AI1603 大作业报告 李嘉睿

一、问题描述

1.1 什么是独立同分布（I-I-D）

独立同分布是指测试数据分布与训练数据分布相似。在机器学习的任务中，我们一般假设数据服从独立同分布，但对于实际问题这个假设并不成立。例如，我们在训练一个模型，让它对图片中的奶牛和骆驼进行分类。在训练集中，奶牛大多在草地上，骆驼总是在沙漠上，这样会导致我们的模型认为背景也是动物特征的一部分。当测试集的一张图片的内容是骆驼站在草地上时，我们的模型就会大概率把这只骆驼认定为奶牛[1]。这也说明，这样的模型的泛化效果并不好。

1.2 Colored Mnist训练集

它是Mnist训练集经过修改后产生的新数据集。在其中，小于5的标签被重新标记为0，大于等于5的标签被重新标记为1，接着把六万张图片平均分为三组，两组作为训练集而一组作为测试集。有25%的图片的标签被反转，这样可以模拟实际问题的数据。同时，根据标签将图像着色为红或绿来创建伪相关性。具体来说，在训练集1、2中，标签为0的图片的80%、90%被着红色，标签为1的图片的80%、90%被着绿色，但在测试集中，标签为0的图片的90%被着绿色，标签为1的图片的90%被着红色，与训练集中大体相反。[1]

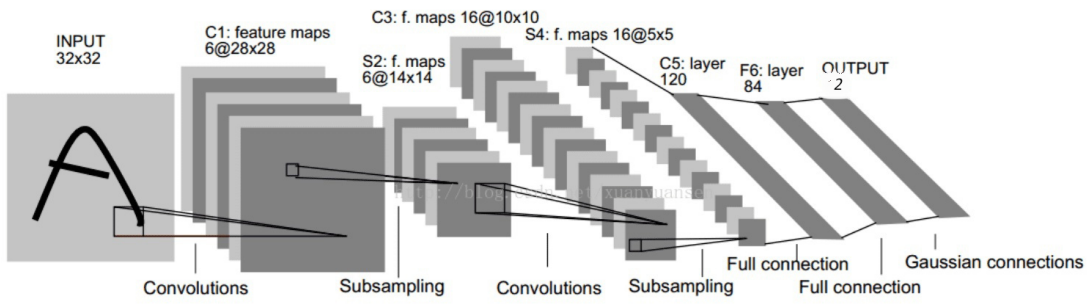
数据集	标签反转	确定颜色占比	理论准确值
训练集1	25%	80%	75%
训练集2	25%	90%	75%
测试集	25%	10%	75%

也就是说，如果用箭头代表因果关系，Mnist训练集能被表示为：数字 -------> 标签 --------> 颜色。从图中我们看出，标签只与数字的值有关，而跟颜色无关。一般来说，数字被称为因果特征，颜色被称为非因果特征。由此也可以看出，因果特征在任何环境下都是不变的。

二、LeNet实现

2.1 LeNet模型

Lenet 是一系列网络的合称，包括 Lenet1 - Lenet5，由 Yann LeCun 等人在 1990 年提出。Lenet是一个 7 层的神经网络，包含 3 个卷积层，2 个池化层，1 个全连接层。其中所有卷积层的所有卷积核都为 5 x 5，步长 stride=1，池化方法都为全局 pooling，激活函数为 Sigmoid，网络结构如下：

