对分布外泛化(OOD)问题的理解及相应对策的探索

——课程大作业报告

一、对 OOD 问题的理解

长久以来,分布外泛化问题已经成为人工智能领域一大热点问题。所谓分布外泛化问题,就是指当测试样本的分布与训练样本的分布不同时,如何保证所学模型的泛化能力。举个经典的例子，就是在训练数据集中有奶牛和骆驼，而我们要训练的模型要实现对奶牛和骆驼的分类，但是训练集和测试集样本的分布不同：训练集中，大部分奶牛在草地的背景中，而大部分骆驼则在沙漠的环境中；而在测试集中，出现了更多在沙漠背景的奶牛和在草地背景的骆驼。在这种情况下，经过训练集训练后的模型很容易将背景与对目标的预测产生关联，即在很大程度上利用环境来直接对目标进行分类：将测试集中草地背景上的目标分为奶牛，将沙漠背景下的目标分为骆驼，而非直接关注目标本身的特性。在这个例子中，背景就是 spurious feature(Xs)，这类特征与目标的分类没有直接关系，但可能在训练中产生虚假的因果关系(spurious correlation),而动物的特征(颜色、性状等)就是 causal feature(Xc)，这类特征直接影响的目标的分类，在这基础上则出现了因果学习(causal learning)。

因果学习的目标是希望解决如何从数据中发现因果模型、因果关系、因果特征等，如果模型能够从数据中发现其中的 causal feature 并且减小 spurious feature 的影响，那么分类的准确性必将大大提高。在这里，把 causal feature 看作因，只有它可以影响果(预测结果)，所以，可以把这种影响关系表示为某种单向赋值函数 f(Y|Xc)，Y 为预测目标的标签，这种关系不受 spurious feature 等非因的影响，只和 causal feature 有关。这类关系也是稳定的，目标的标签总能很好的吻合目标的 causal feature，所以 causal feature 也叫做 invariant feature，而 spurious feature 和目标标签的关系则是不稳定的，例如同一只骆驼分别在草地和沙漠，目标本身骆驼的标签不变，而 spurious feature 却有了巨大的差异。利用这个特性，便衍生出了包含 IRM 在内的许多判断 casual feature 和解决 OOD 问题的策略。

IRM 希望的是学习一个只和 Xc 有关的特征表达(feature representation) Φ，即把图片的特征变化为与 Xc 有关的特征，如 Φ(X)，它仅仅依赖于 Xc。此时，由于 $P^e(Y|\Phi(X))=P(Y|\Phi(X))$，e 是指任意两个环境，P(Y|Φ(X))是指 Y 和 Φ(X)的相关性，便一定可以找到一个对 P(Y|Φ(X))最优的分类器参数 ω，使得它对所有的环境同时达到最优，所以就有了 IRM 中的公式:

$$\min_{\omega,\Phi} = \sum_{e} \mathcal{R}^e(\omega, \mathcal{R})$$
$$s.t. \omega \in arg\,min_{w^e} \mathcal{R}^e(\omega^e, \mathcal{R})$$

这个公式的意思是希望找到一组最优的参数ω和Φ，使得在所有环境 e 下的风险函数总和最小。同时，还必须满足参数ω在每个环境下的风险函数最小化的约束条件。换句话说，就是学习一个参数Φ，使其对所有的分类器参数ω都同时是最优的。

由于这个优化形式是双层优化，所以又出现了很多其他的变形，包括 IRMv1:

$$\underset{\theta}{\text{minimize}} \quad \sum_{e\in\mathcal{E}_{\text{tr}}} [\ell^{(e)}(\boldsymbol{\theta}) + \gamma \|\nabla_{w|w=1.0} \ell^{(e)}(w \circ \boldsymbol{\theta})\|_2^2], \qquad \text{(IRMv1)}$$

REx:

$$\min_{\omega,\Phi} \sum^{e} \mathcal{R}^e(\omega, \Phi) + \lambda \text{Var}(\mathcal{R}^e(\omega, \Phi))$$

