**《深度学习与大数据智能》实验报告二**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **20计算机科学与技术（卓越）01** | | **姓名** | | **黄昊** | **学号** | **20204205** |
| **实验题目** | **FashionMnist分类任务正则化对比实验** | | | | | | | |
| **实验时间** | **2023/05/07** | | **实验地点** | | **DS3302** | | | |
| **学年学期** | **2021-2021（2）** | | **实验性质** | | **□验证性 □设计性 ■综合性** | | | |
| 一、实验目的  1. 进一步理解卷积神经网络的基本构成。  2. 理解并掌握正则化技术在卷积神经网络中的作用。 | | | | | | | | |
| 二、实验项目内容  1. 基本内容：构建基于Fashion-MNIST数据集的分类卷积神经网络模型，该模型中需使用相应的正则化技术。  2. 进阶内容：  1）基于基本实验内容，进行L1/L2/Dropout/BN正则化前后对比实验。  2） 阅读相关文献，使用上述1）中未提到的一种正则化技术进行实验，并分析实验结果。  3）分析实验中使用的正则化技术的特点及其使用范围。 | | | | | | | | |
| 三、实验的过程和主要源程序  ①原理介绍  在模型训练过程中，常常会碰到过拟合问题。过拟合问题表现为，在训练集上的测试性能良好，但在训练集上的表现非常差，即反映了模型的泛化性能较差。而为了解决过拟合问题，我们在模型训练过程中常常会使用各种各样的正则化技术，以缓解模型的过拟合问题。下面介绍一些常见的正则化技术：  A.L1正则化与L2正则化  L1正则化和L2正则化的思想类似，都是在原先的损失函数上面加上一个正则化项目，对于L1，有：  L2，有：  其中*L0*是原来的损失函数，*w*是模型权重。这两个正则化技术的思想类似，认为权重过大有向训练集过拟合的趋势，于是希望模型权重不那么大，以缓解过拟合。其中*λ*是超参数，需要根据模型不断调节/  B.BatchNorm正则化  首先，对于一个模型的输入，我们常常会作正则化处理：  其中*μ*为均值，*σ*为输入的标准差。因此，我们也想在模型中加入一个BatchNorm层，这一层的作用是对上一层的输出作正则化处理。因此，本层需要学习两个参数：*μ*和*σ。*  C.Dropout正则化  Dropout的作用是在每个训练批次中，通过忽略一定比例的节点（让这些节点输出恒定为0），使得减少过拟合现象，减少各个节点之间的相互作用，以提高泛化能力。  D.Early Stop  根据相关文献指出，在模型训练过程中，刚开始训练的时候，模型通常处于欠拟合状态。而到达一定时刻后，模型开始有过拟合的倾向。而Early Stopping的思想是关注测试集上的一定指标，当这个指标开始变差时，就停止训练，而不是训练到指定的训练次数。  ②模型搭建  结合本次实验内容，本次实验搭建了四种模型：不带任何正则化层的模型，带Dropout层的模型，带BatchNorm层的模型和带BatchNorm和Dropout层的模型，如下从左到右依次如下所示：    ③主要源程序  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, Loss  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='CPU')  cfg = edict({      'train\_size': 60000,  *# 训练集大小*      'test\_size': 10000,  *# 测试集大小*  *# 'train\_size': 600,  # 训练集大小*  *# 'test\_size': 100,  # 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': 256,      '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'),  })  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  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()  *# 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  *# 定义一个自定义的回调函数，继承自Callback类*  from mindspore.train.callback import Callback  class EvalCallBack(Callback):      def \_\_init\_\_(self, model, train\_dataset, test\_dataset, dataset\_sink\_mode = False, epochs\_to\_eval = 1):          self.model = model          self.train\_dataset = train\_dataset          self.test\_dataset = test\_dataset  *# epochs\_to\_eval是一个int数字，代表着：每隔多少个epoch进行一次验证*          self.epochs\_to\_eval = epochs\_to\_eval          self.per\_eval = {"epoch": [], "test\_acc": [], "train\_acc": [], "test\_loss": [], "train\_loss": []}          self.dataset\_sink\_mode = dataset\_sink\_mode      def epoch\_end(self, run\_context):  *# 获取到现在的epoch数*          cb\_param = run\_context.original\_args()          cur\_epoch = cb\_param.cur\_epoch\_num  *# 如果达到进行验证的epoch数，则进行以下验证操作*          if cur\_epoch % self.epochs\_to\_eval == 0:  *# 