人工智能编程实践小组大作业汇报

小组成员：杜亦开 李嘉睿 高原

2023年7月22日

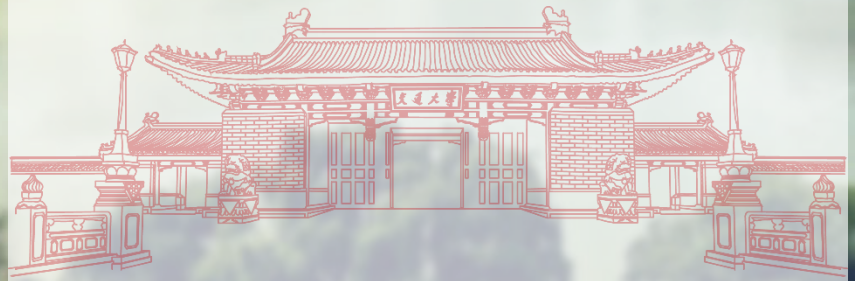
饮 水 思 源 • 爱 国 荣 校

目 录

01

OOD问题

Out of Distribution



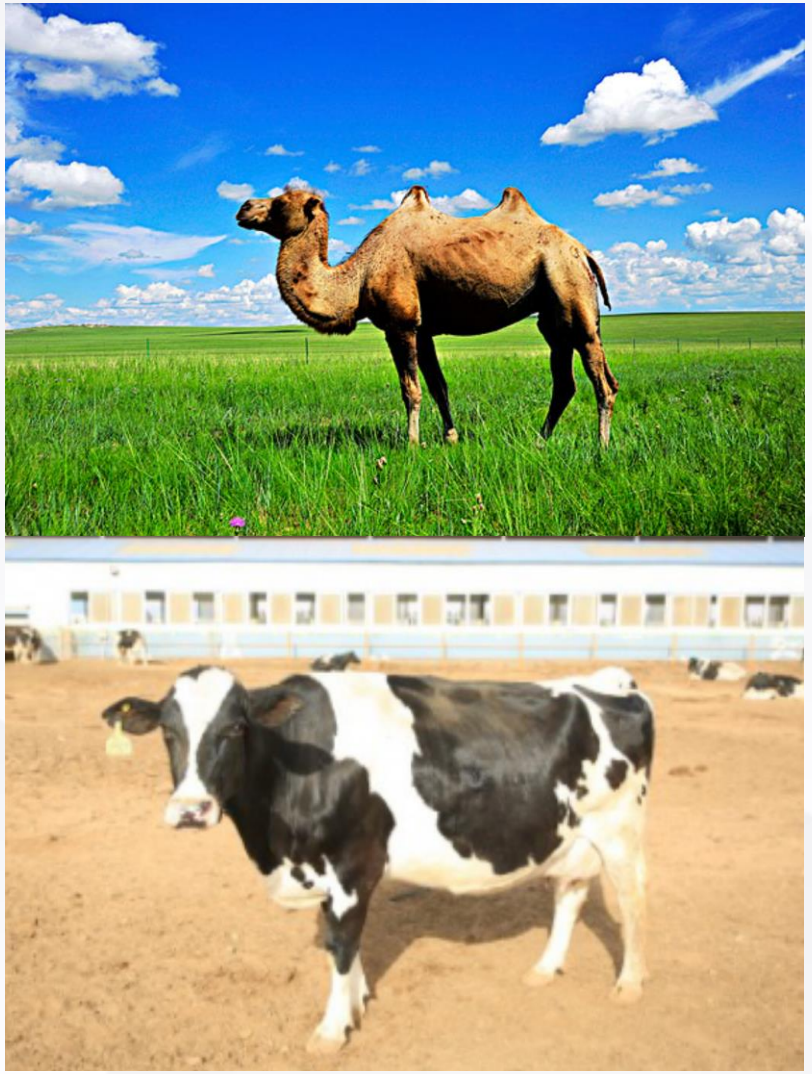
OOD问题

OOD：Out-Of-Distribution

神经网络通常以独立同分布的假设进行训练，即假设测试数据分布与训练数据分布是相似的。然而，实际情况下，这一假设并不成立，会导致训练准确度高的模型在分布域外进行预测时表现显著下降。而常理来说，我们无法将所有情况的数据给予深度神经网络进行训练，因此对人工智能系统提出了分布外泛化的要求。

OOD问题的

一个经典的问题是分类不同环境的动物。
OOD问题将样本中的特征分为两类，一类是spurious feature，例如图中的背景，另一类是我们希望的学到的feature，比如动物形状，即invariant feature。
传统的模型直接学习到的spurious feature，导致在如右图所示的情况下判断失误。
而解决OOD问题的目标就 是令模型更多地学习到invariant feature，从而提高泛化能力。



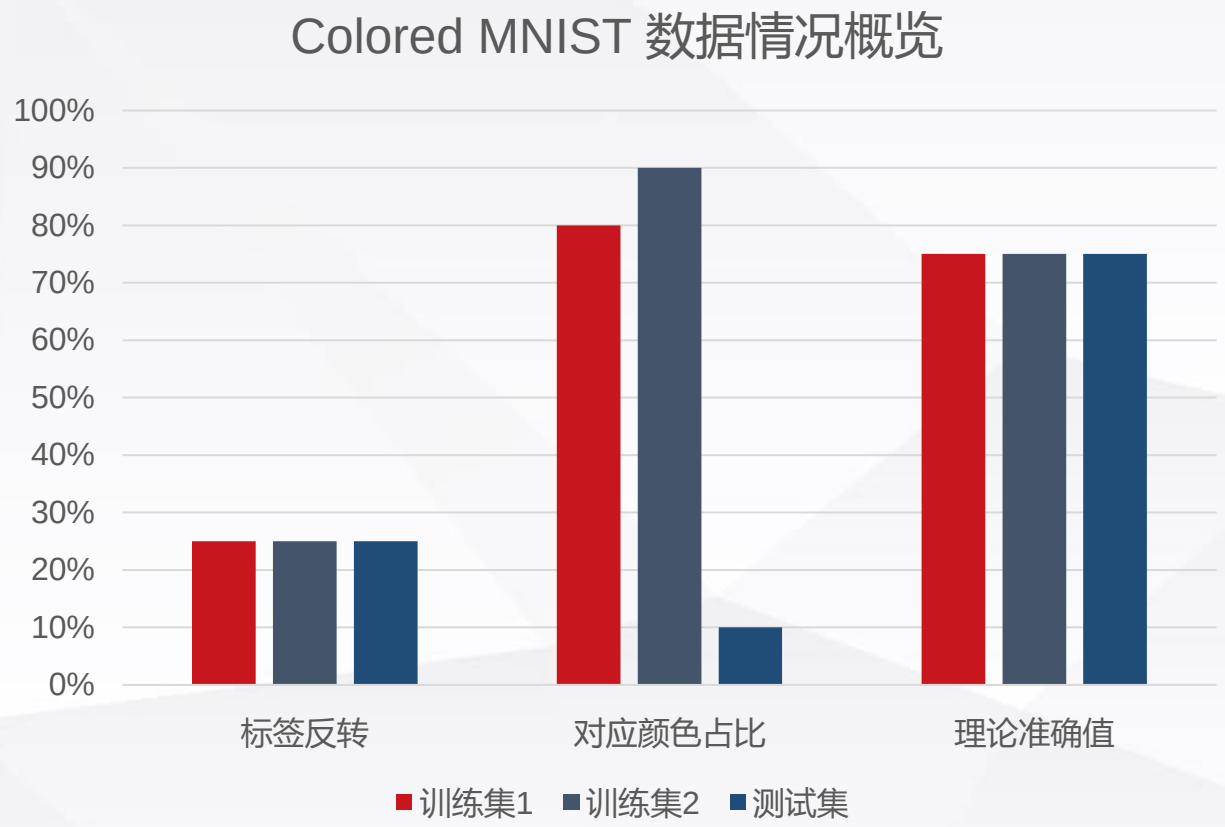
02

Colored MNIST 简述

机器学习的”Hello, world”

