大作业实验报告

一、问题简述与测试集简介

在传统机器学习问题中，我们通常会假设各种条件之间的因果分布在整个测试集与训练集上是相似或不变的，并且通过最小化人为预先设定的一个风险值来训练模型，并使模型达到最优的状态，这种训练手段通常被称之为 ERM (Empirical Risk Minimization)。但在现实世界与实际应用中遇到的却大部分不是这种情况。训练集与测试集之间通常会存在某些具有误导性的因果因素，我们将这种现象称之为 Correlation Shift，并将这一类在不同环境之间具有不同因果性因素的问题统称为 OOD (Out of Distribution) 问题。

ColorMNIST Dataset 是在 MNIST 手写数字数据集基础上改造而成的数据集，通常用来测试与衡量模型在存在 Correlation Shift 的情况下的数据泛化能力。该数据集包含从 0 到 9 的手写数字，其中小于 5 的数字带有 label 0，且数字的颜色为红色，大于等于 5 的数字带有 label 1，且数字的颜色为绿色。此外，如图1中所示，该数据集包含三个环境。在 train1 环境中数字的颜色有 20% 的概率是相反的（即带有 label0 的数字为绿色，带有 label1 的数字为红色），在 train2 环境中的数字有 10% 的概率颜色相反，而在 test 环境中的数字有 90% 的概率颜色相反。在该数据集中，数字的颜色是最终 label 的一个无关特征，而数字的大小才应该是算法应该学习的不变特征。此外，在每个环境中，数据集的大小特征都有 25% 的概率被置换，用以模拟真实情景中的数据噪声。

图 1: ColorMNIST Dataset

二、训练与测试 Lenet

Lenet 网络被广泛运用于手写数字识别的任务中，在识别不带有颜色特征干扰的不同 MNIST 数据集的任务中可以达到 90% 以上的准确率，Lenet 网络由 7 层构成，分别是两层卷积层，两层池化层，以及三层全连接层，巧妙的网络设计令 Lenet 具有了出色的特征提取的能力。Lenet 算法的具体代码详见附录。

我们在 ColorMNIST 数据集上训练 Lenet 网络，得到的结果如图2所示，注意到，在训练阶段网络获得了高达 80% 的准确率，但在测试阶段网络的准确率仅仅只有 10% 左右。这是因为网络捕捉到了颜色与 label 的相关性，即使颜色并不是决定最后 label 的本质特征，换而言之，网络学习到了与 label 毫无关联的干扰特征，而忽略了不同环境之间的不变特征。因此传统的 ERM 算法在 OOD 问题上的表现并不尽如人意。

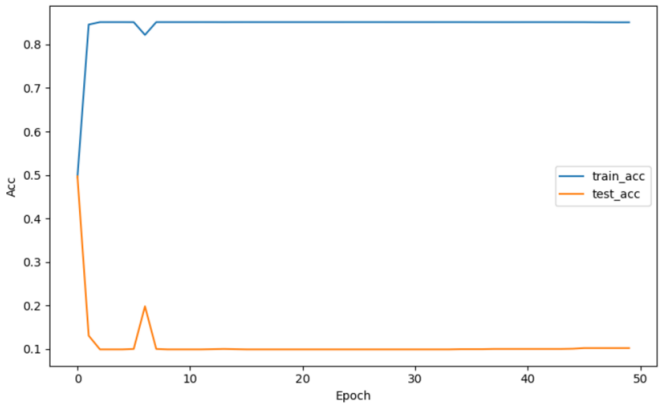


图 2: 在 ColorMNIST 数据集上训练的 Lenet 网络

三、数据增广与网络结构调整

3.1 灰化处理

颜色特征是 ColorMNIST 中最主要的干扰特征，通过灰化函数取消这一颜色特征干扰，我们应该能够获得 75% 的理论最好表现。我们灰化函数的逻辑是在三层通道取平均值，使三层通道的数值是一样的，从而消除颜色的影响。图3展示了我们的灰化函数的代码与最终效果。

```python
def get_mean(img):
    x = torch.mean(img, axis=0)
    x = torch.tensor(np.broadcast_to(x, (3,28,28)))
    return x
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Lambda(lambda img: get_mean(img)),
    transforms.Resize([28, 28])
])
```