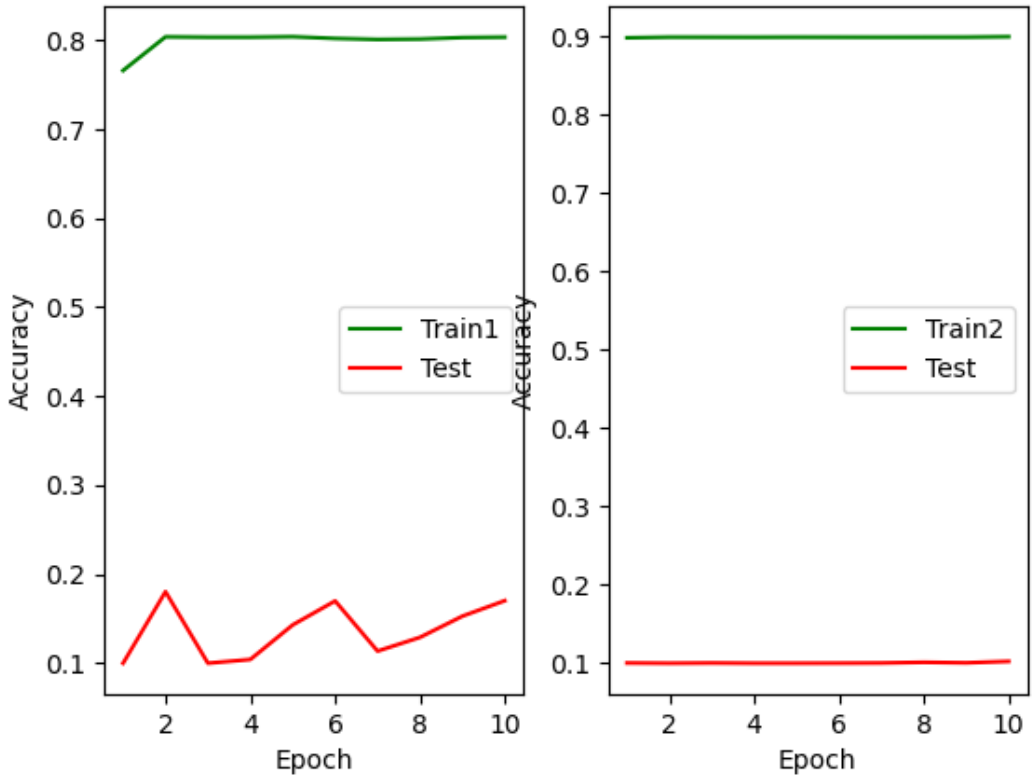


代码实现部分如下：

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 6, 5, 1, 2),nn.Sigmoid(),nn.MaxPool2d(2, 2),
            nn.Conv2d(6, 16, 5, 1, 0),nn.Sigmoid(),
            nn.MaxPool2d(2, 2),nn.Flatten(),
            nn.Linear(400, 120),nn.Sigmoid(),nn.Linear(120, 84),nn.Sigmoid(),
            nn.Linear(84, 2),
        )