和 InvRat:

$$\min_{\omega,\Phi} \max_{\omega^1 \ldots \omega^e} \sum^{e} \mathcal{R}^e(\omega, \Phi) + \lambda(\mathcal{R}^e(\omega, \Phi) - \mathcal{R}^e(\omega^e, \Phi))$$

等。在这里，IRMv1 利用分类器参数ω在所有环境下最优时反向求 loss 对它的导数为 0 的特性，施加一惩罚使其不断向 0 逼近。利用这点，可以很好的在 python 中实现 IRM 的算法。REx 则利用在提取 causal feature 时，$P^e(Y|\Phi(X))=P(Y|\Phi(X))$，此时各个 loss 的差异很小，利用各个 loss 的方差不断对其进行优化。InvRat 则是求出在某个环境 e^x中的ω^x，然后对比ω^x和ω的 loss 的差别，因为当ω最优时，各个 loss 的差异很小。

回到此次大作业，本次大作业使用了 Colored MNIST 数据集，在这里，数字的性状为 causal feature，而数字的颜色为 spurious feature。训练集中数字的颜色和它的性状有很强的相关性，而在测试集中，其相关性则显著降低，这种数据分布也被称为相关性分布(Correlation shift)，所谓相关性分布，就是指数据的特征同时出现在训练和测试环境中，但是这些特征在不同环境下所体现的统计信息是不同的。模型在训练过程中容易把数字的颜色和数字的标签加上一个虚假的因果关系导致出现 OOD 的情况，所以需要 IRM 等其他算法和各种优化的介入来有效解决这类 OOD 的问题。

二、数据预处理及训练测试 LeNet 模型

本次大作业的初级部分：数据预处理要求在 Colored MNIST 上训练和测试 LeNet 和在数据读取过程中增加数据预处理的方式(数据增广等)，提高 OOD 泛化能力。

2.1、无预处理的测试和训练

首先，在没有数据预处理的情况下训练和测试 LeNet 网络。在一开始对给定 Colored

MNIST 数据集进行处理时，发现类中 makedir_exist_ok 函数报错，推测是版本问题，于是用 os.makedirs 成功替代。之后依据 LeNet 的 7 层神经网络模型在 Jupyter Notebook 上复现了 LeNet 模型，并通过 plot_dataset_digits 函数[1]画出了 Colored MNIST 数据集中训练集和测试集中的部分数字，利用每个数字不同于手写 MNIST 数据集中 28*28*1 而是 28*28*3 的张量特性对 LeNet 模型进行了调整，将 LeNet 模型中 input 的 channels 由 1 改为 3 以实现对 Colored MNIST 数据集的训练。

随后，根据 Colored MNIST 类定义了训练数据集和测试数据集及数据加载器，同时将随机梯度下降(SGD)作为模型的优化器并且使用交叉熵损失函数作为损失函数。以 batch_size = 128，epochs = 50 在转化为张量的 Colored MNIST 数据集上训练和测试 LeNet 模型，以每十次返回一个包含 train loss、acc 和 test loss、acc 的结果，输出如图：

```
Colored MNIST dataset already exists
Colored MNIST dataset already exists
Epoch 10/50, Train Loss: 0.3521, Train Accuracy: 89.89%, Test Loss: 1.8687, Test Accuracy: 10.20%
Epoch 20/50, Train Loss: 0.3404, Train Accuracy: 89.89%, Test Loss: 2.0372, Test Accuracy: 10.20%
Epoch 30/50, Train Loss: 0.3380, Train Accuracy: 89.89%, Test Loss: 2.0222, Test Accuracy: 10.20%
Epoch 40/50, Train Loss: 0.3362, Train Accuracy: 89.89%, Test Loss: 2.0147, Test Accuracy: 10.20%
Epoch 50/50, Train Loss: 0.3345, Train Accuracy: 89.89%, Test Loss: 2.0145, Test Accuracy: 10.20%
```