(a) 灰化处理代码

(b) 灰化效果

图 3: 灰化处理

如图4所示，在 100 代的迭代之后，在训练与测试集上都获得了近 75% 的准确率，接近理论最大值。

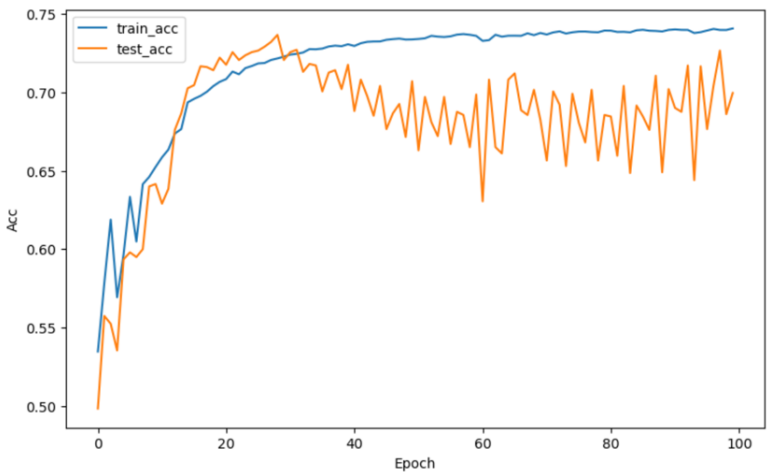


图 4: 采用灰化函数后的训练效果

3.2 高斯噪声

高斯噪声一种引入按照高斯分布随机分布的噪声而削弱图像某些特征的手段，图5是我们实现高斯噪声的代码。

```python
def get_gaussian_noise(img):
    mean = 0
    std = 1
    noise = np.random.normal(mean, std, img.shape)
    noisy_img = np.clip(img+noise,0,255)
    return img
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Lambda(lambda img: RandomGaussianNoise()(img)),
    transforms.Resize([28, 28])
])
```

图 5: 引入高斯噪声的代码

在实验中，我们采用了三个不同的生成高斯噪声的标准差，分别是 1，3，5。图6展示了经过高斯噪声处理后的数据样态。可以观察到，在标准差为 1，3 时数字的形状与颜色的特征还依稀可见，但到了标准差为 5 时，数据的形状已全然消失。图7展示了经过高斯噪声处理过后训练准确度的变化情况，可以看到，算法依然有能力挖掘数据隐藏的颜色特征（即使这些特征已为肉眼所不可见）。

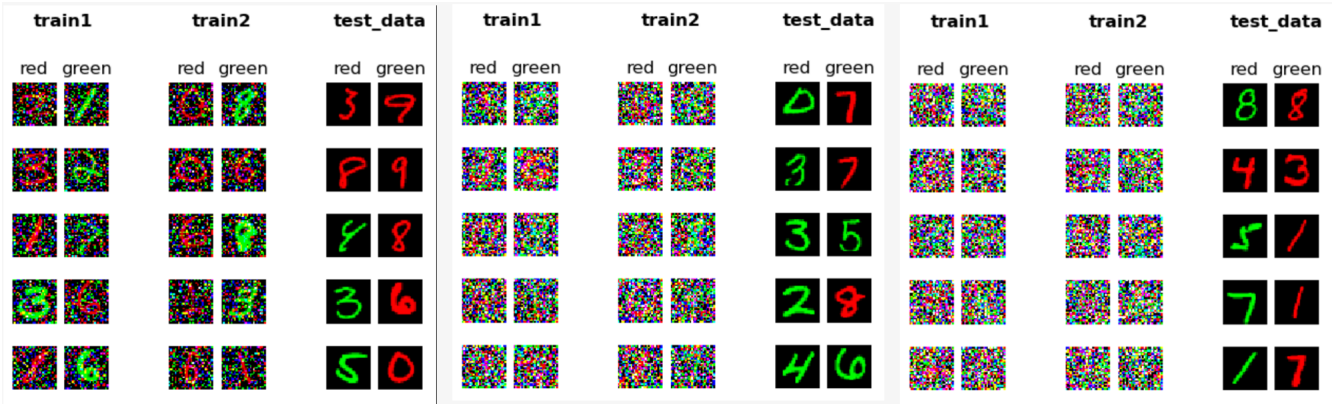
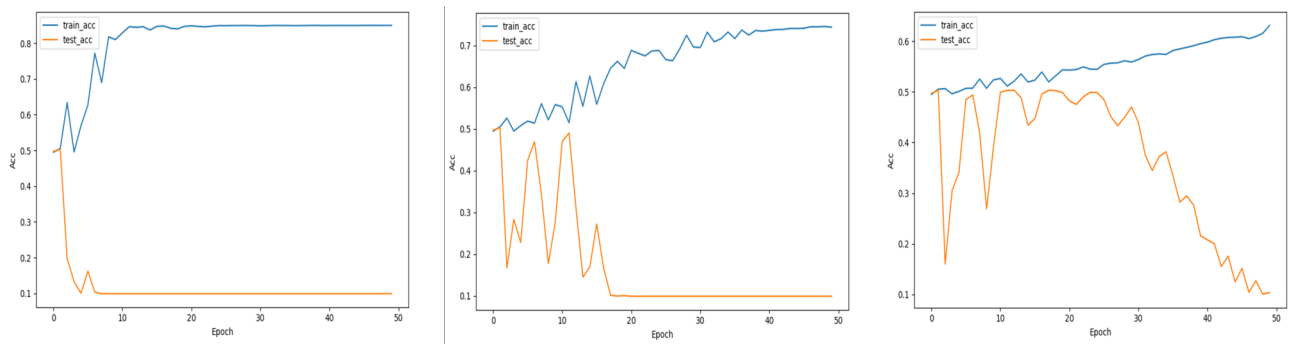