    def forward(self, x):return self.model(x)
```

2.2 LeNet学习效果

下面是LeNet网络在train1和train2的前10次训练中的效果，由训练集准确率和测试集准确率展现。由图可知，train1和train2训练结果相似，都是灾难性的。train1的准确率在0.8左右，train2的准确率在0.9左右，而两者的测试集准确率都在0.1上下波动，这说明LeNet模型在训练集上的表现很好，但是在测试集上的表现极为糟糕，而且这个比例也和Mnist中颜色与标签的比例相同。很明显，它只学习到了颜色和标签之间的伪关系，并没有深入挖掘到形状与标签的真正关系。



2.3 数据预处理

由于LeNet如此糟糕的表现，我们希望给数据集进行一个数据预处理，来提高LeNet的泛化能力。我们对数据集进行了缩放、平移、小幅度旋转、和一次选取多个样本进行训练。具体的代码和过程可以详见杜亦开同学的作业报告。

三、IRM（Invariant Risk Minimization）算法

3.1 基本思想

IRM（Invariant Risk Minimization）是2019年Martin Arjovsky等人提出的一种用于跨域图像分类的新方法，它用于解决有的时候训练集和测试集数据存在差别的问题。IRM算法所寻找的是一种稳定的分类函数，它在不同的环境下的效果都很好，而这种分类函数的分类依据就是因果特征变量。一般来说，以非因果变量为分类依据的分类函数都是不稳定的，它们可能在环境A中表现很好，而在环境B中表现很差。例如，上文的LeNet就是基于颜色这个非因果变量的分类模型，它在训练集中的表现很好，在测试集中的表现却一塌糊涂。所以IRM设置了一些惩罚机制，把稳定的分类函数的风险通过一些数学推导，写成一个凸损失函数，以方便机器学习求解。[1]

3.2 公式理解

IRM背后融入了丰富的数学知识，由于对其中一些数学知识的欠缺，我没有完全理解IRM的核心公式，但是也可以完成一些推导过程。

$$\min_{\Phi:\mathcal{X}\to\mathcal{Y}} \sum_{e\in\mathcal{E}_{\mathrm{tr}}} R^e(\Phi) + \lambda \cdot \left\|\nabla_{w|w=1.0} R^e(w\circ\Phi)\right\|^2, \quad (\ \mathrm{IRMv1}\ )$$

其中，第一项是在环境e中的分类风险，它可以理解为原本的损失函数。它形式多样，可以是MSE损失或交叉熵损失等等，而IRM算法的精髓主要集中在第二项，它用来控制分类函数的稳定性。下面我尝试着进行一些推导，将惩罚项一步步写成一个凸损失函数。

1. 先写出原文的目标损失函数

   $\mathcal{L}(\Phi,w)=\sum_e R(w\circ\Phi)+\lambda D(\Phi,w,e)$，其中第二项代表了$w$距离最小化$R(w\circ\Phi)$的$w$的距离，这就相当于一个惩罚项，如果$w$离最小化$R(w\circ\Phi)$越远，这一项的值就会越大，这个损失函数就会越大，所以可以通过梯度下降来找到它的最小值。而$\lambda\in[0,+\infty)$是一个可调的超参数，用于平衡经验损失（第一项）和不变损失（第二项）。

2. 为方便推导，假设Y与X是线性关系，即

   $$\mathcal{Y}^e = w\cdot\Phi(\mathcal{X}^e), w\ 是\mathcal{Y}和\mathcal{X}\ 的线性关系矩阵$$

3. 写出w的最小二乘解，并将逆矩阵消掉

   $$w_\Phi^e = [\Phi(\mathcal{X}^e)^T\cdot\Phi(\mathcal{X}^e)]^{-1}\Phi(\mathcal{X}^e)^T\mathcal{Y}^e$$