从返回的结果可以看出，在缺少其他预处理、仅仅使用 LeNet 的情况下，模型在训练集上可以取得一个很高的准确率，但是一旦转移到了测试集，模型对数字的分类效果就急剧下降，准确率降为训练集的 1/9。这种情况，是典型的 OOD 情况：由于在测试集中数字的标签和数字的颜色的相关性由训练集 1 中的 0.9 和训练集 2 中的 0.8 降至 0.1，导致训练过程中产生的数字颜色和标签的虚假因果关系极大程度的影响了在测试集中对数字标签的判断。

2.2、带有预处理的测试和训练

在测试完缺少其他预处理的数据集后，为数据集定义了包含随机翻转、随机旋转、色彩抖动、归一化等预处理，以同样的 batch_size 和 epochs 在经过预处理的训练集上训练 LeNet 模型，同时在测试集上进行测试，返回的结果如图：

```
Epoch 40/50, Train Loss: 0.6811, Train Accuracy: 56.41%, Test Loss: 0.6788, Test Accuracy: 54.48%
Epoch 41/50, Train Loss: 0.6801, Train Accuracy: 56.45%, Test Loss: 0.6873, Test Accuracy: 51.97%
Epoch 42/50, Train Loss: 0.6793, Train Accuracy: 57.23%, Test Loss: 0.6737, Test Accuracy: 62.99%
Epoch 43/50, Train Loss: 0.6806, Train Accuracy: 56.55%, Test Loss: 0.6863, Test Accuracy: 50.86%
Epoch 44/50, Train Loss: 0.6794, Train Accuracy: 57.09%, Test Loss: 0.6835, Test Accuracy: 52.99%
Epoch 45/50, Train Loss: 0.6793, Train Accuracy: 57.20%, Test Loss: 0.6741, Test Accuracy: 63.01%
Epoch 46/50, Train Loss: 0.6802, Train Accuracy: 56.65%, Test Loss: 0.6781, Test Accuracy: 55.80%
Epoch 47/50, Train Loss: 0.6782, Train Accuracy: 57.17%, Test Loss: 0.6774, Test Accuracy: 56.34%
Epoch 48/50, Train Loss: 0.6790, Train Accuracy: 57.21%, Test Loss: 0.6768, Test Accuracy: 56.94%
Epoch 49/50, Train Loss: 0.6795, Train Accuracy: 57.01%, Test Loss: 0.6967, Test Accuracy: 51.28%
Epoch 50/50, Train Loss: 0.6779, Train Accuracy: 57.63%, Test Loss: 0.6802, Test Accuracy: 54.86%
```

可以看出，以上预处理在一定程度上很好的解决了原先测试集准确率过低的情况，由原先的 10%提升至 50%以上，有效的提高了 OOD 泛化能力。同时，在以上提到的预处理中，还发现色彩抖动其实是最关键的预处理操作，当预处理操作删去色彩抖动时，得出的结果与没有经

过预处理得出的结果非常接近。这一现象的原因也不难可以虚假的因果关系得出，是模型在训练过程中错误的将数字的颜色和数字的标签产生关联，而在颜色抖动过后，训练集中数字的颜色和数字的标签之间的相关性由原来的 0.9 和 0.8 显著下降，可以减少这种虚假的因果关系的产生。

2.3、对可能的处理的猜想

在得知虚假的因果关系是建立在数字颜色和数字标签上后，便可以在数字颜色至模型上进行各种处理。1.首先可以随机调整训练集中数字的颜色，这样可以使模型在训练过程中接触到不同数字颜色的图像，从而提高对数字颜色的鲁棒性，这种方法减少了数字颜色和标签的相关性，能他提高 OOD 泛化能力。2.同样属于减少了数字颜色和标签的相关性的，还有数字颜色的统一化，例如将所有图片的颜色转换为 RGB 空间中的灰色或者转换至灰度空间中，也可以有效的减少虚假因果关系的产生。3.再有例如噪声添加：为数字部分添加噪声，增加对数字颜色的鲁棒性。4.背景分离：分离出训练集中的背景部分和数字部分，利用没有颜色差异的背景训练，同时再测试中提取出数字的背景来判断数字的标签。5.多任务学习：先将训练集根据数字颜色分类，并引入多个任务，其中每个任务对应一个特定的数字颜色。通过同时训练多个任务，可以使模型对不同数字颜色的图像具有区分性，从而提高 OOD 泛化能力。6.模型集成：使用多个模型进行集成，通过结合多个模型的预测结果，或许可以提高对不同数字颜色的鲁棒性。7.预训练：先在数字颜色和数字标签相关性较小的数据集上进行预训练，然后将模型迁移到 Colored MNIST 数据集上，通过预训练，或许可以提高模型在 Colored MNIST 数据集上的 OOD 泛化能力。