Colored MNIST 数据集的结构

- Colored MNIST 由60000张手写数字图片组成，根据数字类型（0~4 或 5~9）为其赋予红色和绿色的标签，进行25%的随机反转打乱后分别以20%，10%，90%的比例反转图片颜色生成训练集1、2和测试集。



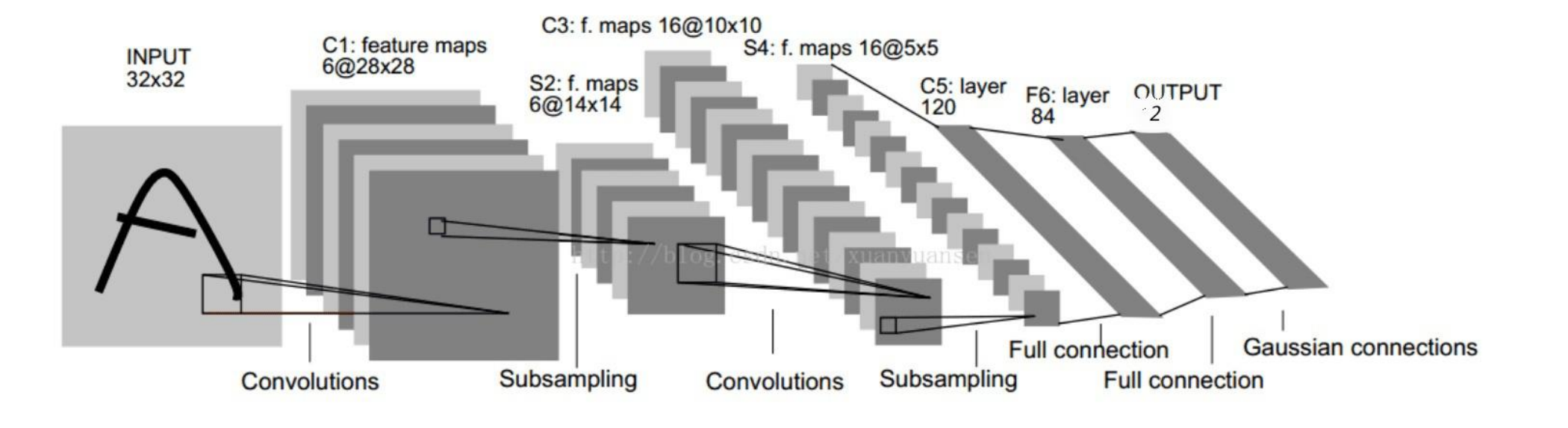
预测目标是指出数字类型

理论准确值最高为75%

03

在CMNIST上训练LeNet

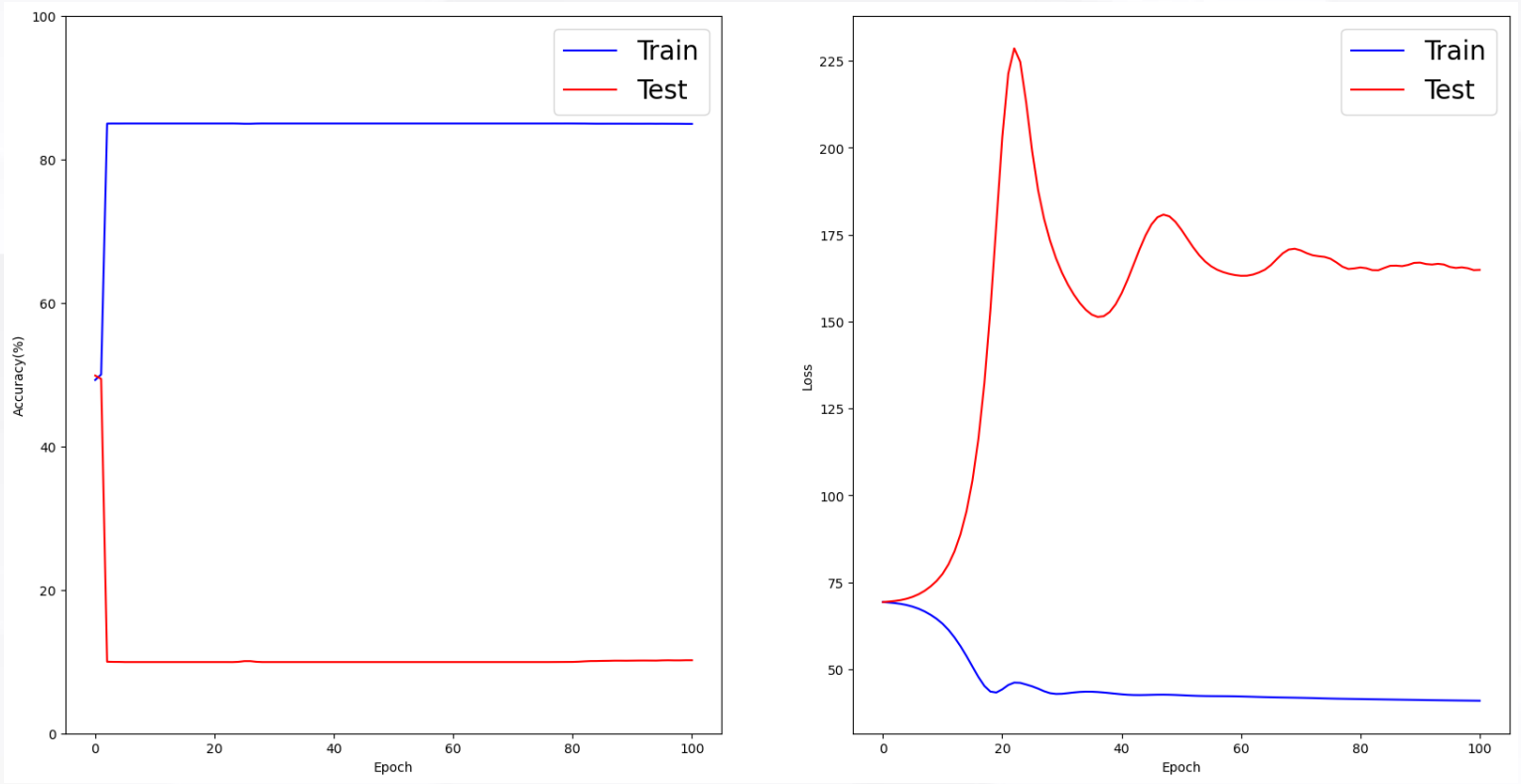
LeNet结构



LeNet 神经网络包含三个卷积层，两个池化层和两个全连接层

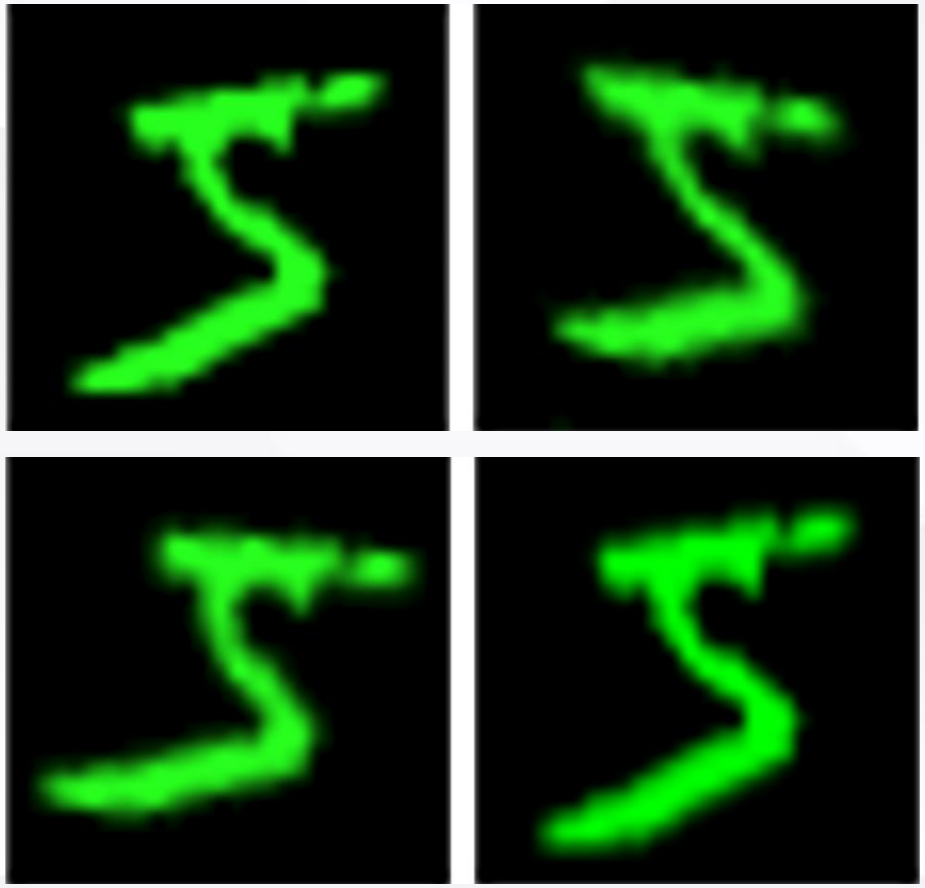
(在实现基于 LeNet 的CMNIST训练用模型时，模型的输入输出形状与示意图中略有不同)