   $$\Phi(\mathcal{X})^T\mathcal{Y}^e - \Phi(\mathcal{X}^e)^T\Phi(\mathcal{X}^e)w_\Phi^e = 0$$

4. 所以我们的w与最小化$R(w\circ\Phi)$的w_0的距离可用表示为

   $$D(\Phi,w,e) = ||\Phi(\mathcal{X})^T\mathcal{Y}^e - \Phi(\mathcal{X}^e)^T\Phi(\mathcal{X}^e)w||^2$$

5. 然后发现这一项**正好是最小二乘法的偏导数**。记$C(w)=\Phi(\mathcal{X}^e)w-\mathcal{Y}$，则有 [2]

   $$||\Phi(\mathcal{X})^T\mathcal{Y}^e - \Phi(\mathcal{X}^e)^T\Phi(\mathcal{X}^e)w||^2 = \frac{\lambda}{2}\frac{\partial}{\partial w}[(\Phi(\mathcal{X}^e)w-\mathcal{Y})^T(\Phi(\mathcal{X}^e)w-\mathcal{Y})] = \frac{\lambda}{2}\frac{\partial}{\partial w}C(w)^TC(w)$$

6. 固定w为w_0，当假设$\mathcal{X}$和$\mathcal{Y}$为线性关系时，$C(w)$就是熟知的均方误差损失函数MSELoss，即$C(w)=\Phi(\mathcal{X}^e)w-\mathcal{Y}=R(w,\Phi)$。于是原目标损失函数可以写为

   $$\mathcal{L}_{w_0} = \sum_{e\in\mathcal{E}_{\mathrm{tr}}} R^e(w\circ\Phi) + \lambda||\nabla_w R^e(w,\Phi)_{w_0}||^2$$

   值得说明的一点是，这里的风险函数R默认为均方误差损失函数，但其实当用于分类问题时，R也可以是交叉熵函数等等。而且我们可以发现，最后得出的损失函数的两项有着较为相似的形式，它们都是从经典损失函数$R(w\circ\Phi)$得出来的，可以用python程序去较好的模拟。

至此我完成了公式的部分推导。从原文的目标损失函数出发，我推出了IRM算法的核心数学公式。在整个推导过程中，我认为步骤5是最为精妙的，它地将距离D用损失函数的偏导数表示，从而统一了损失函数中看似完全不同的两项，并使其易于实现，真是巧夺天工！

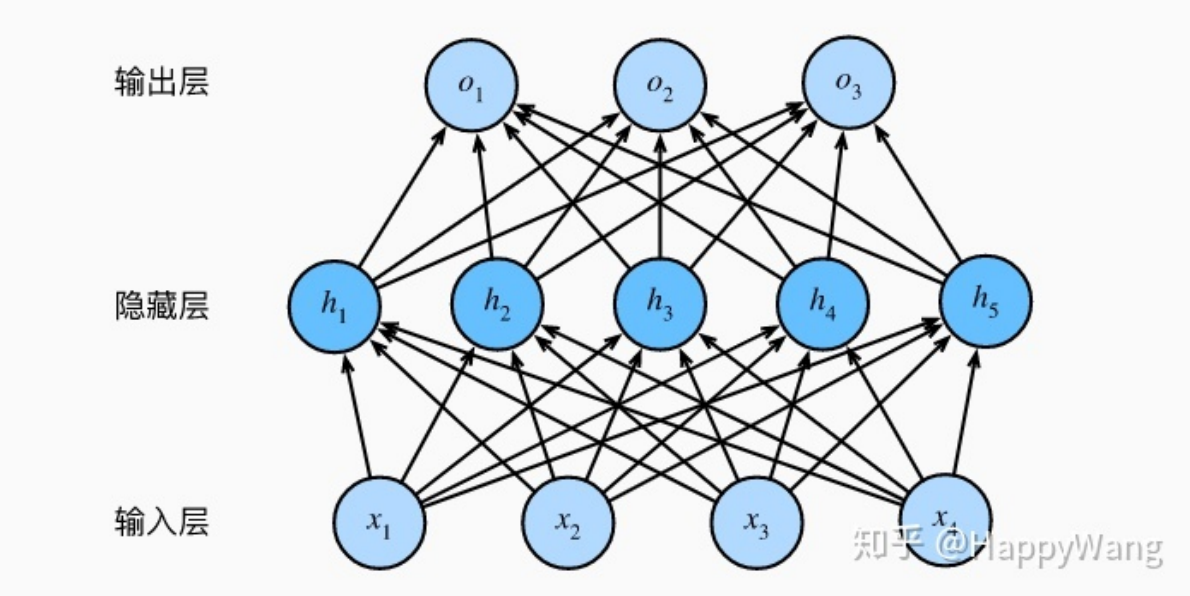
3.3 IRM算法程序复现

3.3.1 网络的选取

在IRM原文中，作者没有使用LeNet卷积神经网络作为baseline，而是使用了多层感知机MLP。经我分析，原因最主要是MLP更适合与IRM算法的实现。对于特别复杂的网络来说，它的参数很多，IRM算法的训练效果不是那么好。刚开始我使用LeNet来实现IRM算法的结果，结果不尽如人意，它的测试集和训练集的准确率都在50%左右浮动，这跟随机猜的准确率相同，也说明LeNet网络并不与IRM算法适配。

3.3.2 MLP多层感知机

多层感知机（MLP）是一种前向结构的人工神经网络，包含输入层、输出层及多个隐藏层，3层感知机的神经网络图如下所示：[3]



在本题中，出于对参数数量和简化模型的考虑，我们使用了只有一个隐藏层的多层感知机，用它来替代一开始的LeNet网络，并且为了增加模型的稳定性，我们对MLP网络使用Xavier初始化和全0初始化。如下是它的python代码实现：

