对：标签为蓝色；错：标签为红色。最后预测15个图像与标签，将预测的结果以柱状图形状显示蓝对红错。

# 预测15个图像与标签，并展现出来

num\_rows = 5

num\_cols = 3

num\_images = num\_rows \* num\_cols

plt.figure(figsize=(2 \* 2 \* num\_cols, 2 \* num\_rows))

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

for i in range(num\_images):

plt.subplot(num\_rows, 2 \* num\_cols, 2 \* i + 1)

pred\_np\_ = predictions[i, :]

pred\_np\_ = softmax\_np(pred\_np\_)

plot\_image(pred\_np\_, test\_['y'][i], test\_['x'][i, 0, ...])

plt.subplot(num\_rows, 2 \* num\_cols, 2 \* i + 2)

plot\_value\_array(pred\_np\_, test\_['y'][i])

plt.show()

输出结果：



### 观察总结

如果神经网络具有较少的层数和神经元数量，则它可能在训练数据集上表现不佳，即可能导致欠拟合问题。但是，如果神经网络具有很多的层数和神经元数量，它可能在训练数据集上表现得很好（高精度），但是可能在测试数据集上表现不好，即可能导致过拟合问题。

不幸的是，没有确定公式来确定模型的正确结构（层数和每层中的神经元数量），搭建网络时将需要不断尝试，来找到表现不错的架构。

虽然对于神经网络，我们还有其他正则化技术，如L1, L2正则化，Dropout是所有正则化技术中最有效的，也是最常用的。Dropout会随机地丢弃层的一些输出特征（神经元），即设置为零。假设在应用Dropout之前层的输出为[0.5, 0.8, 2.2, 0.9, 0.1]，在应用Dropout之后，输出将为[0, 0.8, 2.2, 0, 0.1]。“Dropout率”是层中被置为零的特征（神经元）的分数。一般来说，Dropout率保持在0.2至0.5之间。

请注意，Dropout只在训练阶段使用。在对测试数据集进行预测时，我们不需要从模型中手动移除Dropout层。 在测试阶段，不会应用Dropout。

因此，为了减少本案例中的过拟合问题，让我们应用正则化技术-Dropout来提升测试表现。

* 1. 实验总结

本章提供了一个FashionMnist正则化前后对比实验。该实验选取FashionMnist灰度数据集将模型进行训练预测，当初始模型表现过拟合时，参数量多的模型的性能反而不如参数量少的模型，我们加入正则化技术重新建立新模型并预测。

* 1. 创新设计

基于本实验描述，尝试L1, L2正则化前后对比实验。