LeNet直接训练结果



结果不出意外，LeNet表现极差，表现出训练集准确率高，测试集准确率极低的特点
可见该模型学到了大量spurious feature

数据增广



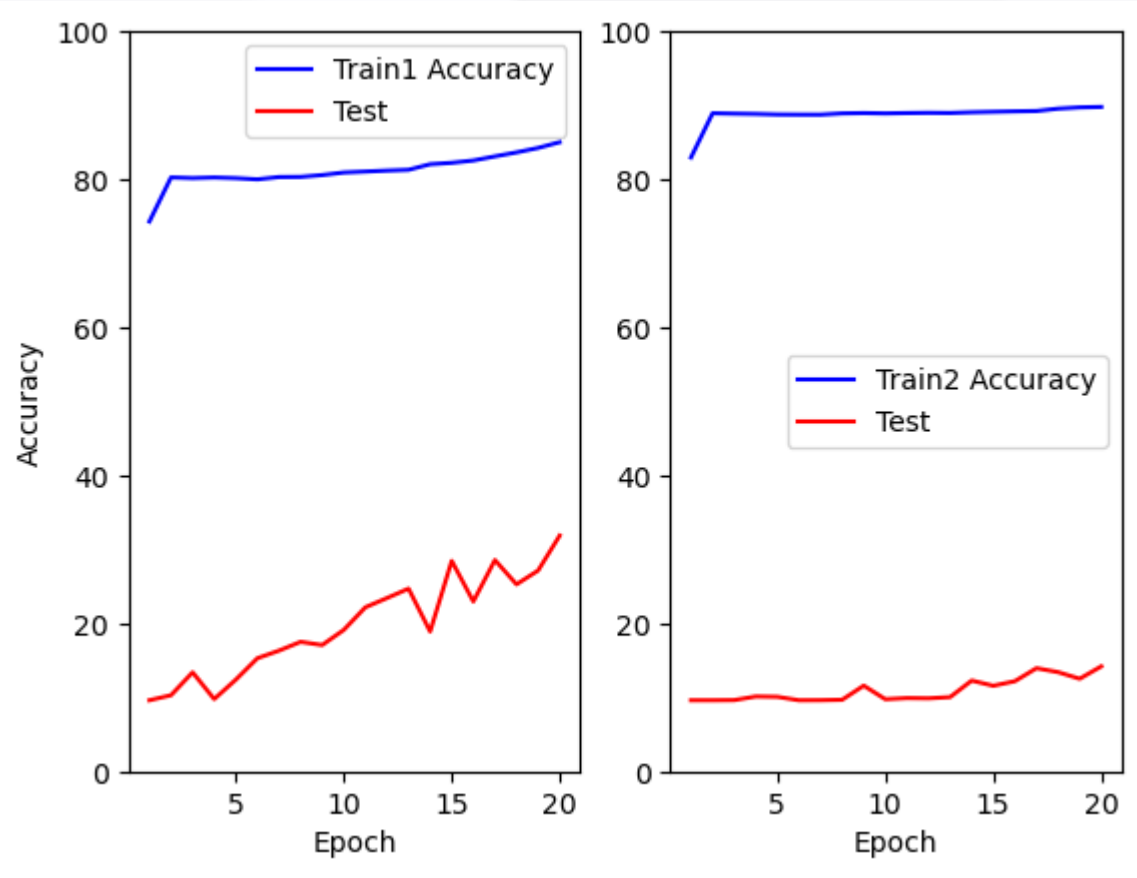
但是LeNet真的无可救药了吗?

尝试使用数据增广（Data Augment）

数据增广在图片处理方面有旋转、翻转、比例缩放、裁剪、位移、添加噪声等方法

本次实验采用旋转、剪切、缩放三种操作用于数据增广训练LeNet

数据增广



由于数据增广后训练负载较大，故减少训练轮数

数据增广的训练结果如左图所示

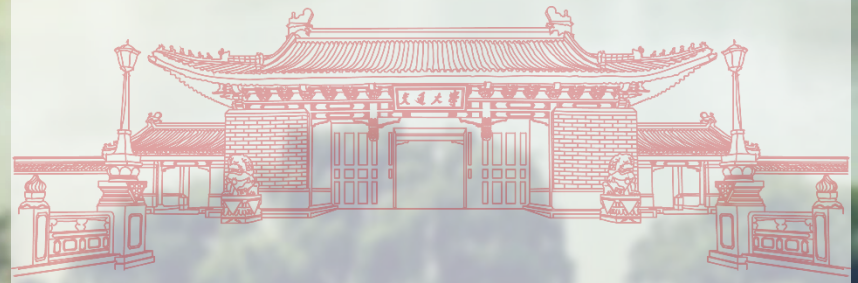
虽然显然还是无法胜任OOD任务，但是准确率获得了略微进步

04

IRM

Invariant Risk Minimization

复现与尝试优化



IRM: Invariant Risk Minimization

Invariant Risk Minimization

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, David Lopez-Paz

1 Introduction

Machine learning suffers from a fundamental problem. While machines are able to learn complex prediction rules by minimizing their training error, data are often marred by selection biases, confounding factors, and other peculiarities [49, 48, 23]. As such, machines justifiably inherit these data biases. This limitation plays an essential role in the situations where machine learning fails to fulfill the promises of artificial intelligence. More specifically, minimizing training error leads machines into recklessly absorbing all the correlations found in training data. Understanding which patterns are useful has been previously studied as a correlation-versus-causation dilemma, since spurious correlations stemming from data biases are unrelated to the causal explanation of interest [31, 27, 35, 52]. Following this line, we leverage tools from causation to develop the mathematics of spurious and invariant correlations, in order to alleviate the excessive reliance of machine learning systems on data biases, allowing them to generalize to new test distributions.

As a thought experiment, consider the problem of classifying images of cows and camels [4]. To address this task, we label images of both types of animals. Due to a selection bias, most pictures of cows are taken in green pastures, while most pictures of camels happen to be in deserts. After training a convolutional neural network on this dataset, we observe that the model fails to classify easy examples of images of cows when they are taken on sandy beaches. Bewildered, we later realize that our neural network successfully minimized its training error using a simple cheat: classify green landscapes as cows, and beige landscapes as camels.