```
class MLPModel(nn.Module):
    def __init__(self):
        super(MLPModel, self).__init__()
        layer1 = nn.Linear(3 * 28 * 28, 256)
        layer2 = nn.Linear(256, 256)
        layer3 = nn.Linear(256, 1)
        for layer in [layer1, layer2, layer3]:
             nn.init.xavier_uniform_(layer.weight)  # Xavier初始化
             nn.init.zeros_(layer.bias)             # 全0初始化
        self.model = nn.Sequential(nn.Flatten(),layer1,
                                   nn.ReLU(True),layer2,
                                   nn.ReLU(True),layer3,
                                   nn.Sigmoid()
        )

    def forward(self, inputs):
        return self.model(inputs)
```

3.3.3 IRM核心部分复现

由3.3.1的公式推导可知，最终的损失函数有两项，这里我们来分别用python程序来实现。

第一项：$\sum R^e(w \circ \Phi)$。这一项较好实现，因为这就是最原始的损失函数。对本题，我们的函数R使用交叉熵函数。

```
loss = torch.nn.CrossEntropyLoss(reduction = "none")
# reduction = “none”可以使计算出来的loss仍然是向量
for x,y in train_iter:
    x=torch.flatten(x,start_dim=1) # 展平图片
    y_hat = net(x) * dummy_w       # 这里的dummy_w为第二项的计算做准备
    error_e = loss(y_hat,y)
```

第二项：$\sum \left\| \nabla_{w|w=1.0} R^e(w \circ \Phi) \right\|^2$。这一项也较好实现，并且在使用随机梯度下降的时候，这一项可以改写为

$$\sum_{k=1}^{b} [\nabla_{w|w=1.0} \mathcal{L}(w \circ \Phi(\mathcal{X}_k^{e,i}), \mathcal{Y}^{e,i}) \cdot \nabla_{w|w=1.0} \mathcal{L}(w \circ \Phi(\mathcal{X}_k^{e,j}), \mathcal{Y}^{e,j})]$$

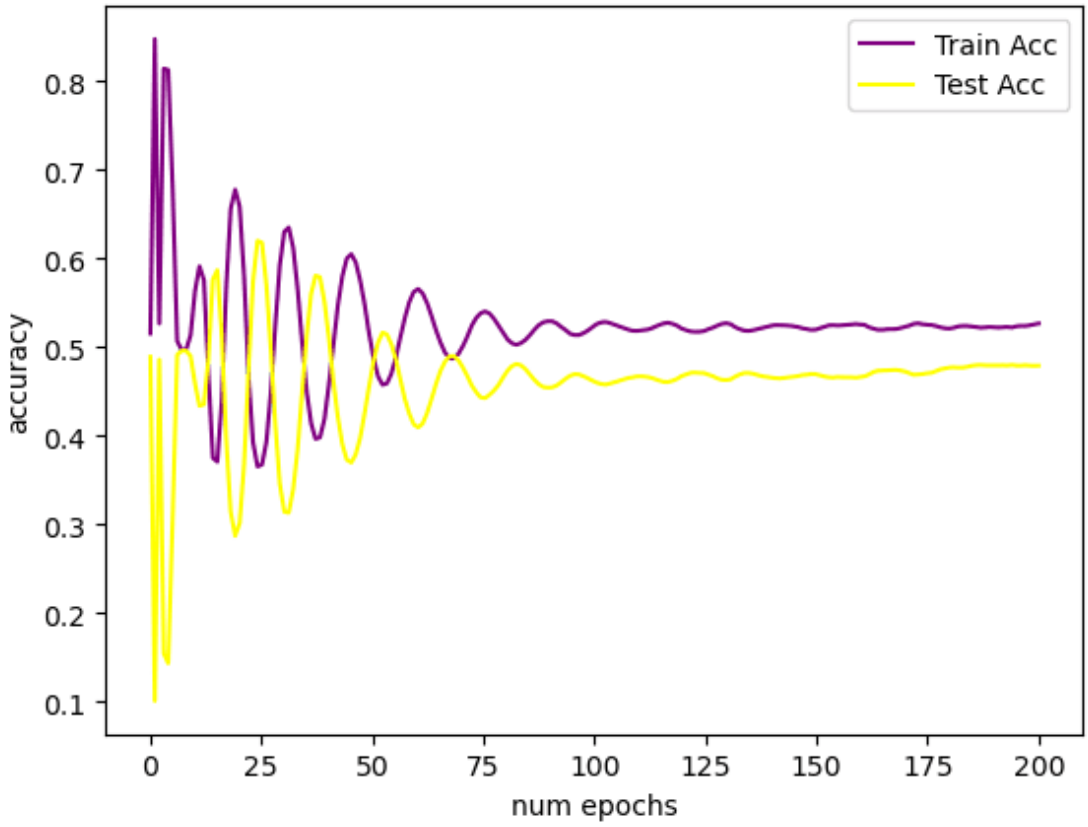
其中$(\mathcal{X}^{e,i}, \mathcal{Y}^{e,i})$和$(\mathcal{X}^{e,j}, \mathcal{Y}^{e,j})$是环境e的两个随机小批量数据，$\mathcal{L}$是损失函数。在实现的时候，我们可以让它们是环境e中的所有数据，或者是奇偶数据等等。在这里我们使用全部数据。

