Invariant Risk Minimization

problem

1. machine learning所使用的data往往受到selection bias.confounding bias等的影响.
2. spurious correlations源自data中的bias,这部分correlation一般不会展现出stable的性质
3. 在训练模型中采用shuffling的操作会丢失掉数据分布如何发生变化的信息

related work

1. Invariant causal prediction的方法是假设在观测变量之间存在causal graph.这在图像识别领域很难开展因为像素点之间causal graph可能并不存在很强的物理含义.并且ICP模型只适用于线性模型并且规模会随着变量数量的增加指数级增长

problem setting

从多个环境$e \in \mathcal{E}_{tr}$中搜集数据:$D_e := \{(x_i^e, y_i^e)\}_{i=1}^{n_e}$,使用多个环境下的数据集学习得到一个预测器$Y \approx f(X)$使得,下式最小化:

$$R^{\mathrm{OOD}}(f) = \max_{e \in \mathcal{E}_{\mathrm{all}}} R^e(f) \tag{1}$$

其中$R^e(f) := \mathbb{E}_{X^e, Y^e}[\ell(f(X^e), Y^e)]$表示在环境$e$下的预测器的误差.下面是OOD问题的一个实例.考虑如下的SCM(**example 1**):

$$\begin{aligned} X_1 &\leftarrow \mathrm{Gaussian}(0, \sigma^2), \\ Y &\leftarrow X_1 + \mathrm{Gauseian}(0, \sigma^2), \\ X_2 &\leftarrow Y + \mathrm{Gaussian}(0, 1). \end{aligned} \tag{2}$$

一共存在两个环境:

$$\mathcal{E}_{\mathrm{tr}} = \{\mathrm{replace} \sigma^2 \mathrm{by} 10, \mathrm{replace} \sigma^2 \mathrm{by} 20\}. \tag{3}$$

针对不同的变量进行线性拟合,可以得到:

- regress from X_1^e, to obtain $\hat{\alpha}_1 = 1$ and $\hat{\alpha}_2 = 0$,
- regress from X_2^e, to obtain $\hat{\alpha}_1 = 0$ and $\hat{\alpha}_2 = \sigma(e)^2/(\sigma(e)^2 + \frac{1}{2})$,
- regress from (X_1^e, X_2^e), to obtain $\hat{\alpha}_1 = 1/(\sigma(e)^2+1)$ and $\hat{\alpha}_2 = \sigma(e)^2/(\sigma(e)^2+1)$.

可以看到只有将X_1作为因变量来进行拟合时得到的系数是在各个环境中都保持不变的,当一起考虑X_1与X_2时,取较大的方差值时,X_2前的系数会非常大,而且随方差σ而变(matlab的代码如下).

```
clear;clc
%说明只有在各个环境中特征前的系数保持不变就说明这个特征在各个环境中是invariant
sigma=1;%改变方差来生成多个环境的数据
num_sample=10000;
X_1=normrnd(0,sigma^0.5,num_sample,1);
Y=X_1+normrnd(0,sigma^0.5,num_sample,1);
X_2=Y+normrnd(0,1,num_sample,1);
X_cat=[ones(length(Y),1) X_1 X_2];
p1=polyfit(X_1,Y,1);
p2=polyfit(X_2,Y,1);
[b,bint,r,rint,stats]=regress(Y,X_cat);
```

解决OOD问题的主要方法

1. ERM(empirical risk minimization)将多个环境的数据直接混合学习训练误差最小的模型.但可以从上述案例中看到如果将两个环境的数据混合起来,那么X_2之前的系数会非常大.从而学到的特征并不是causal的或者说随着环境的变化并不invaraint
2. 最小化

$$R^{\text{rob}}(f) = \max_{e \in \mathcal{E}_{\text{tr}}} R^e(f) - r_e \tag{4}$$

其中r_e是一个环境的baseline,如果将baseline设置为0,那么相当于最小化在误差最大的环境下的误差.有定理显示,这类robust learning的手段相当于最小化在各个环境下的加权误差值.所以ERM方法相当于是各个环境中权重一致的robust learning的手段.这一类方法很难超越训练集的环境范围,即无法应用至无限的OOD问题之上.例如我们训练集中的数据σ都取的非常大,那么环境混合之后得到的模型可能都会给X_2一个很大的权重系数,而当我们的测试集中σ很小,那么模型在测试集中的表现就会非常糟糕

Proposition 2. *Given KKT differentiability and qualification conditions,* $\exists \lambda_e \geq 0$ *such that the minimizer of* R^{rob} *is a first-order stationary point of* $\sum_{e \in \mathcal{E}_{tr}} \lambda_e R^e(f)$.