此处model设定的metrics是准确率Accuracy*              train\_acc = self.model.eval(self.train\_dataset, dataset\_sink\_mode=self.dataset\_sink\_mode)["train\_acc"]              test\_acc = self.model.eval(self.test\_dataset, dataset\_sink\_mode=self.dataset\_sink\_mode)["test\_acc"]              train\_loss = self.model.eval(self.train\_dataset, dataset\_sink\_mode=self.dataset\_sink\_mode)["train\_loss"]              test\_loss = self.model.eval(self.test\_dataset, dataset\_sink\_mode=self.dataset\_sink\_mode)["test\_loss"]              self.per\_eval["epoch"].append(cur\_epoch)              self.per\_eval["test\_acc"].append(test\_acc)              self.per\_eval["train\_acc"].append(train\_acc)              self.per\_eval["train\_loss"].append(train\_loss)              self.per\_eval["test\_loss"].append(test\_loss)              print("Epoch{}: train\_acc: {}, test\_acc: {}, train\_loss: {}, test\_loss{}".format(cur\_epoch, train\_acc, test\_acc, train\_loss, test\_loss))  *# 定义卷积神经网络，BaseLine*  class ForwardFashion\_baseline(nn.Cell):      def \_\_init\_\_(self, num\_class=10):  *# 一共分十类，图片通道数是1*          super(ForwardFashion\_baseline, 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  *# 定义卷积神经网络，批量归一化*  class ForwardFashionWithBatchNorm(nn.Cell):      def \_\_init\_\_(self, num\_class=10):  *# 一共分十类，图片通道数是1*          super(ForwardFashionWithBatchNorm, 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(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  *# 定义卷积神经网络，有Dropout正则化*  class ForwardFashionWithDropout(nn.Cell):      def \_\_init\_\_(self, num\_class=10):  *# 一共分十类，图片通道数是1*          super(ForwardFashionWithDropout, 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(0.8)          self.flatten = nn.Flatten()          self.fc1 = nn.Dense(3200, 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.dropout(x)          x = self.fc2(x)          return x  *# 定义卷积神经网络，批量归一化*  class ForwardFashionWithBatchNormAndDropout(nn.Cell):      def \_\_init\_\_(self, num\_class=10):  *# 一共分十类，图片通道数是1*          super(ForwardFashionWithBatchNormAndDropout, 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(3200, 128)          self.bn = nn.BatchNorm1d(128)          self.fc2 = nn.Dense(128, self.num\_class)          self.dropout = nn.Dropout()      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  from mindspore import ops  *# 定义具有L1正则化的交叉熵损失函数*  class L1CrossEntropyLoss(nn.Cell):      def \_\_init\_\_(self, params, l1\_weight = 1e-4, reduction = "mean"):          super(L1CrossEntropyLoss, self).