```
dummy_w = torch.nn.Parameter(torch.Tensor([1.0])) # 这个就是w
def compute_penalty(losses, dummy_w):
    g = grad(losses.mean(), dummy_w, create_graph=True)[0]
    return ( g * g ).sum()
```

这样我们就计算好了全部的核心损失函数。

3.3.4 训练结果

在这次训练中，我按照IRM论文给的示例代码，将超参数λ设置为[10,100,1000,10000]，并且循环多次，最终选中效果最好的1000.0作为λ的最终值。下图展示的是这个最原始的IRM的训练效果。



这幅图可以看出，训练集和测试集的准确率在经历过60次循环以后逐渐稳定下来，最终训练集准确率在55%左右，测试集准确率在50%，可见这个最原始的IRM模型仍然学到了一些形状与标签的关系，但是效果还不是很好，50%左右的测试集准确率并不能让我满足，于是我们对这个模型进行了一些改进。

3.3.5 超参数λ的改进

通过对λ赋予不同的取值，我发现整个模型对这个惩罚权重是极为敏感的，如果取小有不同的权重值，最终的结果会发生极大的区别。关于敏感性分析问题，可以详见杜亦开同学的报告。

在对最优的λ的值进行探索的时候，我们连连碰壁，最好结果也就是测试集51.2%准确率。于是我们突然想到，能不能把λ化固定为活动，让它跟随着模型的改进而更正为合适的值。同时我们发现，随着训练次数不断增加，损失函数的第二项（也就是惩罚项）会逐渐减小，最终趋近于0，这也就意味着惩罚项最后不复存在，IRM算法也会退化到原来的不用惩罚项的算法。所以我们希望这个惩罚权重变得越来越大，以此来抵消惩罚项的减小。

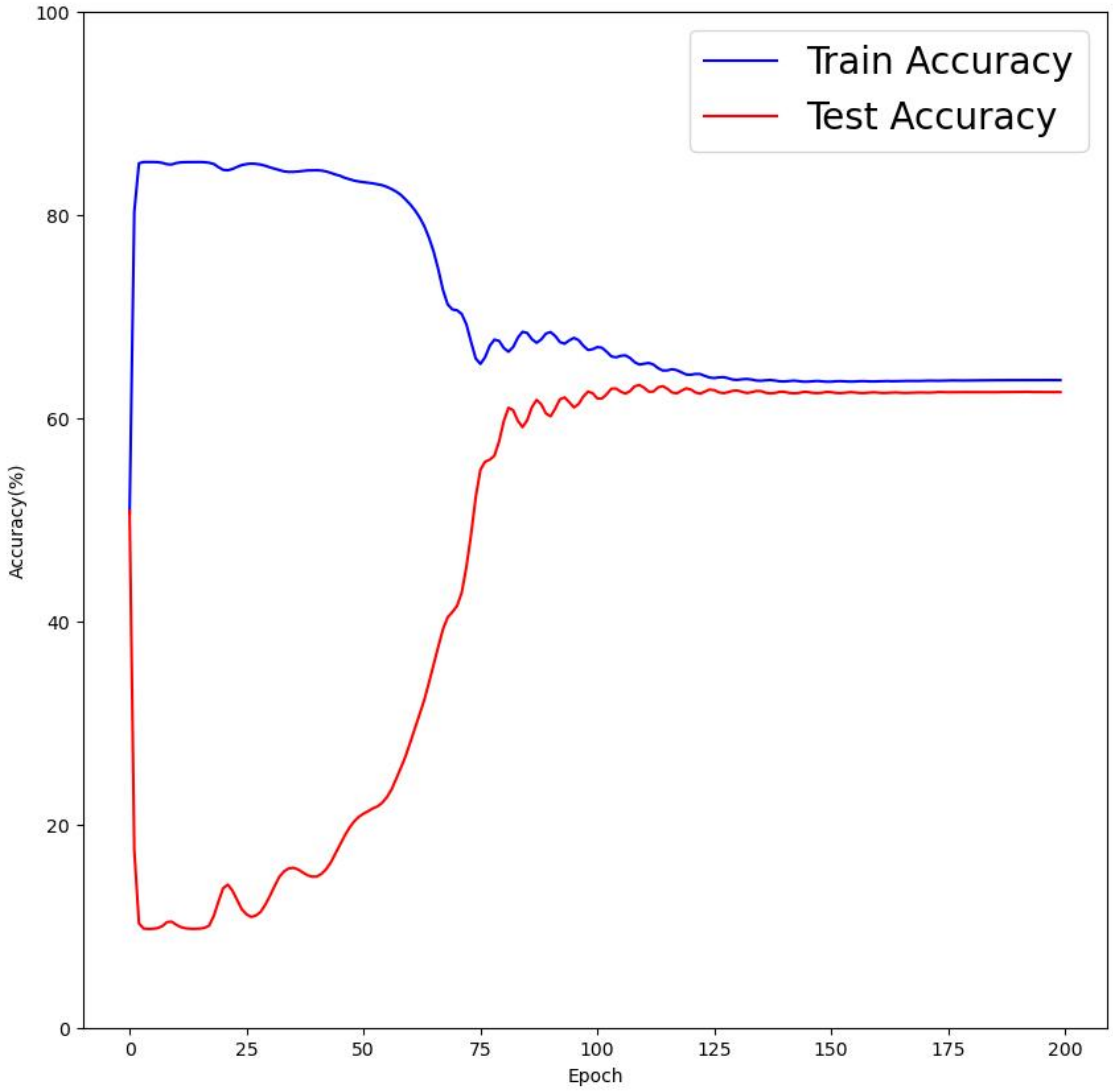
基于以上想法，我们尝试将整个惩罚值“ 固定 ”为一个不算太小的与网络有关的值。经过挑选，我们选择了MLP网络所有参数的范数的平方和。这样，当经过多次循环训练，网络参数逐渐收敛到最优值的时候，惩罚值也不至于趋于0。