3. domain adaptation方法是尝试学到表征$\Phi(X_1, X_2)$在所有环境中保持分布一致.使用上面的例子,causal representation X_1在各个环境中的分布可能是不同的,因为方差不同.

4. ICP方法,ICP方法在案例环境中也没办法使用,因为环境的intervention直接作用在了target变量Y之上,所以导致Y的残差的方差在各个环境中其实是会发生改变的.

contribution

提出了IRM,从多个环境学习用于估计非线性的具有invaraint特性的causal predictor.

IRM的具体形式

IRM的任务主要分为两个部分,首先利用映射$\Phi:\mathcal{X}\to\mathcal{H}$获得invaraint representation.接着在这些representation之上训练权重参数(分类器):$w:\mathcal{H}\to\mathcal{Y}$,权重$\omega$在各个环境中能够同时达到最优:$w\in\arg\min_{\bar{w}:\mathcal{H}\to Y}R^e(\bar{w}\circ\Phi)\ for\ all\ e\in\mathcal{E}$.

IRM的目标可以分为两部分:预测准确与在环境之间invaraint.目标函数为

$$\min_{\substack{\Phi:\mathcal{X}\to\mathcal{H}\\ \omega:\mathcal{H}\to\mathcal{Y}}}\sum_{e\in\mathcal{E}_{\text{tr}}}R^e(w\circ\Phi) \tag{5}$$

$$\text{subject to } w\in\arg\min R^e(\bar{w}\circ\Phi),\text{for all } e\in\mathcal{F}_{\text{tr}}. \tag{6}$$

由于双重优化问题很难求解,就转化为为如下的形式

$$\min_{\Phi:\ X\to Y}\sum_{e\in\mathcal{E}_{\text{cr}}}R^e(\Phi)+\lambda\cdot\|\nabla_{w\|w=1.0}R^e(w\cdot\Phi)\|^2, \tag{7}$$

penalty项表示在每个环境中ω都是最优的.

从IRM到IRMv1

将IRM中的硬约束转化为惩罚项:

$$L_{\text{IRM}}(\Phi,w)=\sum_{e\in\mathcal{E}_{\text{tr}}}R^e(w\circ\Phi)+\lambda\cdot\mathbb{D}(w,\Phi,e) \tag{8}$$

其中$\mathbb{D}$表示ω与ω^e(能令$R^e(\omega\circ\Phi)$最小)之间的距离.

考虑**线性分类器**(即上述的案例,其中$\omega=[1\ 0]$),即ω^e可以直接使用下式进行计算

$$w_\Phi^e=\mathbb{E}_{X^e}\left[\Phi(X^e)\Phi(X^e)^\top\right]^{-1}\mathbb{E}_{X^e,Y^e}\left[\Phi(X^e)Y^e\right], \tag{9}$$

于是ω与最优值之间的差距可以描述为

$$\mathbb{D}_{\text{dist}}(w, \Phi, e) = \|w - w_{\Phi}^{e}\|^2. \tag{10}$$

其中$\Phi(x) = x * \text{Diag}([1, c]), \omega = (1, 0)$.**我们已知当$c = 0$则取到最佳的$\Phi$.而(10)其实并不是一个优秀的惩罚项**,随着c的变化,(10)的变化见下图:

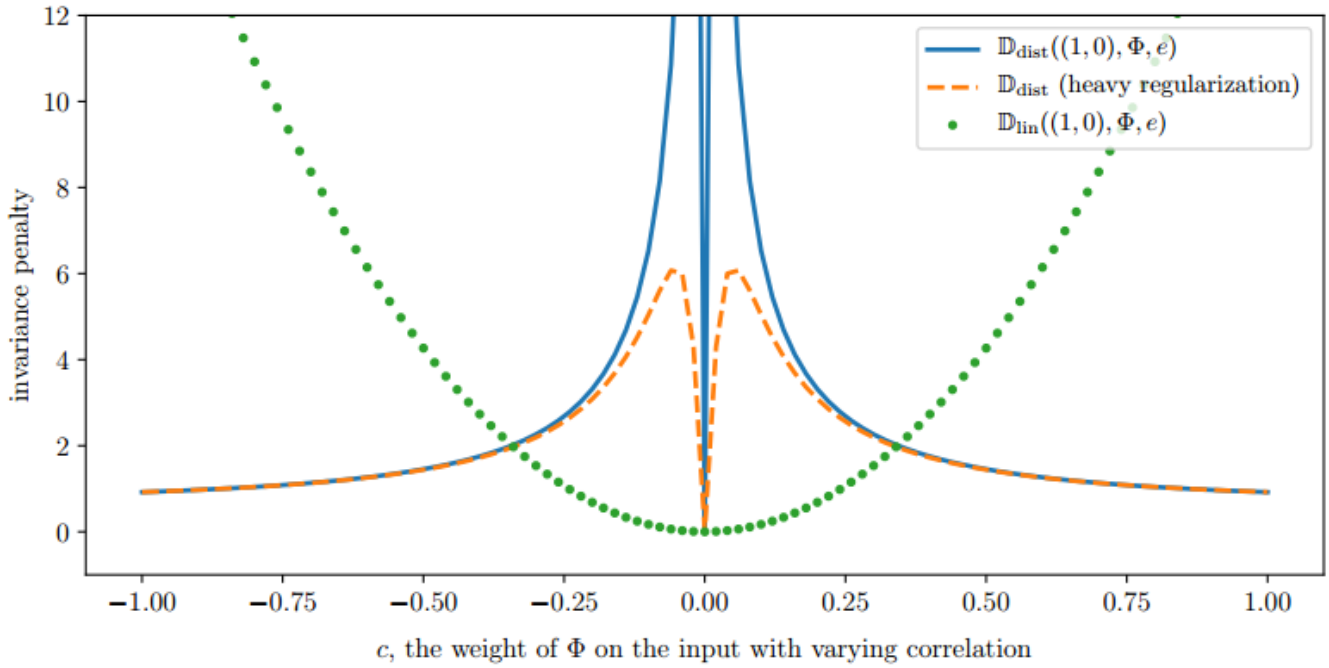


Figure 1: Different measures of invariance lead to different optimization landscapes in our Example 1. The naïve approach of measuring the distance between optimal classifiers $\mathbb{D}_{\text{dist}}$ leads to a discontinuous penalty (solid blue unregularized, dashed orange regularized). In contrast, the penalty $\mathbb{D}_{\text{lin}}$ does not exhibit these problems.

在c靠近0时惩罚项会趋于无穷大,这是因为产生了一个极大的w来弥补X_2对应的映射较小的问题.而将penalty项修改为

$$\mathbb{D}_{\text{lin}}(w, \Phi, e) = \left\|\mathbb{E}_{X^e}\left[\Phi(X^e)\Phi(X^e)^{\mathrm{T}}\right] w - \mathbb{E}_{X^e, Y^e}\left[\Phi(X^e)Y^e\right]\right\|^2, \tag{11}$$

效果会有所改善.而这个过程其实也会存在问题,**显然当c取的非常大的时候,penalty项的值也会比较小,但注意此时X_2的权重越来越大,(8)中前面的预测误差也会越来越大的.所以这里面还可以继续深入讨论**,而且要使用该penalty就得提前知道ω,这很难实现

固定权重参数

那么在使用(11)作为penalty时,也会存在问题,即我们考虑变换对$(\gamma\Phi, \frac{1}{\gamma}w)$,这个变量对不会改变损失函数中的ERM项因为

$$w \circ \Phi = \underbrace{\left(w \circ \Psi^{-1}\right)}_{\bar{w}} \circ \underbrace{\left(\Psi \circ \Phi\right)}_{\bar{\Phi}}. \tag{12}$$

但是**假如γ取接近于0的值,那么$\mathbb{D}_{lin}$就会接近于0,所以最终的最优解就变成了无representation,并且权重系数极大.**

为此如果固定$\omega = \tilde{\omega}$,于是可以求出该ω对应的最佳的representation Φ,作者固定了$\omega = (1, 0)$,来确定最优的representation Φ:

$$L_{\text{IRM},w=1.0}(\Phi^\top) = \sum_{e \in \mathcal{E}_{\text{tr}}} R^e(\Phi^\top) + \lambda \cdot \mathbb{D}_{\text{lin}}(1.0, \Phi^\top, e). \tag{13}$$