\_\_init\_\_(reduction)          self.params = params          self.ce = nn.CrossEntropyLoss()          self.l1 = nn.L1Regularizer(l1\_weight)          self.reduce\_sum = ops.ReduceSum()          self.concat = ops.Concat(axis=0)          self.reshape = ops.Reshape()      def construct(self, pred, label):          flatten\_params = self.concat([self.reshape(p, (-1,)) for p in self.params])          ce\_loss = self.ce(pred, label)          l1\_loss = self.reduce\_sum(self.l1(flatten\_params))      *# 计算L1范数惩罚项*          total\_loss = ce\_loss + l1\_loss          return total\_loss  from mindspore import ops  *# 定义具有L2正则化的交叉熵损失函数*  class L2CrossEntropyLoss(nn.Cell):      def \_\_init\_\_(self, params, l2\_weight = 1e-4, reduction = "mean"):          super(L2CrossEntropyLoss, self).\_\_init\_\_(reduction)          self.params = params          self.ce = nn.CrossEntropyLoss()          self.l1 = nn.L1Regularizer(1)          self.l2\_weight = l2\_weight          self.reduce\_sum = ops.ReduceSum()          self.concat = ops.Concat(axis=0)          self.reshape = ops.Reshape()      def construct(self, pred, label):          flatten\_params = self.concat([self.reshape(p, (-1,)) for p in self.params])          ce\_loss = self.ce(pred, label)          l1\_loss = self.reduce\_sum(self.l1(flatten\_params)) \* self.reduce\_sum(self.l1(flatten\_params)) \* self.l2\_weight      *# 计算L1范数惩罚项*          total\_loss = ce\_loss + l1\_loss          return total\_loss  def train(network, loss\_fn):      ds\_train, ds\_test = create\_dataset()  *# 定义模型的损失函数，优化器*      net\_opt = nn.Adam(network.trainable\_params(), cfg.lr)  *# 训练模型*      model = Model(network, loss\_fn=loss\_fn, optimizer=net\_opt, metrics={'test\_acc': Accuracy(), 'train\_acc': Accuracy(), 'test\_loss': Loss(),  'train\_loss': Loss()})      loss\_cb = EvalCallBack(model=model, train\_dataset=ds\_train, test\_dataset=ds\_test)      print("============== Starting Training ==============")      model.train(cfg.epoch\_size, ds\_train, callbacks=[loss\_cb], dataset\_sink\_mode=True)      print(loss\_cb.per\_eval)      metricDic = loss\_cb.per\_eval      return model, metricDic  def saveMetric(path, metric):      with open(path, encoding="utf8", mode="w") as f:          f.write(str(metric))  from matplotlib import pyplot as plt  def drawAndSaveFig(path, metricList, title):      print(metricList)      epochNum = len(metricList["train\_acc"])      plt.subplot(1,2,1)      plt.plot([i for i in range(epochNum)], metricList["train\_acc"], metricList["test\_acc"])      plt.legend(["train\_acc", "test\_acc"])      plt.title(title + "\_acc")      plt.subplot(1,2,2)      plt.plot([i for i in range(epochNum)], metricList["train\_loss"], metricList["test\_loss"])      plt.legend(["train\_loss", "test\_loss"])      plt.title(title + "\_loss")      plt.savefig(path)      plt.show()  *# Baseline*  network = ForwardFashion\_baseline(cfg.num\_classes)  loss\_fn = nn.CrossEntropyLoss()  model, metricDic = train(network, loss\_fn)  saveMetric("./metric/baseline\_metric.txt", metricDic)  saveModel("./model/baseline\_model.ckpt", network)  drawAndSaveFig("./img/metric\_baseLine", metricDic, "BaseLine")  *# L1*  network\_L1 = ForwardFashion\_baseline(cfg.num\_classes)  loss\_fn\_L1 = L1CrossEntropyLoss(network\_L1.trainable\_params(), l1\_weight=1e-8)  model\_L1, metricDic\_L1 = train(network\_L1, loss\_fn\_L1)  saveMetric("./metric/baseline\_metric\_L1.txt", metricDic\_L1)  saveModel("./model/baseline\_model\_L1.ckpt", network\_L1)  drawAndSaveFig("./img/metric\_L1", metricDic\_L1, "metric\_L1")  *# L2*  network\_L2 = ForwardFashion\_baseline(cfg.num\_classes)  loss\_fn\_L2 = L2CrossEntropyLoss(network\_L2.trainable\_params(), l2\_weight=1e-15)  model\_L2, metricDic\_L2 = train(network\_L2, loss\_fn\_L2)  saveMetric("./metric/baseline\_metric\_L2.txt", metricDic\_L2)  saveModel("./model/baseline\_model\_L2.ckpt", network\_L2)  drawAndSaveFig("./img/metric\_L2", metricDic\_L2, "metric\_L2")  *# BN*  network\_BN = ForwardFashionWithBatchNorm(cfg.num\_classes)  loss\_fn = nn.CrossEntropyLoss()  model\_BN, metricDic\_BN = train(network\_BN, loss\_fn)  saveMetric("./metric/baseline\_metric\_BN.txt", metricDic\_BN)  saveModel("./model/baseline\_model\_BN.ckpt", network\_BN)  drawAndSaveFig("./img/metric\_BN", metricDic\_BN, "metric\_BN")  *# Dropout*  network\_Drop = ForwardFashionWithDropout(cfg.num\_classes)  loss\_fn = nn.CrossEntropyLoss()  model\_Drop, metricDic\_Drop = train(network\_Drop, loss\_fn)  saveMetric("./metric/baseline\_metric\_Drop.txt", metricDic\_Drop)  saveModel("./model/baseline\_model\_Drop.ckpt", network\_Drop)  drawAndSaveFig("./img/metric\_Dropout", metricDic\_Drop, "metric\_Dropout")  *# Dropout + BN*  network\_Drop\_BN = ForwardFashionWithBatchNormAndDropout(cfg.num\_classes)  loss\_fn = nn.CrossEntropyLoss()  model\_Drop\_BN, metricDic\_Drop\_BN = train(network\_Drop\_BN, loss\_fn)  saveMetric("./metric/baseline\_metric\_Drop\_BN.txt", metricDic\_Drop\_BN)  saveModel("./model/baseline\_model\_Drop\_BN.ckpt", network\_Drop\_BN)  drawAndSaveFig("./img/metric\_Dropout\_BN", metricDic\_Drop\_BN, "metric\_Dropout\_BN")  from mindspore.train.callback import Callback  class EvalCallBackWithEarlyStop(Callback):      def \_\_init\_\_(self, model, train\_dataset, test\_dataset, dataset\_sink\_mode = False, epochs\_to\_eval = 1, acc\_delta\_limit = 0.01, patience = 0):          self.model = model          self.train\_dataset = train\_dataset          self.test\_dataset = test\_dataset  *# epochs\_to\_eval是一个int数字，代表着：每隔多少个epoch进行一次验证*          self.epochs\_to\_eval = epochs\_to\_eval          self.per\_eval = {"epoch": [], "test\_acc": [], "train\_acc": [], "test\_loss": [], "train\_loss": []}          self.dataset\_sink\_mode = dataset\_sink\_mode          self.accDelta = acc\_delta\_limit          self.patience = patience          self.failEpoch = 0      def epoch\_end(self, run\_context):  *# 获取到现在的epoch数*          cb\_param = run\_context.original\_args()          cur\_epoch = cb\_param.cur\_epoch\_num  *# 如果达到进行验证的epoch数，则进行以下验证操作*          