2019年由Martin Arjovsky等人提出的Invariant Risk Minimization方法为OOD问题的攻关提出了新的方向。

IRM的推导

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

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

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

3. 写出w的最小二乘解，并将逆矩阵消掉

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

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

IRM的推导

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

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

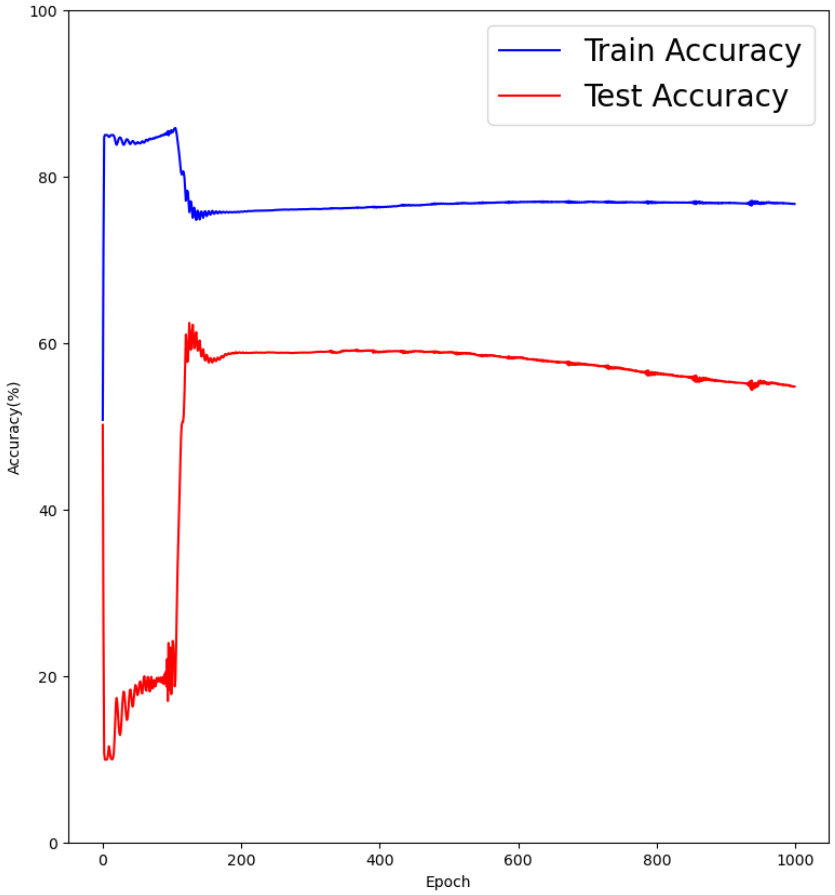
值得说明的一点是，这里的风险函数R默认为均方误差损失函数，但其实当用于分类问题时，R也可以是交叉熵函数等等。而且我们可以发现，最后得出的损失函数的两项有着较为相似的形式，它们都是从经典损失函数$R(w \circ \Phi)$得出来的，可以用python程序去较好的模拟。

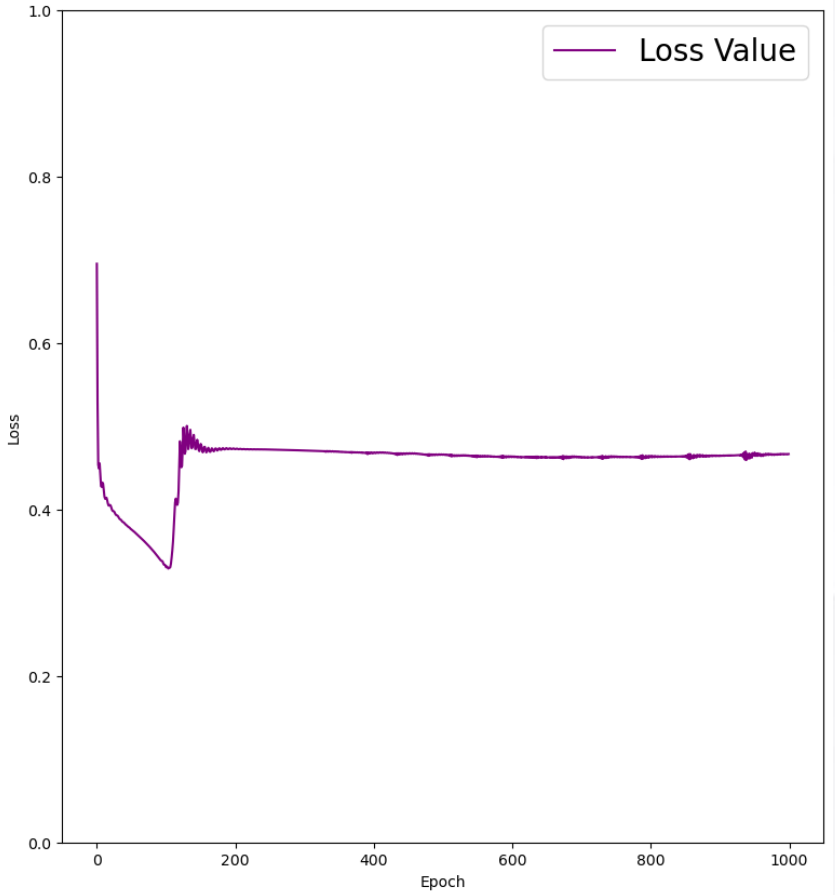
至此我完成了公式的部分推导。实际上原文的目标损失函数的意义我还不是很理解，但是从它出发，可以推出IRM算法的核心数学公式。在整个推导过程中，我认为步骤5是最为精妙的，它地将距离D用损失函数的偏导数表示，从而统一了损失函数中看似完全不同的两项，并使其易于实现，真是巧夺天工！

IRM的基础实现

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

$$\lambda = \begin{cases} 1.0 & if\ iteration < anneal\ limit \\ penalty\ weight & other \end{cases}$$





根据论文及其附录示例代码复现的IRM表现如左图所示，可见其准确率相比LeNet有了质的飞跃。

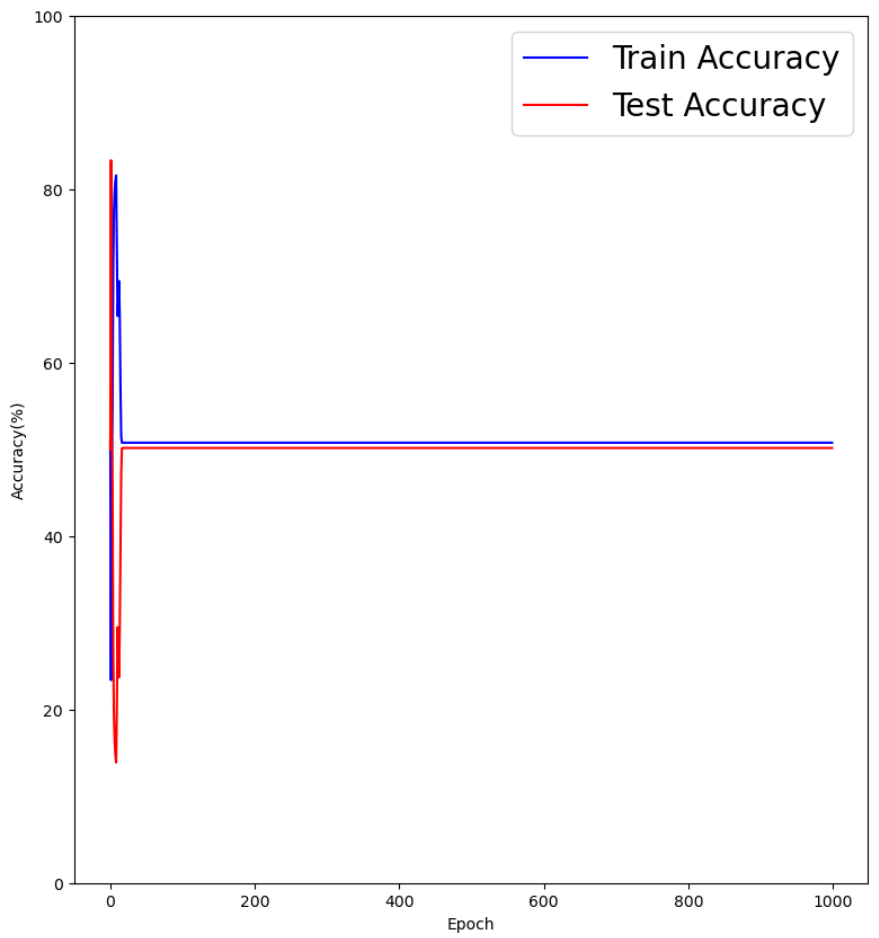
但是一个问题显现出来：测试集的准确率并没有达到稳定，而是逐渐下降，有退化的趋势。