接着后一项又可以写成$\mathbb{D}(1.0, \Phi, e) = \|\nabla_{w|w=1.0} R^e(w \cdot \Phi)\|^2$.并且作者说明了在linear Invariant prediction中使用$\omega = 1$就完全能够学到一个合适的Φ.**我对这一部分属实难以理解,我认为你得提前知道最佳的ω才能确定真正的Φ,不然怎么做到最优呢?当然在这种线性的情况下确实只需要一个标量ω即可,不需要一个正确值,因为如果差了一个倍数,可以通过Φ补充回来**

Invariance, causality and generalization

本文提出的IRM principle 能够帮助模型**在训练环境中**有低误差以及Invariant的能力,那么如何确保这种效果能在所有的环境$\mathcal{E}$中保持呢?

作者首先给出了IRM适用于哪种环境或者说对于数据的分布需要什么假设.相较于ICP论文中要求intervention不作用在Y之上,本文将假设放宽了,允许intervention改变Y的方差.即environment 需要满足以下条件则IRM方法可以泛化至所有的环境之中

Definition 7. *Consider a SEM $\mathcal{C}$ governing the random vector $(X_1, \ldots, X_d, Y)$, and the learning goal of predicting Y from X. Then, the* set of all environments *$\mathcal{E}_{\text{all}}(\mathcal{C})$ indexes all the interventional distributions $P(X^e, Y^e)$ obtainable by* valid interventions *e. An intervention $e \in \mathcal{E}_{all}(\mathcal{C})$ is valid as long as (i) the causal graph remains acyclic, (ii) $\mathbb{E}[Y^e|\text{Pa}(Y)] = \mathbb{E}[Y|\text{Pa}(Y)]$, and (iii) $\mathbb{V}[Y^e|\text{Pa}(Y)]$ remains within a finite range.*

从$\mathcal{E}_{tr}$到$\mathcal{E}_{all}$

作者分两步从$\mathcal{E}_{tr}$推广到$\mathcal{E}_{all}$

实验与代码

合成数据实验

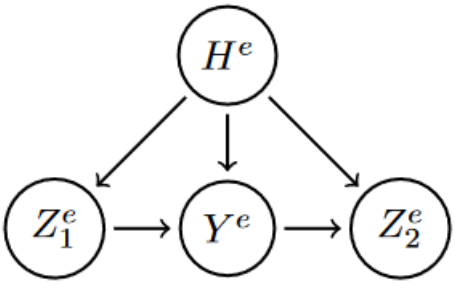
实验设置

1. 将example 1中的两个输入特征$X = (X_1, X_2)$拓展到10维
2. 引入了10维的环境confounder 变量H
3. 考虑了特征Z无法被直接观测到,于是引入了scramble version $X = SZ$,即观测到的变量可能是原变量的一种混合

三种设置分别可以有以下的取值:

- *Scrambled* (S) observations, where S is an orthogonal matrix, or *unscrambled* (U) observations, where $S = I$.
- *Fully-observed* (F) graphs, where $W_{h\to 1} = W_{h\to y} = W_{h\to 2} = 0$, or *partially-observed* (P) graphs, where $(W_{h\to 1}, W_{h\to y}, W_{h\to 2})$ are Gaussian.
- *Homoskedastic* (O) Y-noise, where $\sigma_y^2 = e^2$ and $\sigma_2^2 = 1$, or *heteroskedastic* (E) Y-noise, where $\sigma_y^2 = 1$ and $\sigma_2^2 = e^2$.

实验的causal graph表示如下



$$
\begin{aligned}
H^e &\leftarrow \mathcal{N}(0, e^2) \\
Z_1^e &\leftarrow \mathcal{N}(0, e^2) + W_{h\to 1} H^e \\
Y^e &\leftarrow Z_1^e \cdot W_{1\to y} + \mathcal{N}(0, \sigma_y^2) + W_{h\to y} H^e \\
Z_2^e &\leftarrow W_{y\to 2} Y^e + \mathcal{N}(0, \sigma_2^2) + W_{h\to 2} H^e
\end{aligned}
$$

Figure 3: In our synthetic experiments, the task is to predict Y^e from $X^e = S(Z_1^e, Z_2^e)$.