```
# 代码实现

parameter_norm = torch.tensor(0.).to(device)
    for param in net.parameters():
        parameter_norm += param.norm().pow(2)     # 计算网络所有参数的范数

penalty_weight = parameter_norm.item() / penalty # “固定”惩罚值
total_loss = error.clone()
total_loss += penalty * penalty_weight           # 损失函数计算
```

同时，为了防止出现过拟合的情况，我们可以再对模型进行正则化，在原有损失函数上加上正则项：

```
norm_weight = 0.001
total_loss += norm_weight * parameter_norm
```

这样我们对超参数λ的优化完成了。训练结果如下（有数据增广过程）：



可以看出在调整λ和添加正则化后，模型的测试集和训练集的准确率都在63%左右，效果较好！到这里，我们较为完美的复现了IRM算法，并对它进行了一些优化。

四、AndMask算法

4.1 概述

AndMask算法是由Giambattista在2019年提出的。这种算法的核心在于，作者发现一般的梯度下降方法是类似于逻辑或（∨）的模式，但这种方法没有办法找到稳定的模型。于是他们开发了一种基于逻辑与（∧）新型梯度下降方式，并且发现了这种梯度下降的方式对于解决OOD问题是很有帮助的。

4.2 公式理解

有关AndMask算法的知识在网上普遍很少，所以以下内容均出自于我对***Learning Explanations That are Hard to Vary***论文的理解与分析。下面我就尝试去跟着论文的思路，推导一下里面的数学公式。[4]

1. 首先文章中定义了集合$\Theta^*_{\mathcal{A}}$,它表示算法$\mathcal{A}$在用所有环境的数据进行训练的时候的收敛点，换句话说它就是所有的局部最小值点。接着对于给定的数据环境e定义了集合$N^{\epsilon}_{e,\theta^*}$，它是空间上最大的路径联通区域，并同时包含了$\theta^*$和集合$\{\theta \in \Theta \ s.t. \ |\mathcal{L}_e(\theta) - \mathcal{L}_e(\theta^*)| \leq \epsilon\}$，利用它给每个收敛点定义了不稳定度（inconsistency）。

$$\mathcal{I}^{\epsilon}(\theta^*) = \max_{(e,e^1)\in\varepsilon^2} \max_{\theta\in N^{\epsilon}_{e,\theta^*}} |\mathcal{L}_{e^1}(\theta) - \mathcal{L}_e(\theta)|$$

2. 接着就可以定义算法$\mathcal{A}$的稳定学习一致性得分（invariant learning consistency）：

$$ILC(\mathcal{A}, p_{\theta_0}) = -\mathbb{E}_{\theta^0\sim p(\theta^0)}[\mathcal{I}(\mathcal{A}_{\infty}(\theta^0, \varepsilon))].$$

3. 由$H_A = \nabla^2\mathcal{L}_A(\theta^*)$，$H_B = \nabla^2\mathcal{L}_B(\theta^*)$定义新的Hessians矩阵，并通过矩阵的均值不等式可知

$$H_{A\wedge B} = H_A \wedge H_B，\text{就是每一项对应相与（}\wedge\text{）所构成的矩阵}$$

$$0 \le det(H_{A\wedge B}) \le det(H_{A+B})$$

因此当A、B的形状十分相似时，不确定性就很小。

4. 计算梯度：为了简化问题，假设每个H_e都是对角矩阵并且元素都为正，那它们的几何平均可以计算为$H^{\wedge} = diag((\prod_{e\in\varepsilon}\lambda_1^e)^{1/|\epsilon|}, \dots, (\prod_{e\in\varepsilon}\lambda_n^e)^{1/|\epsilon|})$。所以我们用于下降的梯度是$\nabla\mathcal{L}^{\wedge}(\theta) = H^{\wedge}(\theta^k - \theta^*)$，再通过这个值去更新模型的参数：

$$\theta^{k+1} = \theta^k - \eta H^{\wedge}(\theta^k - \theta^*)$$

4.3 简化理论

在4.2中我推出了它的基本公式，但是这个公式过于复杂，很难用python进行实现。所以接下来，论文对公式做了一些简化，并最终提出了AndMask算法。

1. 把在不同环境中，梯度的正负不稳定的项的梯度设置为0（即mask），这可以通过梯度矩阵（G）点乘mask矩阵（M，且是已经平均后的）得到（mask矩阵中的元素是0或者1，不稳定的项对应乘0，而稳定的项对应乘1），由于$1\wedge a = a, 1\times a = a$，$0\wedge a = 0, 0\times a = 0$，所以这种矩阵相乘的形式就相当于两者相或。

2. 在本题中，不同环境指的是每张图片不同的R、G、B三个图层，所以每个网络参数的位置，都会有三个参数，分别代表三个图层的参数值，从而它们的梯度矩阵也存在三个值，分别代表三个图层参数的梯度。要实现这个，首先要计算出参数梯度的符号矩阵（S）。对每个梯度值正则设为1，负则设为-1，0仍然设为0。如果某个位置三个符号值之和的平均值的绝对值小于一个超参数tau，我们就认定这个参数不够稳定，它对应的mask矩阵中的值应该是0。

3. 在这里，tau设定为$(0, 1)$范围内的随机数，而且可以看出S矩阵中的每个元素都应该是$0, \pm1, \pm\frac{2}{3}, \pm\frac{1}{3}$中的一个。下面是个计算的例子（$M_1$、$M_2$、$M_3$表示同一位置不同图层的梯度）：

设$M_1 = \begin{bmatrix} 0.5 & 1.3 & -2.1 \\ 3 & 3.4 & -5 \\ -6 & 3.5 & -2.8 \end{bmatrix}, M_2 = \begin{bmatrix} -1 & 1 & -2 \\ -3 & 4 & -5 \\ -6 & 7 & -8 \end{bmatrix}, M_3 = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}, tau = 0.5,$

这样M_1的符号矩阵为$\begin{bmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix}$，$M_2$和$M_3$的符号矩阵同理计算，于是$M$的符号矩阵

$S = \begin{bmatrix} 0 & 1 & -\frac{1}{3} \\ \frac{1}{3} & 1 & -\frac{1}{3} \\ -\frac{1}{3} & 1 & -\frac{1}{3} \end{bmatrix}$，于是mask矩阵$M = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$。

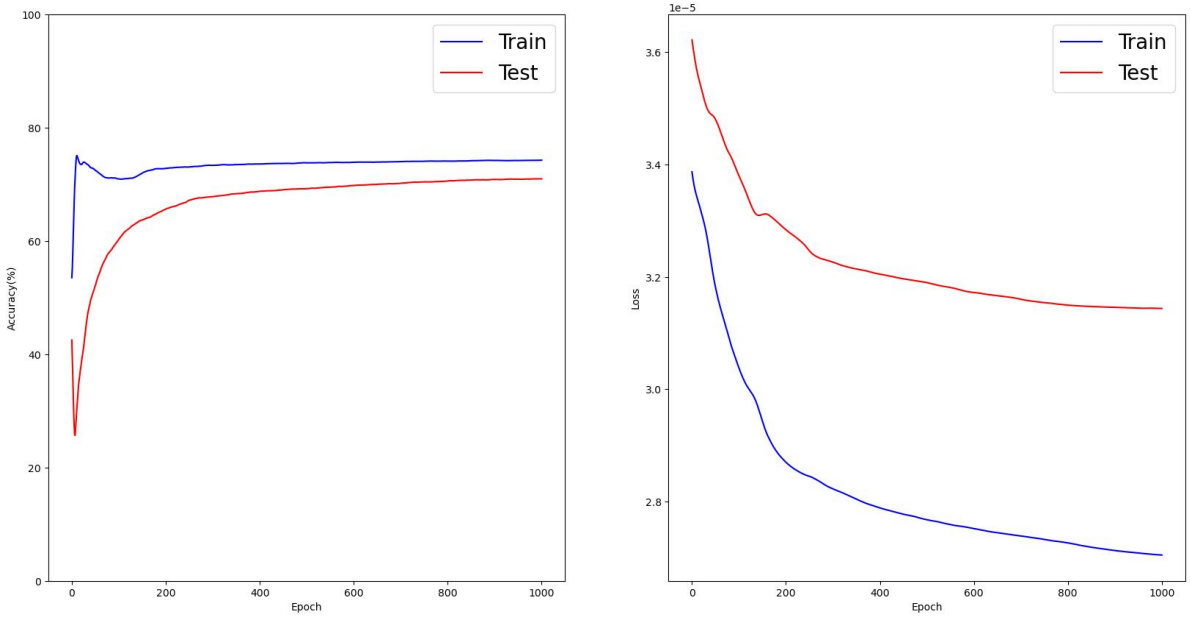
4.4 代码实现

这是实现AndMask算法的代码：[5]