图 6: 经过高斯噪声处理的数据



(a) std=1　　(b) std=2　　(c) std=3

图 7: 应用高斯噪声的数据集样态

3.3 归一化

在对数据进行预处理时加入对数据的归一化化可以减少数据集中内在的协变量平移，从而增加模型在不同环境之间的泛化能力，并且增强模型在训练集上的预测表现。图8展现了经过归一化处理的数据在 50 代之后的测试准确率获得了准确率的提升，证明算法从中学习到了不变特征。

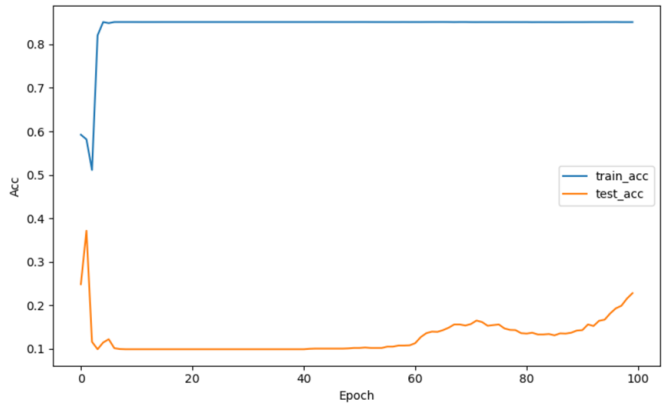


图 8: 归一化训练准确度变化

3.4 增加 BatchNorm 层

通过简单地调整 Lenet 网络，我们可以获得意想不到训练表现增益。我们在每一个池化层后面都跟上一层 BatchNorm 层。BatchNorm 独立地对所有 Channel 在一整批数据的范围内进行归一化，这使得不同环境之间的信息得以传递，从而完成了数据的流通，因此对于不同环境具有一定程度的泛化能力。图9展示了在加入 Batchnorm 层之后的训练准确度变化，可以发现，相较未进行任何针对性处理的训练表现有了显著的提升。

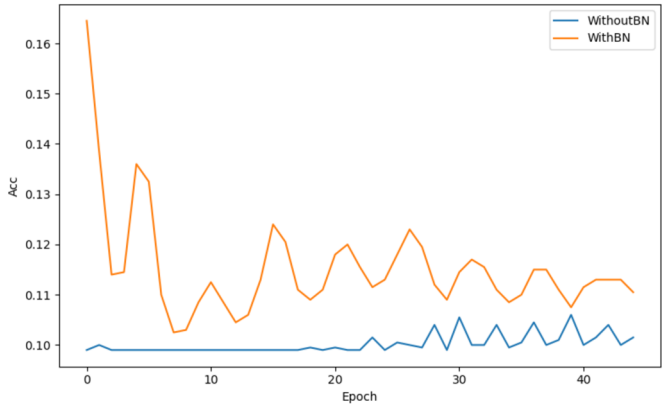


图 9: 增加 BatchNorm 层训练准确度变化

3.5 综合提高表现

我们综合上文了上文所提及的归一化与增加 BatchNorm 层的手段，进行训练并观察表现。如图10所示，经过三百代的训练，网络在测试集上的准确率能够达到 25% 且时而能够达到 30%，这可以说是十分优秀的表现。