三、算法复现

3.1、对 Invariant Risk Minimization(IRM)算法的理解及复现

正如前面提到的，IRM 算法希望学习一个仅仅依赖于 Xc 的参数Φ，使其对所有的分类器参数ω都同时最优。从代码角度而言，IRM 算法与一般的流程所不同的是引入了一个 IRM 惩罚项，它通过计算不同环境下的损失并进行平均和计算梯度，将每个环境下的梯度相乘并求和得到。IRM 惩罚项会与 ERM 损失相结合组成总损失，并在之后的反向传遍中利用总损失优化参数和模型。具体来说，IRM 算法首先计算在每个环境下的损失函数和梯度。然后，通过对这些梯度进行相似度的比较和惩罚，迫使模型在不同环境中产生相似的梯度，减少梯度的差异，以降低不稳定的 spurious feature 对模型的影响。通过使模型在不同环境中具有相似的梯度并使其向 0 逼近，IRM 方法能够消除环境特定的偏差，使模型更加关注输入数据

causal feature，而不是 spurious feature 同时还有良好的预测能力。通过减少不同环境下梯度的差异，IRM 惩罚项可以提高模型的鲁棒性和 OOD 泛化能力。当模型在不同环境中具有相似的梯度时，它能够更好地适应新的、未见过的环境，而不会过度依赖于特定的 spurious feature。这使得模型更具可迁移性，能够在不同的数据分布和环境中表现良好。

利用 github 中的开源代码[1]，在 Colored MNIST 数据集上用 IRM 算法进行了训练和测试。在 epoch 为 35 的时候得到了期望的结果：

```
Performance on train1 set: Average loss: 0.5201, Accuracy: 14845/20000 (74.22%)

Performance on train2 set: Average loss: 0.5071, Accuracy: 14863/20000 (74.31%)

Performance on test set: Average loss: 0.8663, Accuracy: 12311/20000 (61.55%)
```

利用 IRM 算法，模型对训练集的准确率达到了 74%，对测试集的准确率达到了 61%，足以证明 IRM 算法对 OOD 情况有着很好的泛化能力。

3.2、对 Out-of-Distribution Generalization via Risk Extrapolation(REx)算法的理解及复现

在本次大作业中，根据 Colored MNIST 数据集的 Correlation shift 的特征，选择了对 Correlation shift 数据集的准确率较高的 REx 算法进行复现。REx 算法是 IRM 算法出来后针对 OOD 问题的一个算法。该算法的主要思想是通过减小不同训练环境中的风险并使其尽可能一致，使模型更加关注不变的 causal feature，以降低模型对于 spurious feature 的依赖，同时引入了风险外推，将假设的测试领域是训练领域的线性组合拓展到更多可能的情况，使模型拥有更好的 OOD 泛化能力。

论文作者提出了 REx 算法的两个基本原则：1.降低训练集上的风险 2.使训练集上的风险尽可能一致。用公式表达，可以表达为 Minimax-REx(MM-REx)和 Variance-REx(V-REx)：

$$\mathcal{R}_{\text{MM-REx}}(\theta) \doteq \max_{\substack{\Sigma_e \lambda_e = 1 \\ \lambda_e \geq \lambda_{\min}}} \sum_{e=1}^{m} \lambda_e \mathcal{R}_e(\theta) \tag{6}$$

$$= (1 - m\lambda_{\min}) \max_e \mathcal{R}_e(\theta) + \lambda_{\min} \sum_{e=1}^{m} \mathcal{R}_e(\theta)$$

$$\mathcal{R}_{\text{V-REx}}(\theta) \doteq \beta \operatorname{Var}(\{\mathcal{R}_1(\theta), ..., \mathcal{R}_m(\theta)\}) + \sum_{e=1}^{m} \mathcal{R}_e(\theta)$$