```
def mask_grads(tau = np.random.uniform(0, 1), gradients, params):
    for param, grads in zip(params, gradients):
        grads = torch.stack(grads, dim=0)
        grad_signs = torch.sign(grads)                          # 取出每个梯度的符号
        mask = torch.mean(grad_signs, dim=0).abs() >= tau       # 这里就是mask矩阵，
mean()是对三                                                              个数求平均
        mask = mask.to(torch.float32)
        avg_grad = torch.mean(grads, dim=0)

        mask_t = (mask.sum() / mask.numel())
        param.grad = mask * avg_grad                            # 梯度的重新计算，有一些
梯度被屏蔽
        param.grad *= (1. / (1e-10 + mask_t))                   # 梯度的重新计算

optimizer.zero_grad()
mask_grads(param_gradients, net.parameters())                   # 新的梯度下降
optimizer.step()
```

4.5 结果展示

这是AndMask算法训练的结果：



可以看出，AndMask算法对于这个Colored Mnist测试集的适配度非常好。训练集准确率在75%左右，测试集准确率高达70%，而且肉眼可见的还在缓慢增加，甚至优于优化后的IRM算法！

五、探索OOD算法

5.1 IRM的推理能力与收敛性

1. IRM的损失函数由两项组成，一项是经典的损失函数，这一项是用来提高训练集准确率的，经过梯度下降之后，这一项越变越小，训练集准确率不断提高并稳定；
2. 第二项$\sum_{e\in\mathcal{E}_{\text{tr}}}\lambda\cdot\left\|\nabla_{w|w=1.0}R^e(w\circ\Phi)\right\|^2$，它实际上代表了在**所有环境**中最小化的$w_0$与当前的$w$的距离（前面3.2节已经证明这个），经过梯度下降之后，这一项的值变小，意味着w离最终的w_0更近，也保证了最后筛选出的特征是与结果成因果关系（也就是算法的稳定性）；

3. 这两项各司其职，一项提高准确率，一项控制稳定性，这两项综合起来运用，构成了IRM算法的核心，也是IRM算法在这个OOD问题上表现如此好的原因。

5.2 AndMask的推理能力与收敛性

1. AndMask算法最重要的一点在于，它所取的所有环境指的是R、G、B三个图层，也就是说它正好击中了Colored Mnist的痛点：颜色是干扰变量！
2. 在经过多次训练之后，算法会发现颜色对于标签影响的不稳定性（梯度的符号相差较大），于是它就会逐渐将颜色这个影响因素用mask矩阵进行屏蔽，这个最大的干扰因素被排除在外；
3. 这样的话，模型就会逐渐找到稳定的那个真正影响因素------形状，也就是达成了本题的目标，所以最后的准确率当然很高。

六、总结与创新

6.1 总结

以上就是我和我对小组对大作业的报告。首先，我创建了LeNet网络并运用其在Colored Mnist上训练，最终得到的结果是它并不能分辨出因果关系和非因果关系。其次，我通过IRM论文的阅读复现了IRM算法，发现了模型对惩罚权重的敏感性，并凭借自己的理解对它的超参数λ的选取进行了改进，同时对整个模型进行正则化，显著提升了模型的训练效果和训练稳定性。接着我继续探索其他的OOD算法，主要是AndMask算法，在阅读论文之后对它也进行了复现，而且效果出人意料的好，这也意味着AndMask也是解决这个问题的一个优秀算法。

6.2 创新点

1. 对训练集进行了平移、旋转和批量处理等方式，小小的提升了LeNet的训练效果；
2. 发现了IRM算法对惩罚权重的敏感性，并通过活化惩罚权重和正则化的方式改进IRM算法，使其训练效果更好。

引用

[1] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David LopezPaz. Invariant Risk Minimization. arXiv:1907.02893, 2019

[2] Yearn. Invariant Risk Minimization Reading Notes. https://zhuanlan.zhihu.com/p/273209891, 2022.

[3] HappyWang. Machine Learning(4) Multilayer Perceptron(MLP), 2020

[4] Giambattista Parascandolo, Alexander Neitz, Antonio Orvieto, Luigi Gresele, Bernhard Schölkopf. Learning explanations that are hard to vary. arXiv:2009.00329, 2020

[5] lopezpaz. DomainBed. https://github.com/facebookresearch/DomainBed/blob/main/domainbed/algorithms.py, 2023