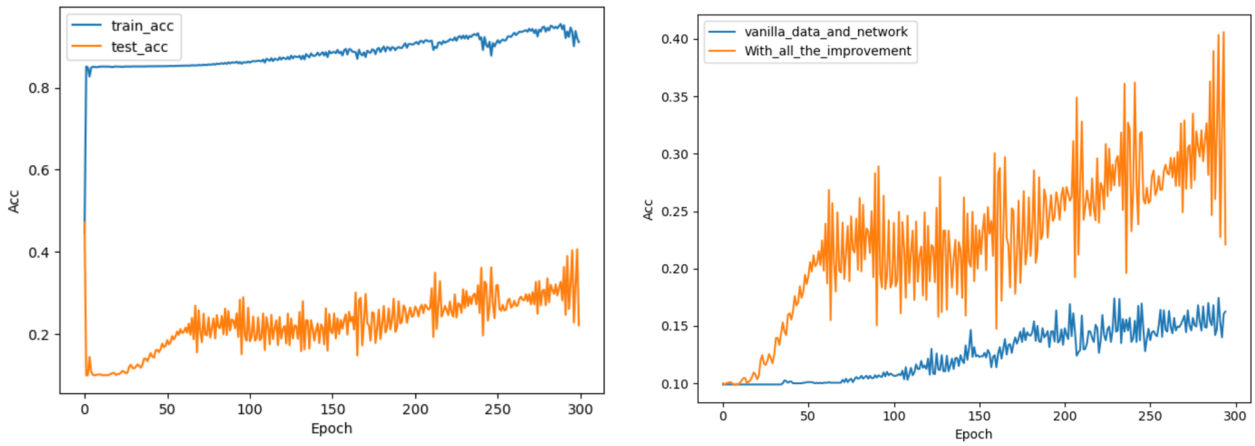


图 10: 综合处理

四、IRM 算法复现

4.1 算法简述

IRM 算法是在 2019 年被首次提出的一种解决 OOD 问题的算法。算法主要的工作是尝试用一个数据映射 $\Phi : \mathcal{X} \to \mathcal{H}$ 提取出数据集在不同环境之下的不变特征 (Invariant Feature)，并在此基础上进行预测。由于对于此不变特征，在不同环境的预测应不受到环境中其它因素的干扰，因此最优化的任务转变为了寻找出这个映射不变数据的数据映射与在所有环境当中寻找最优的分类器。用数学公式表明即为：

$$\min_{\substack{\Phi:\mathcal{X}\to\mathcal{H} \\ w:\mathcal{H}\to\mathcal{Y}}} \sum_{e\in\mathcal{E}_{tr}} R^e(w\circ\Phi)$$
$$\text{s.t.} \quad w \in \arg\min_{\bar{w}:\mathcal{H}\to\mathcal{Y}} R^e(\bar{w}\circ\Phi), \forall e \in \mathcal{E}_{tr}$$

经过复杂的数学变换，最后我们可以将该优化问题的损失函数转化为如下一个数学式子：

$$\mathcal{R}_{\text{IRM}}(\theta) \doteq \sum_{e\in\mathcal{E}_{\text{tr}}} R^e(\Phi(\theta)) + \lambda \cdot \left\|\nabla_{w|w=1.0} R^e(w\cdot\Phi(\theta))\right\|^2 \tag{1}$$

其中 $\left\|\nabla_{w|w=1.0} R^e(w\cdot\Phi(\theta))\right\|^2$ 表示的是对于分类器在不同环境下不同表现的惩罚。