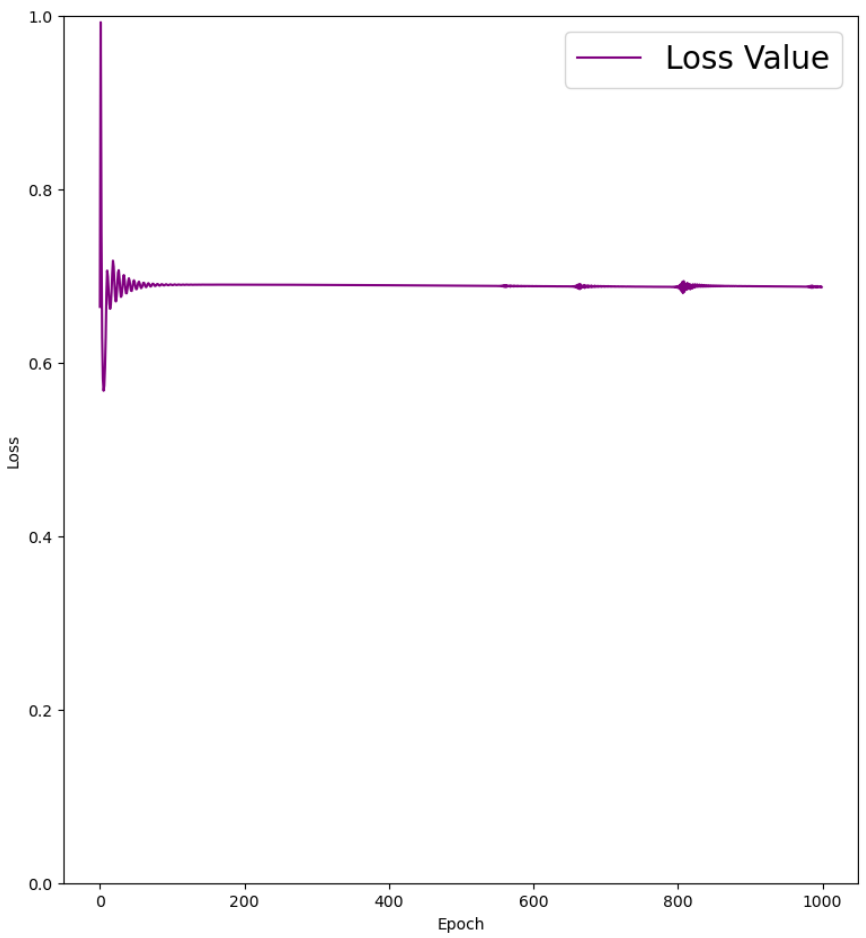
Penalty weight的优化尝试

Penalty weight，即λ，是调节普通模型的损失项与IRM算法惩罚项penalty的关系的重要超参数

参考论文原文文末的代码，最初我们将超参数设为等价的10^5，但是获得了右图所示的令人失望的结果

Loss = ln(0.5) = 0.693147





Penalty weight的优化尝试

而后经过多次测试，总结出如下规律：

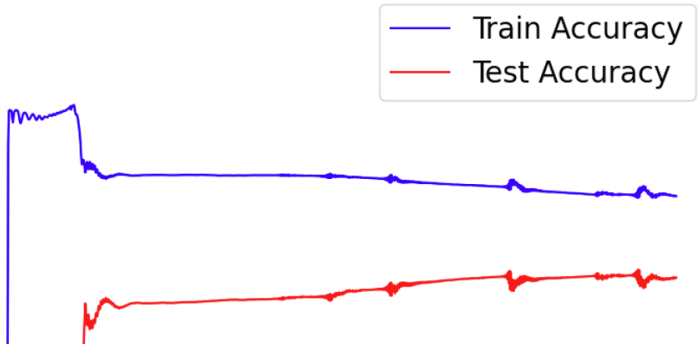
·当λ过小时，呈现训练准确率高，测试准确率极低的特点，这与LeNet的训练结果较为相似。并且，存在过拟合现象，测试集的训练准确率高于理论准确率。

·当λ过大时，整体表现趋于随机猜测。

这两点特征可以直观理解。λ过小时，模型主要对预测损失进行优化，IRM特有的惩罚项penalty在总损失中占比小，因此IRM起到的的作用小，训练结果更多地受到普通损失的影响，趋向于LeNet的训练结果也是合情合理。而当λ过大时， 损失权重过小，模型无法进行最基本的收敛。

在后续的尝试中，一种调节λ的方式表现出了一定效果：

$$\lambda = \begin{cases} 1.0 & if\ iteration < anneal\ limit_1 \\ 10000.0 & elif iteration < anneal\ limit_2 \\ 1000000.0 & else \end{cases}$$