在 MM-REx 中，m 意味着领域的数量，λmin 意味着风险外推的程度。当 λmin = 0 时，公式变为风险内插(risk interpolation，RI)，此时 MM-REx 更加侧重于单个领域的风险，而

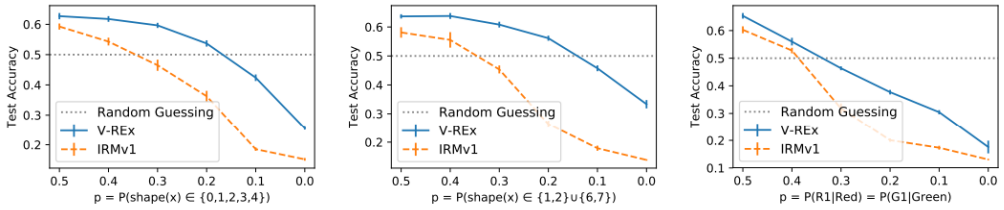
不太考虑多个领域之间的风险一致。当 $\lambda_{min} < 0$ 即为风险外推，并且当 $\lambda_{min} \to -\infty$时，为了最小化风险，各个训练领域的风险必须严格相等，即各个域的风险必须一致。在 V-REx 中，$\beta \in [0, \infty)$，β 的取值决定了是降低平均风险还是令各领域的风险一致。当 $\beta=0$ 时，就变成了普通的 ERM，而当 $\beta \to \infty$ 时 V-REx 就迫使各个领域的风险严格一致，使方差为 0，进而最小化风险。

REx 算法对于 IRM 算法的一个优点在于 REx 算法对于协变量转变(Covariate Shift)有一定的鲁棒性，所谓的协变量转变，是指训练数据和测试数据之间的输入特征分布发生变化，即 X 自身发生变化。REx 算法通过引入一个 V-REx 惩罚项，减少对不相关特征的依赖，降低各个领域的风险差异，还通过风险外推，以应对更多可能出现的情况，提高 OOD 泛化能力。

通过开源代码[2]，完成了在 Colored MNIST 数据集上 REx 算法的复现，结果如图：

```
Restart 0
step            train nll       train acc       rex penalty     irmv1 penalty
test acc
0               0.67663         0.62734         1.97640e-05     0.00021
0.40870
100             0.36623         0.85162         0.01707         0.00436
0.11740
200             0.59050         0.70236         4.01823e-07     0.00096
0.70400
300             0.58892         0.70318         1.27174e-07     0.00106
0.70480
400             0.58638         0.70528         1.22833e-07     0.00119
0.70310
500             0.58384         0.70678         1.20670e-07     0.00132
0.70190
highest test acc this run: 0.7031
Final train acc (mean/std across restarts so far):
0.70677996 0.0
Final test acc (mean/std across restarts so far):
0.7019 0.0
Highest test acc (mean/std across restarts so far):
0.7031 0.0
```

可以看出 REx 算法在 Colored MNIST 数据集上，不管是训练集还是测试集，都取得了 70%的准确率，并且在协变量转变的情况下，REx 的表现显著优于 IRM 算法：



三幅图分别带表三种不同的协变量转变情况[3]。结合之前 IRM 算法的结果，可以发现 REx 算法在 Colored MNIST 数据集上的准确率要略微高于 IRM 算法，并且还有一定能力解决协变量转化的情况，是一种解决 OOD 问题的有效手段。

3.3、对 IRM 算法的改进

IRM 算法对 penalty weight 参数较为敏感，它直接影响了 IRM 惩罚项在总体损失中的权重，即它对于 IRM 惩罚项在总损失中的贡献多少起了关键作用，不同的 penalty weight