4.2 代码复现

我们复现了论文中使用 IRM 算法在 ColorMNIST 数据集上的训练结果（具体代码详列在附录当中），如图11所示，在经历 200 代的训练之后，模型的训练准确率收敛到了 71.7%，而测试准确率收敛到了 66.6%，这证明 IRM 算法的惩罚项确实使得模型学习到了在各个环境之间都潜伏的不变的特征，并用于提升训练的准确率。

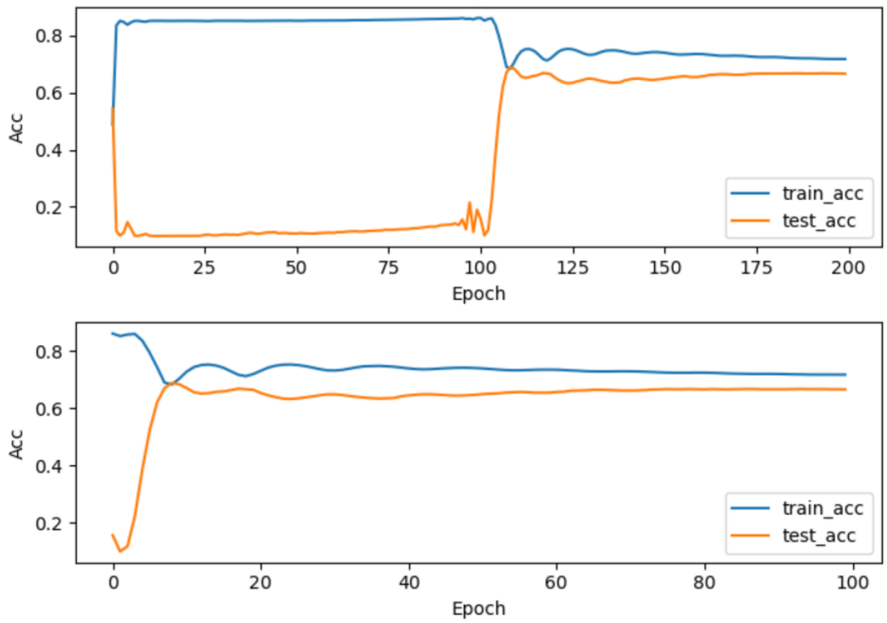


图 11: 使用 IRM 在 ColorMNIST 上的训练结果

五、VREx 算法复现

5.1 算法简介

与 IRM 算法相同，VREx 算法也试图通过减小不同环境之间的训练表现差异来减小对于干扰因果信息的学习、增强对于不变特征的学习。不同于 IRM，VREx 算法的思路是将不同环境之间风险的方差作为惩罚项：

$$\mathcal{R}_{\mathrm{V-REx}}(\theta) \doteq \sum_{e \in \mathcal{E}_{\mathrm{tr}}} \mathcal{R}_e(\theta) + \lambda \cdot \mathrm{Var}\left(\{\mathcal{R}_1(\theta), \ldots, \mathcal{R}_m(\theta)\}\right) \tag{2}$$

其中 $\mathrm{Var}\left(\{\mathcal{R}_1(\theta), \ldots, \mathcal{R}_m(\theta)\}\right)$ 表示的是对于分类器在不同环境下不同表现的惩罚。

5.2 算法复现

我们复现了论文中使用 VREx 算法在 ColorMNIST 数据集上的训练结果，并获得了图12的训练结果。可以发现，最终训练集上的准确率收敛到了 69.45%，测试集上的准确率收敛到了 70.9%。

可以发现，使用 VREx 的表现比使用 IRM 的表现稍微好一些，这或许是因为 IRM 算法假定在所有数据环境上的最优分类器都是一个线性分类器，并且将所有的优化任务都集中到寻找不变特征的数据映射上，然而 Lenet 作为一个较为浅层的神经网络，可能并不具有这么强大的数据表征，因此最后的不变特征不一定都是线性可分的，换而言之，假定最佳分类器的线性性的 IRM 算法因此失去了部分的非线性数据的表征能力。而 VREx 算法使用方差作为惩罚项，虽然简单并且直观，但可能因此更加具有非线性的表征能力，因此表现更好。