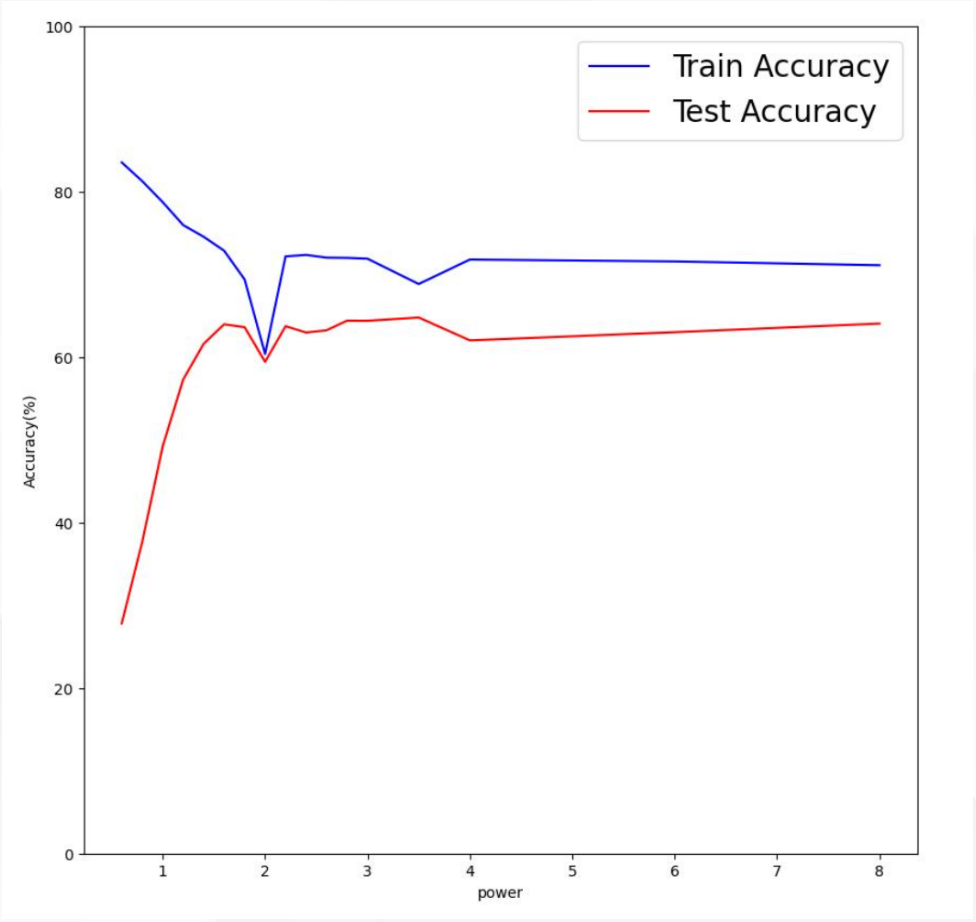


Penalty weight的最终优化

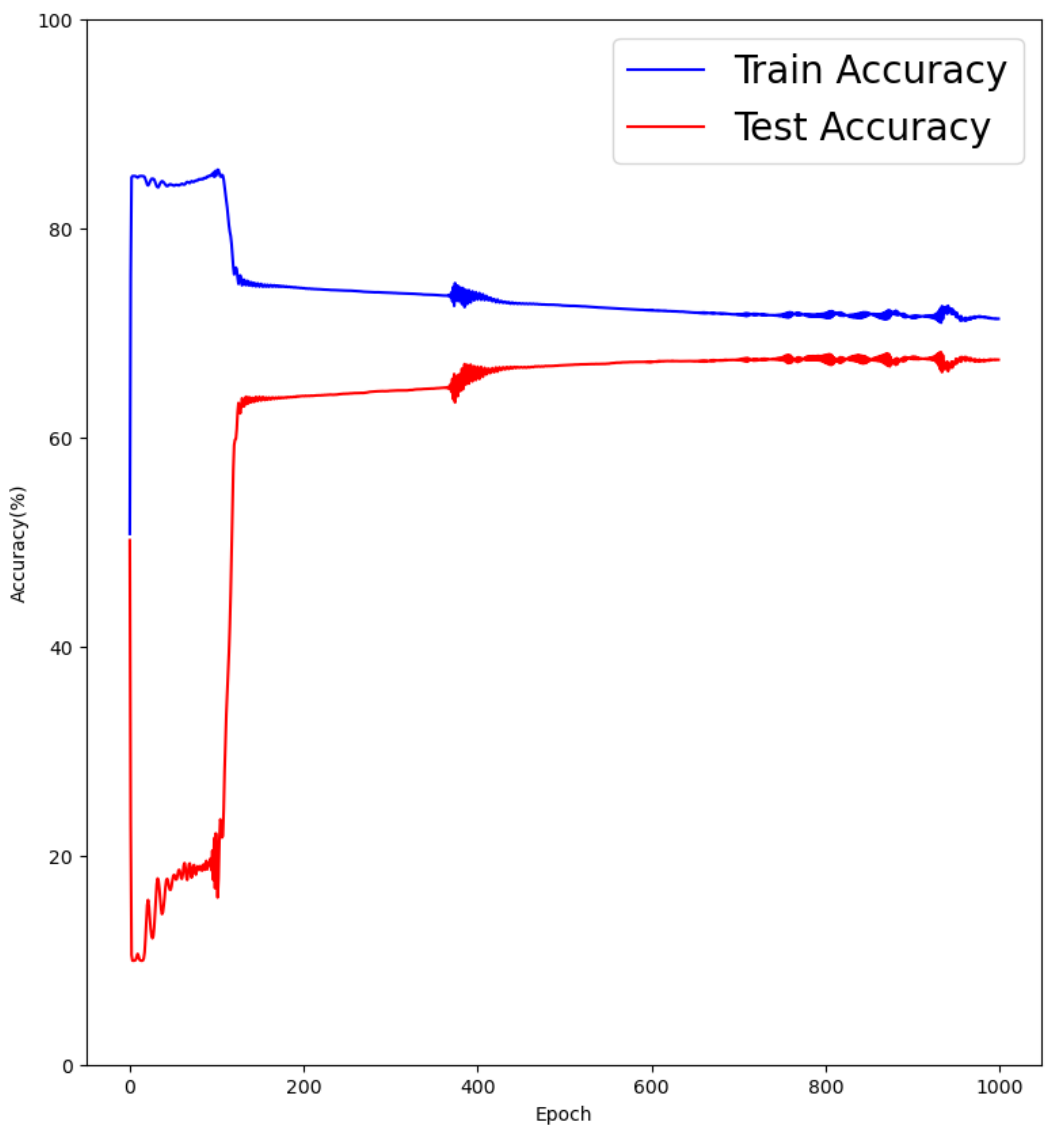
若希望模型的准确率进一步提高，随着epoch增加不断地提高λ的大小似乎是有效的。
经过包括线性提高λ等方法在内的各种尝试，多次试错后发现以高次多项式速度提高λ可以达到较好的效果（若为指数速度增长则会在很快产生溢出），即：

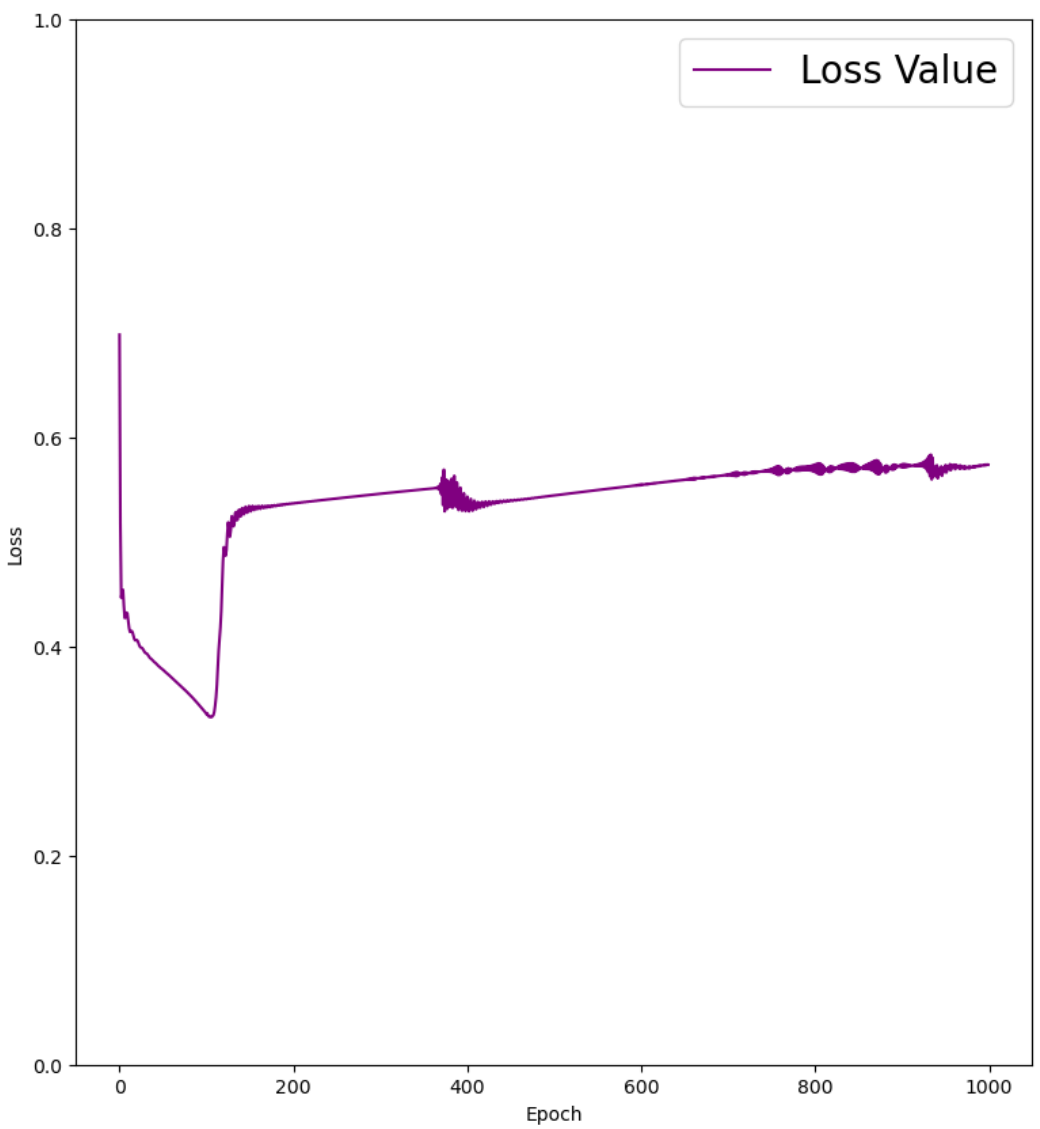
$$\lambda = \begin{cases} 1.0 & if\ iteration < anneal\ limit_1 \\ k \cdot iteration^p & else \end{cases}$$

右图为k=1.0，epoch=500的情况下次幂与训练集和测试集最终收敛准确率的关系。
该尝试仅为数值估计，但是在模型收敛上表现较为良好。



Penalty weight的最终优化





05

AndMask

AndMask公式

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

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

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

因此当A、B的形状十分相似时，不确定性就很小。

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$H^{\wedge} = diag((\prod_{e \in \varepsilon} \lambda_1^e)^{1/|\epsilon|}, \ldots, (\prod_{e \in \varepsilon} \lambda_n^e)^{1/|\epsilon|})$。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：