if cur\_epoch % self.epochs\_to\_eval == 0:  *# 此处model设定的metrics是准确率Accuracy*              train\_acc = self.model.eval(self.train\_dataset, dataset\_sink\_mode=self.dataset\_sink\_mode)["train\_acc"]              test\_acc = self.model.eval(self.test\_dataset, dataset\_sink\_mode=self.dataset\_sink\_mode)["test\_acc"]              train\_loss = self.model.eval(self.train\_dataset, dataset\_sink\_mode=self.dataset\_sink\_mode)["train\_loss"]              test\_loss = self.model.eval(self.test\_dataset, dataset\_sink\_mode=self.dataset\_sink\_mode)["test\_loss"]              self.per\_eval["epoch"].append(cur\_epoch)              self.per\_eval["test\_acc"].append(test\_acc)              self.per\_eval["train\_acc"].append(train\_acc)              self.per\_eval["train\_loss"].append(test\_acc)              self.per\_eval["test\_loss"].append(train\_acc)              print("Epoch{}: train\_acc: {}, test\_acc: {}, train\_loss: {}, test\_loss{}".format(cur\_epoch, train\_acc, test\_acc, train\_loss, test\_loss))              delta = test\_acc              if len(self.per\_eval["test\_acc"]) != 1:                  delta -= self.per\_eval["test\_acc"][-2]              print(delta, self.failEpoch, self. patience)              if delta < self.accDelta:                  if self.failEpoch >= self.patience:                      print("Stop.")                      run\_context.request\_stop()                  else:                      self.failEpoch += 1                      print("本次更新未达标，相关参数: ", self.failEpoch)              else:                  self.failEpoch = 0      *# 达标，清空*      *# early stop*  cfg.epoch\_size = 100  def train\_earlyStop(network, loss\_fn):      ds\_train, ds\_test = create\_dataset()      net\_opt = nn.Adam(network.trainable\_params(), cfg.lr)      model = Model(network, loss\_fn=loss\_fn, optimizer=net\_opt, metrics={'test\_acc': Accuracy(), 'train\_acc': Accuracy(), 'test\_loss': Loss(),  'train\_loss': Loss()})      loss\_cb = EvalCallBackWithEarlyStop(model=model, train\_dataset=ds\_train, test\_dataset=ds\_test, acc\_delta\_limit = 0.01, patience = 2)      print("============== Starting Training ==============")      model.train(cfg.epoch\_size, ds\_train, callbacks=[loss\_cb], dataset\_sink\_mode=True)      print(loss\_cb.per\_eval)      metricDic = loss\_cb.per\_eval      return model, metricDic  network\_earlyStop = ForwardFashion\_baseline(cfg.num\_classes)  loss\_fn = nn.CrossEntropyLoss()  model\_earlyStop, metricDic\_earlyStop = train\_earlyStop(network\_earlyStop, loss\_fn)  saveMetric("./metric/baseline\_metric\_EarlyStop.txt", metricDic\_earlyStop)  saveModel("./model/baseline\_model\_EarlyStop.ckpt", network\_earlyStop)  *# 绘图*  drawAndSaveFig("./img/metric\_EarlyStop", metricDic\_earlyStop, "metric\_EarlyStop")  *# -------------------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*  class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']  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*  ds\_test, \_ = create\_dataset()  test\_ = ds\_test.create\_dict\_iterator(output\_numpy=True).