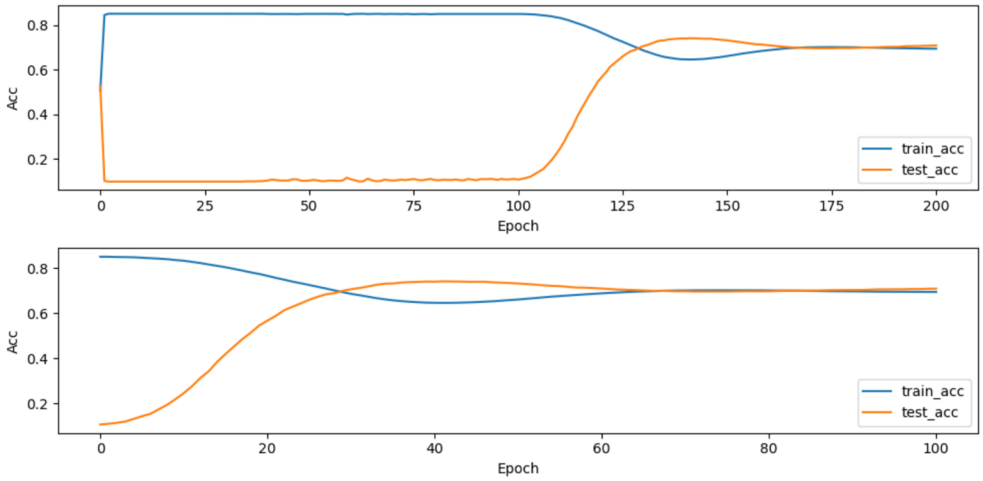


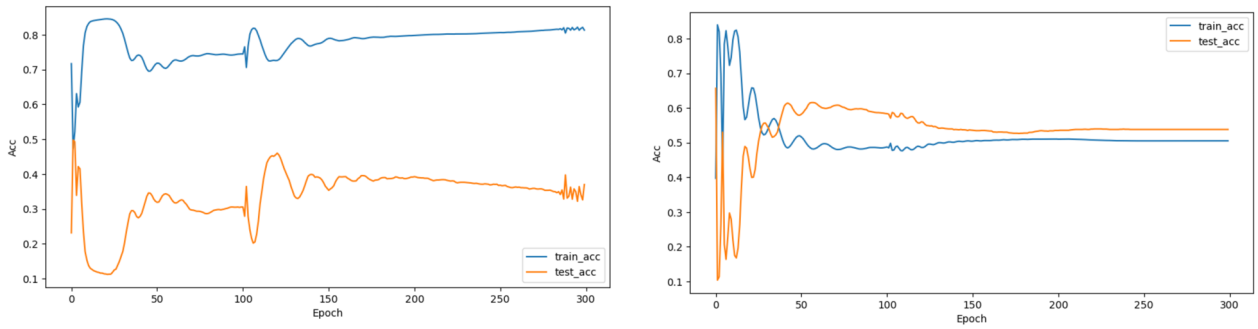
图 12: 使用 VERx 在 ColorMNIST 上的训练结果

六、提升 IRM 算法稳定性

IRM 算法对于 pennalty weight 这一超参数过于敏感，初始化的选择稍有不慎，就会导致算法难以收敛，或者算法收敛时间较长，如何缓解 penalty weight 的敏感性是我们需要解决的问题。

6.1 Penalty Anneal

在 IRM 与 VREx 算法的论文中都使用了 penalty anneal 这一技巧，即让 penalty weight 在前 100 代的训练中保持为 1.0，使得模型能够在这些训练中学习到足够的特征，并在 100 代之后将 penalty weight 突变为预设的值，这是一个非常有效的手段，事实上，本报告之前所有的实验数据中都使用了这一技巧。图13展示了若不使用 penalty anneal 的实验情况，可以发现，当 penalty anneal 预设的较小时模型很难获得足够的不变特征的信息，而当其预设的较大时，模型又会呈现 50% 的瞎猜状态，这是我们所不希望看到的。



(a) weight=5e1　　(b) weight=1e4

图 13: 不采用 Penalty Anneal 时的情况

6.2 Penalty Fluctuation

人为设定的超参除非经过仔细地调整，否则难以获得最佳的效果，倘若能在训练过程中汲取相应的信息，并以此自动调节 penalty weight 的话，训练或许能够获得更好的表现。