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

AndMask简化

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。

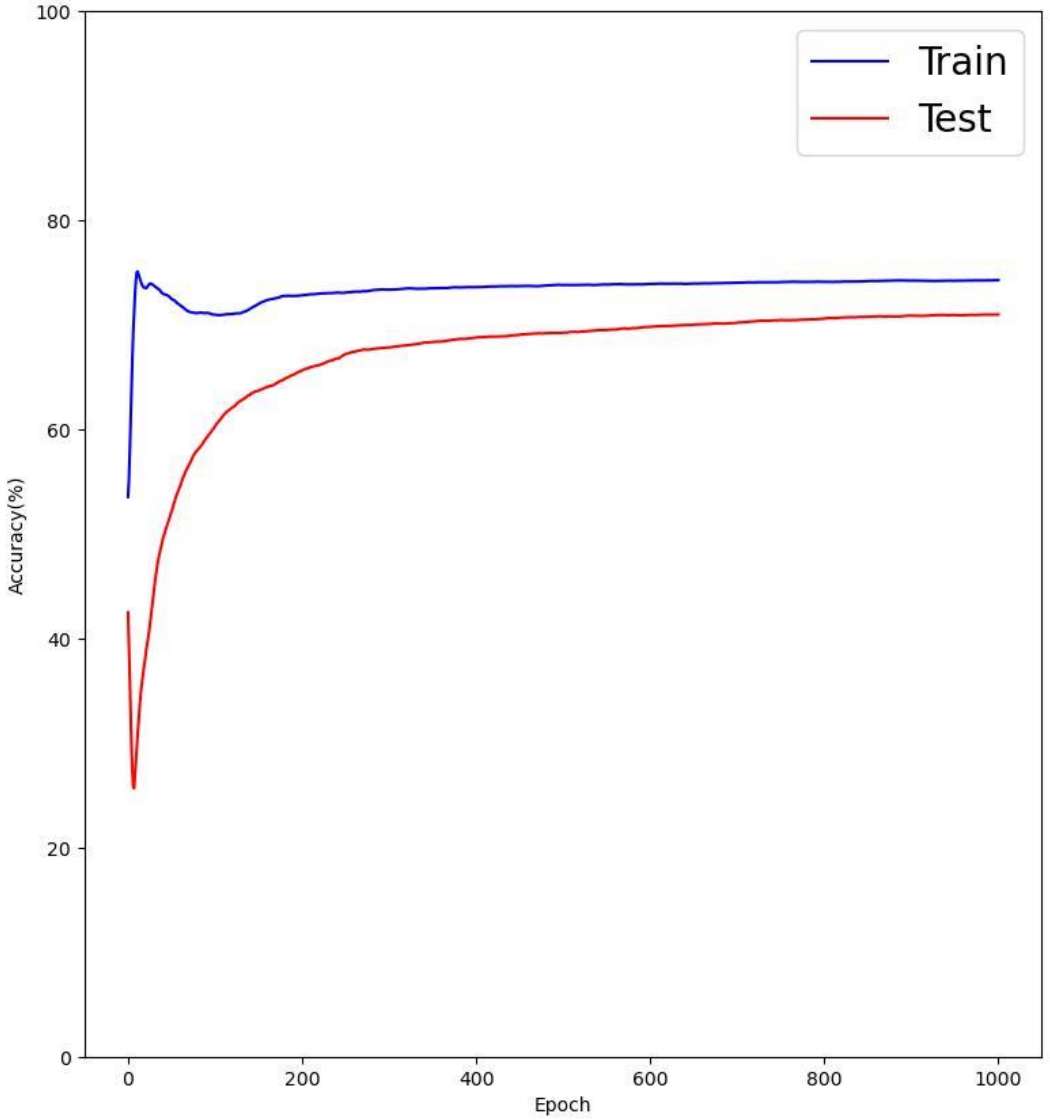
2. 在本题中，不同环境指的是每张图片不同的R、G、B三个图层，所以每个网络参数的位置，都会有三个参数，分别代表三个图层的参数值，从而它们的梯度矩阵也存在三个值，分别代表三个图层参数的梯度。要实现这个，首先要计算出参数梯度的符号矩阵（S）。对每个梯度值正则设为1，负则设为-1，0仍然设为0。如果某个位置三个符号值之和的平均值的绝对值小于一个超参数tau，我们就认定这个参数不够稳定，它对应的mask矩阵中的值应该是0。

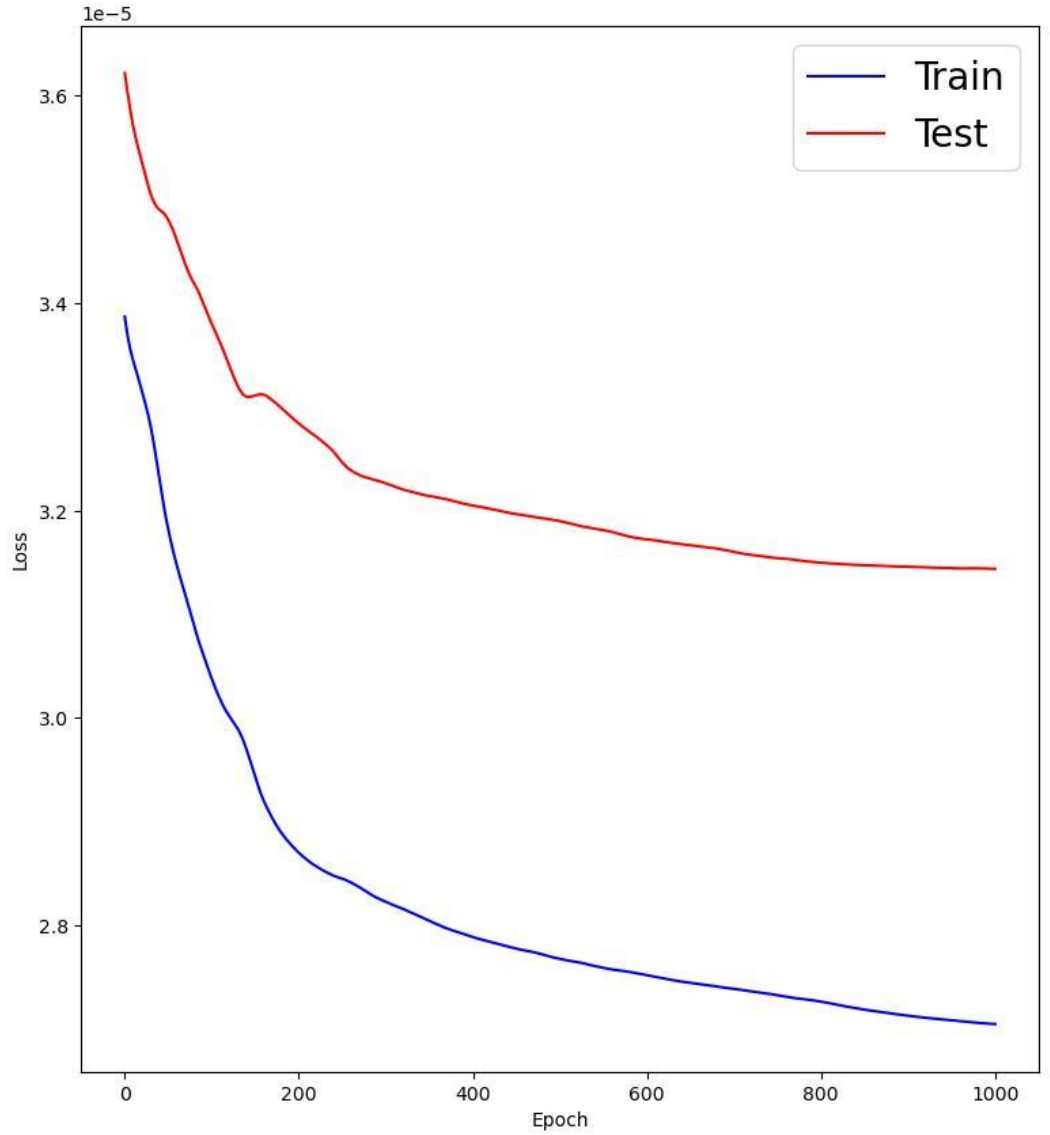
3. 在这里，tau设定为$(0,1)$范围内的随机数，而且可以看出S矩阵中的每个元素都应该是$0, \pm 1, \pm\frac{2}{3}, \pm\frac{1}{3}$中的一个。下面是个计算的例子（$M_1$、$M_2$、$M_3$表示同一位置不同图层的梯度）：