参数对于模型最终的预测能力以及 OOD 泛化能力将起到重大作用。当 penalty weight 参数过小时，IRM 算法对于保持不变性的约束不够敏感。这可能导致算法更加依赖于 ERM loss，而忽视了不变性的要求，结果可能是在不同环境中的预测性能较好，但对于新环境的 OOD 泛化能力较差。另一方面，当 penalty weight 参数过大时，IRM 算法会过度强调不变性的要求，这可能导致算法过度抑制不同环境中的变化，从而降低了预测性能，此时，模型可能在训练环境中表现出很好的不变性，但对于新环境的适应能力较差。为了提高 IRM 算法的稳定性，便需要对 IRM 算法做出改良，在这里，提出了几种可能的方法。

1.动态调整 penalty weight 参数

在源码中，penalty weight 参数是一个和 epoch 有关的参数，可以考虑根据训练过程中的模型预测性能和损失情况动态地调整 penalty weight 的值。通过监控模型在不同环境下的表现，并根据需要增加或减小 penalty weight 的权重，或许可以提高 IRM 算法的稳定性。

2.修改总损失计算函数

总损失是由 ERM loss 和 IRM 惩罚项组成的一个参数，可以将总损失拓展，引入其他参数和 ERM loss 和 IRM 惩罚项一起约束总损失，比如不同环境下模型的预测性能差异和不同环境下的平均预测性能等。

3.引入多个 penalty weight 参数

通过引入多个 penalty weight 参数，并根据每个环境的数据分布特点来选择相应的参数，可以更加灵活高效地调整模型的性能和泛化能力。对于一个环境中具有较大数据方差的情况，可以使用相对较大的参数，以提高模型的泛化能力。同时，对于一个环境中数据方差较小的情况，可以使用相对较小的参数，以更好提高模型的预测性能。

4.正则化

可以在训练过程中对 penalty weight 参数施加 L1 或 L2 正则化约束，适当选择正则化参数的值，平衡模型性能和泛化能力，并减轻对参数的敏感性，从而提高 IRM 算法的稳定性。

四、总结

报告中，提出了对 ODD、IRM、REx 的理解，同时在 Colored MNIST 数据集上用有无预处理的数据训练和测试了 LeNet 模型，并据此提出了一些预处理的猜想。随后在 Colored MNIST 数据集上复现了 IRM 算法和 REx 算法，在对结果进行分析后提出了一些针对 penalty weight 参数的可能的 IRM 改良措施。本次大作业让我对 ODD 问题、机器学习的流程、数据预处理有

了更加深入的理解，也第一次接触了 IRM 算法和 REx 算法等其他针对 ODD 问题的算法，增强了读写代码能力，通过阅读论文，也让我对 IRM 和 REx 有了很深的印象，最重要的也是自己理解、思考和想办法解决问题的能力。

五、个人贡献及创新点

本次大作业全部为个人完成，本人在大作业中贡献为：理解 OOD、IRM、REx，阅读相关论文资料，复现 LeNet，调整所给 Colored MNIST 类代码，分析并测试相关预处理，完成训练与测试的代码并进行训练测试，利用开源代码测试 IRM 和 REx 在 Colored MNIST 数据集的性能，提出预处理的猜想和 IRM 算法的优化措施。

创新点：调整 Colored MNIST，引入不同的预处理，实现 LeNet 上训练和测试的代码，提出可能预处理的猜想，提出 IRM 算法优化措施。

参考

[1]https://colab.research.google.com/github/reiinakano/invariant-risk-minimization/blob/master/invariant_risk_minimization_colored_mnist.ipynb#scrollTo=9hYJRewnv80x

[2]https://github.com/capybaralet/REx_code_release/tree/master/InvariantRiskMinimization

[3] Krueger D, Caballero E, Jacobsen J H, et al. Out-of-distribution generalization via risk extrapolation (rex)[C]//International Conference on Machine Learning. PMLR, 2021: 5815-5826.

[4] Ye N, Li K, Bai H, et al. Ood-bench: Quantifying and understanding two dimensions of out-of-distribution generalization[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022: 7947-7958.

[5] Arjovsky M, Bottou L, Gulrajani I, et al. Invariant risk minimization[J]. arXiv preprint arXiv:1907.02893, 2019.