基于这种思考，我们提出了 Penalty Fluctuation 用以自动在训练过程中调节 penalty weight 来获得最好的模型表现。这一算法的创造动机如下：在训练过程中，我们不希望看到训练准确率骤增，因为这一般意味着算法学习到了不变特征以外的干扰特征，这会使得模型的泛化能力下降，因此当训练准确率骤增时我们希望 penalty weight 能相应地增加，增强模型对不变特征的学习能

力；同样地，我们也不希望看到训练准确率骤降，以为这一般意味着模型过于注重不同环境之间的泛化，开始往盲猜的方向发展，这时候我们希望 penalty weight 能够相应的下降，增强模型对于数据特征的学习。

因此，我们在原有算法的基础上增加了以下的 Penalty Fluctuation 部分用以自动调节 penalty weight:

$$\beta = \frac{\text{LastTrainAcc}}{\text{TrainAcc}} \tag{3}$$

$$\hat{\lambda} = \lambda_{\text{initial}} \cdot 10000^{-tanh(\beta^2-1)} \tag{4}$$

$$\lambda = \alpha\lambda + (1-\alpha)\hat{\lambda} \tag{5}$$

其中式(3)衡量了前后 epoch 之间训练准确率的变化率。式(4)定义了当前 epoch 的的惩罚权重，通过非线性的放缩以及指数的乘积，使得当前权重 $\hat{\lambda}$ 能够在初始预设超参 λ_{initial} 与十的正负 4 之间的乘积的区间内浮动。式(5)利用了指数移动加权平均，使得 penalty weight 具有了跨 epoch 的记忆性，使权重的变化更加平稳，算法更加稳定。

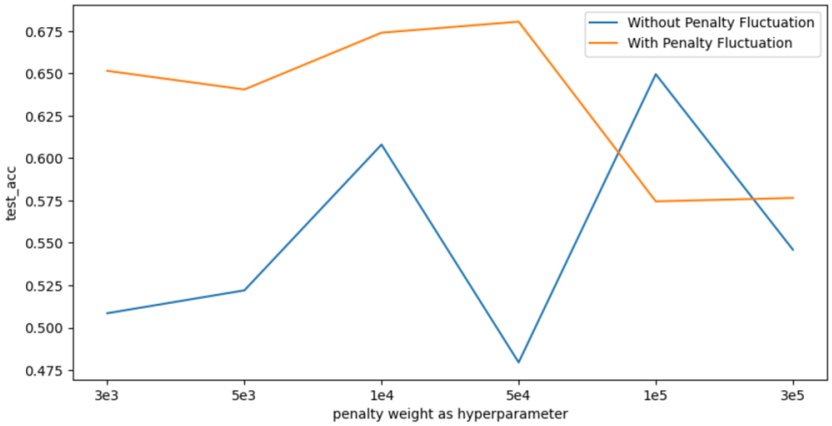


图 14: 增加 Penalty Fluctuation 后的训练结果

我们用使用了 Penalty Fluctuation 的 IRM 算法与没有使用 Penalty Fluctuation 的 IRM 算法在 penalty weight 为 $3e3, 5e3, 1e4, 5e4, 1e5, 3e5$ 时进行训练并记录他们最终收敛后的测试集预测准确率，结果如图14所示。可以看到，在十的三次方到十的五次方这一极大的范围内，前者都获得了不俗的表现。在后者的预测准确率仅仅比 50% 高一点时，前者能获得 65% 的准确率。即使是在后者表现出色时，前者的表现能比其更加出色。因此，我们可以发现，Penalty Fluctuation 的引入降低了 penalty weight 的敏感性，使得算法咋一个极大的权重范围内都能获得不俗的表现。

七、总结

在本次大作业的实践中，我了解了 OOD 问题的起因与重要性，尝试了各种可能缓解 OOD 问题的数据增广措施，了解了 IRM 与 VREx 算法的动机与精妙的数学推导，复现了它们在 ColorMNIST 数据集上的表现，并针对性地对 IRM 算法的 penalty weight 过于敏感这一问题作出了创新，提出了自己的解决方案与算法，可谓是收获颇丰!

八、附录

本次实践所涉及到的所有具体代码请查看：
https://github.com/bydwqq/AI1603_Exploration_on_OOD_Generalization