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

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

AndMask在CMNIST上的效果





06

V-REx

V-REx

- V-REx算法通过减少训练领域之间的风险差异来降低模型对各种极端分布变化的敏感性，包括输入包含因果与反因果元素的具有挑战性的环境。而V-REx作为REx的变体，提出对训练风险方差给予惩罚以此来达到REx算法的理论目的。这种算法可以恢复目标的因果机制，同时对输入分布的变化（协变量偏移）也有一定的鲁棒性（robustness）。

  由论文中可得知loss函数为 $R_{V-REx}(\theta) \doteq \beta Var(\{R_1(\theta), \ldots, R_m(\theta)\}) + \sum_{e=1}^{m} R_e(\theta)$

- 其中β控制着降低平均风险和强制风险平等之间的平衡，当它为0时算法将恢复为ERM，当它趋于无穷时则将完全专注于使风险平等。

V-REx

网络设置

与IRM算法类似，我们这里选择使用MLP，理由也是一样的，这里不再赘述。有一点点小的变动是，为了使模型更高效的运转，我们选择了压缩图片为14*14大小，网络代码如右图所示

```
class MLP(nn.Module):
    def __init__(self):
      super(MLP, self).__init__()
      lin1 = nn.Linear(2 * 14 * 14, 256)
      lin2 = nn.Linear(256, 256)
      lin3 = nn.Linear(256, 1)
      for lin in [lin1, lin2, lin3]:
        nn.init.xavier_uniform_(lin.weight)
        nn.init.zeros_(lin.bias)
      self.model = nn.Sequential(lin1,
                                 nn.ReLU(True),
                                 lin2,
                                 nn.ReLU(True),
                                 lin3)
    def forward(self, input):
      out = input.view(input.shape[0], 2 * 14 *
14)   out = self.model(out)
      return out
```

V-REx

损失函数

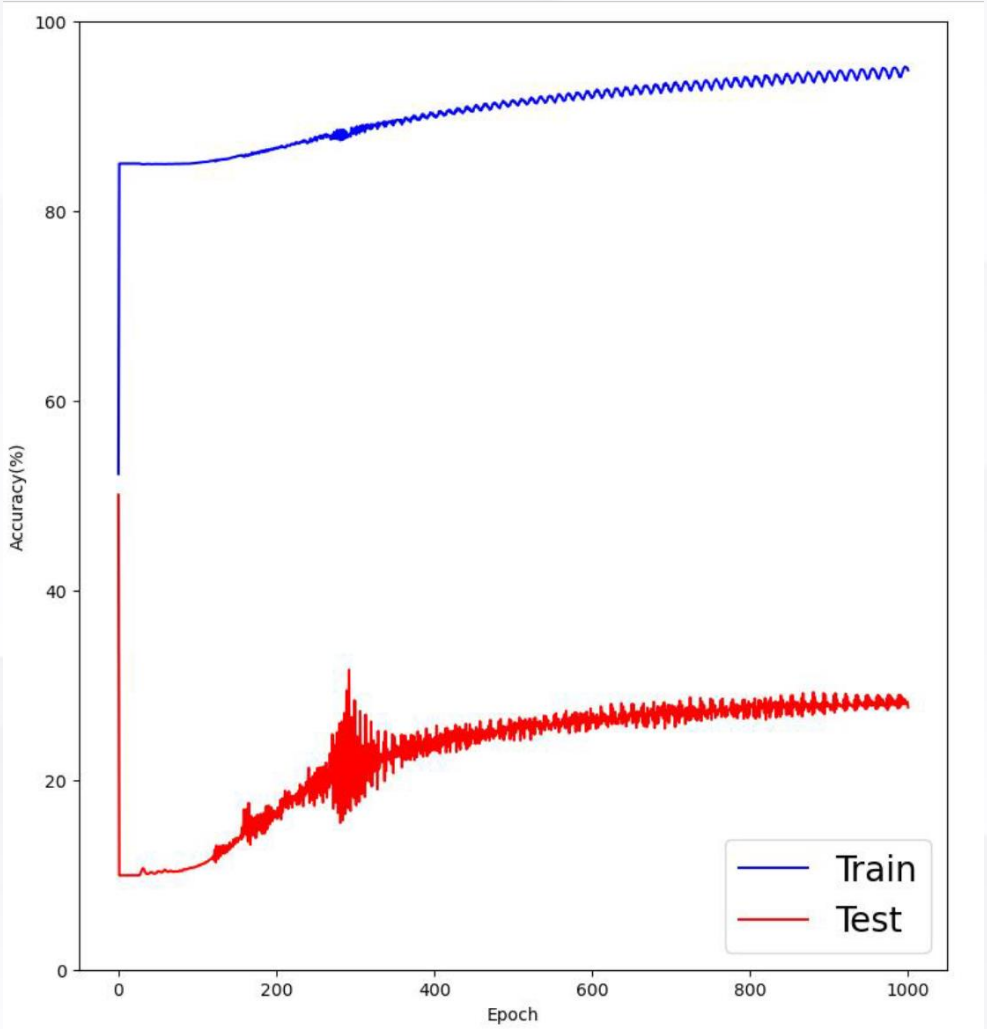
借鉴了官方的Github中对损失函数的定义：

```
# Define loss function helpers
  # logits : output of final fc
  def mean_nll(logits, y):
    return nn.functional.binary_cross_entropy_with_logits(logits, y)

  def mean_accuracy(logits, y):
    preds = (logits > 0.).float()
    return ((preds - y).abs() < 1e-2).float().mean()

  def penalty(logits, y):
    scale = torch.tensor(1.).cuda().requires_grad_()
    loss = mean_nll(logits * scale, y)
    grad = autograd.grad(loss, [scale], create_graph=True)[0]
    return torch.sum(grad**2)
```

V-REx

运行结果

可以看到在epoch = 800后，测试集准确率逐渐收敛至0.3，比普通的使用LeNet效果要好。



References

·Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David LopezPaz. Invariant Risk Minimization. https://arxiv.org/abs/1907.02893

·卷积神经网络之Lenet https://zhuanlan.zhihu.com/p/116181964

·Invariant Risk Minimization系列阅读笔记 https://zhuanlan.zhihu.com/p/273209891

·Invariant Risk Minimization (IRM)的前世，今生，和未来探讨 https://zhuanlan.zhihu.com/p/567666715

·HappyWang. Machine Learning(4) Multilayer Perceptron(MLP), 2020

·Giambattista Parascandolo, Alexander Neitz, Antonio Orvieto, Luigi Gresele，Bernhard Schölkopf. Learning explanations that are hard to vary. https://arxiv.org/abs/2009.00329

·lopezpaz. DomainBed. https://github.com/facebookresearch/DomainBed/blob/main/domainbed/algorithms.py

·IRM Algorithm from Facebook Research github repository https://github.com/facebookresearch/InvariantRiskMinimization/blob/main/code/colored_mnist/main.py

·David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Remi Le Priol, Aaron Courville Out-of-Distribution Generalization via Risk Extrapolation (REx) https://arxiv.org/abs/2003.00688

·REx official github repository https://github.com/capybaralet/REx_code_release

成员	参与工作
杜亦开	实现LeNet，数据增广，复现IRM与AndMask代码，IRM参数调节，PPT制作
李嘉睿	实现LeNet，推导IRM与AndMask的有关数学证明
高原	实现LeNet，复现V-ERx

上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

感谢聆听

饮 水 思 源　爱 国 荣 校