\_\_next\_\_()  predictions = model\_regL2.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])  *# 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() | | | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程、实验总结与体会  1、实验运行结果截图  Baseline效果截图（未使用任何正则化技术）    L1正则化效果截图    L2正则化效果截图    Dropout正则化效果截图    BatchNorm正则化效果截图    Dropout+BatchNorm正则化效果截图    Early Stopping效果截图    最终预测结果截图  2、程序编写过程中遇到的问题及解决思路、方法等  本次实验遇到的问题主要是对mindspore框架的callback回调函数的不熟，主要是mindspore的训练过程直接封装到train函数里面去了，然后对mindspore的设计思路不熟。最后查阅文档，发现mindspore对于每个模型设计了一个metric类，metric类可以传入常见的Accuracy和Loss作为度量标准进行计算，最后callback函数里面可以读取metric来打印中间的loss和accuracy变化过程。最后主要的问题还是来自于对mindspore设计的不熟，这只能多查文档，多查资料一步步解决。  另外，在实验时，发现L1和L2正则化使用后，分类效果急剧下降，接近于随机：    通过分析公式并思考，正则项是权重大小的绝对值或绝对值平方之和，此时正则化项的超参数系数如果设置得过大，模型将会倾向于使模型的所有权重趋于0，导致欠拟合。因此，正则化项的超参数选取十分重要。  3、实验总结和体会  **①正则化效果前后对比**  本次实验实际上有6组实验：baseline（即没有任何正则化技术使用），L1正则化，L2正则化，Dropout，BatchNorm，Dropout+BatchNorm以及Early Stopping。  从结果上看，L1正则化，L2正则化使用后的效果和Baseline的效果实际上差不多：训练集和测试集上的准确率相近，训练集上的loss一直下降，测试集上的Loss是先下降后上升，表现出了过拟合的特征。值得一提的是，L1和L2的超参数的选取十分重要，最开始选择的是1e-4的正则化系数，结果发现准确率急剧下降，猜测是该系数对于这个模型的权重过大，导致模型无法学到特征。后来换成了1e-8，才能正常学习。但这里有一个问题，是这个正则化系数的选取有没有抑制过拟合的问题，从结果上看并没有。然而，通过进行实验确定最佳正则化系数的选取的代价是昂贵的。  Dropout这一组的表现良好，主要体现在测试集上的准确率没有明显变化外，测试集上的loss一直下降，没有出现先下降后上升的问题。  BatchNorm正则化技术使用后，在测试集上的准确率略有提升（超过了92%，Baseline，L1，L2和Dropout均在90-92%之间），但测试集上的Loss也是先下降后上升。  而BatchNorm和Dropout两者的结合，体现了上述的两个优点：在测试集上的准确率略有提升，并且测试集上的Loss没有体现过拟合的特征（即先下降后上升的特征）  **②Early Stop效果及分析**  根据相关文献，Early Stop的思想是测试集上的某些指标变差时就停止试验。本次实验，我选择观测训练集上的准确率指标，当连续超过两轮时，准确率不能提高1%时，则停止实验。从结果上看，在不牺牲准确率的同时，loss也没有出现先下降后上升的趋势，即缓解了过拟合问题。  **③正则化技术的特点及使用范围**  通过这次实验，可以做如下总结：  L1正则化技术和L2正则化技术往往更适合希望产生稀疏解的情况下使用。结合具体公式，L2正则化技术更适合产生平滑解。但正则化系数的选取是一个问题。如果正则化系数选择过大，会导致模型的欠拟合；选择过小，则不能发挥缓解过拟合的作用。  Dropout的作用就是在训练过程中，每次随机使网络中的某些单元失效，减少各个节点之间的相互作用，提高泛化能力。其适用于缓解模型的过拟合问题，效果比较显著。  BatchNorm更大的作用在于提升模型的准确率，而正则化效果不太明显。  Early Stopping的思想很简单：在测试集上的效果变差就不再训练，这一思想可以直接帮助缓解过拟合的问题。适合在不希望模型出现过拟合的场景使用。  **④最终总结与体会**  最后的感受是纸上得来终觉浅，绝知此事要躬行。本来以为是只要能够用了正则化技术，就一定能解决过拟合的问题，但其中的细节在做之前是不知道的。比如本次的L1和L2正则化，在做之后才发现，正则化系数的选取相当重要，选得不好不仅不能发挥作用，可能使模型变得更差。联系到本次实验一跑一个模型就要等很久，在实际生产生活中，跑更复杂的模型，通过试验确定最佳参数是不可能的。而BatchNorm相当于对网络中间某层的输出进行了标准化处理，联想到机器学习某个实验，进行标准化后模型准确率的蹭蹭上升，也能理解为什么这次实现，BatchNorm的效果是提高准确率了。而Dropout的思想很粗暴，但是在模型训练过程中，缓解过拟合的问题是最有用的，效果最明显的。最后的Early Stopping也并不复杂，很符合直觉——一旦有变差的趋势就不训练了。这是符合直觉的。 | | | | | | | | |

实验报告填写说明：

1、第一、二部分由老师提供；

2、第三部分填写源程序和算法，源程序要符合程序编写风格（缩进、注释等）；

3、第四部分主要填写程序结果（截图）、解决问题的方法、总结和体会等；

4、报告规范：包含报告页眉、报告的排版、内容是否填写，命名是否规范等。

5、源程序和实验报告命名：学号姓名序号.py 学号姓名序号.docx，例如学号20181234的张三同学，**他的第一次实验命名为：20181234张三1.py和20181234张三1.docx**

**6、向蓝墨云提交两个文件，一个实验报告，一个程序文件（代码合并）**