采用了三种训练环境$\mathcal{E}_{tr} = \{0.2, 2, 5\}$

实验代码

参数设置

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Invariant regression')
    parser.add_argument('--dim', type=int, default=10)
    parser.add_argument('--n_samples', type=int, default=1000)
    parser.add_argument('--n_reps', type=int, default=10)
    parser.add_argument('--skip_reps', type=int, default=0)
    parser.add_argument('--seed', type=int, default=0)  # Negative is random
    parser.add_argument('--print_vectors', type=int, default=1)
    parser.add_argument('--n_iterations', type=int, default=100000)
    parser.add_argument('--lr', type=float, default=1e-3)
    parser.add_argument('--verbose', type=int, default=0)
    parser.add_argument('--methods', type=str, default="ERM,ICP,IRM")
    parser.add_argument('--alpha', type=float, default=0.05)
    parser.add_argument('--env_list', type=str, default=".2,2.,5.")
    parser.add_argument('--setup_sem', type=str, default="chain")
    parser.add_argument('--setup_ones', type=int, default=1)
    parser.add_argument('--setup_hidden', type=int, default=0)
    parser.add_argument('--setup_hetero', type=int, default=0)
    parser.add_argument('--setup_scramble', type=int, default=0)
    args = dict(vars(parser.parse_args()))
```

其中参数 `verbose` 参数表示是否要输出日志,最后的三个参数对应上述的三种设置.通过调用 `run_experiment(args)` 来得到结果.

实验数据生成

1. 实验的随机种子设置

```
if args["seed"] >= 0:
    torch.manual_seed(args["seed"])
    numpy.random.seed(args["seed"])
    torch.set_num_threads(1)
```

numpy与torch的随机种子都需要设置,numpy的随机种子只能限制numpy的随机数生成代码: `np.random.randn(100)` ;torch的随机种子可以约束torch的随机数生成 `torch.randn(100)` ,可以参考
2. 实验设定(使用format函数进行字符串的填充)

```
if args["setup_sem"] == "chain":
    setup_str = "chain_ones={}_hidden={}_hetero={}_scramble={}".format(
        args["setup_ones"],
        args["setup_hidden"],
        args["setup_hetero"],
        args["setup_scramble"])
```

3. 多环境数据生成

```
#一共循环n_reps次,并且每次循环都要产生三种环境
for rep_i in range(args["n_reps"]):
    if args["setup_sem"] == "chain":
        sem = ChainEquationModel(args["dim"],
                                 ones=args["setup_ones"],
                                 hidden=args["setup_hidden"],
                                 scramble=args["setup_scramble"],
                                 hetero=args["setup_hetero"])

        env_list = [float(e) for e in args["env_list"].split(",")]
        environments = [sem(args["n_samples"], e) for e in env_list]#不同的env表示不同的方差
    else:
        raise NotImplementedError

    all_sems.append(sem)
    all_environments.append(environments)#搜集各个环境的数据
```

4. 有向无环图生成类

```
class ChainEquationModel(object):
    def __init__(self, dim, ones=True, scramble=False, hetero=True, hidden=False):
        self.hetero = hetero
        self.hidden = hidden
        self.dim = dim // 2

        if ones:
            self.wxy = torch.eye(self.dim)
            self.wyz = torch.eye(self.dim)
        else:
            self.wxy = torch.randn(self.dim, self.dim) / dim
            self.wyz = torch.randn(self.dim, self.dim) / dim

        if scramble:
            self.scramble, _ = torch.qr(torch.randn(dim, dim))
        else:
            self.scramble = torch.eye(dim)

        if hidden:
            self.whx = torch.randn(self.dim, self.dim) / dim
            self.why = torch.randn(self.dim, self.dim) / dim
            self.whz = torch.randn(self.dim, self.dim) / dim
        else:
            self.whx = torch.eye(self.dim, self.dim)
            self.why = torch.zeros(self.dim, self.dim)
            self.whz = torch.zeros(self.dim, self.dim)

    def solution(self):
        w = torch.cat((self.wxy.sum(1), torch.zeros(self.dim))).view(-1, 1)
        return w, self.scramble

    def __call__(self, n, env):
        h = torch.randn(n, self.dim) * env
        x = h @ self.whx + torch.randn(n, self.dim) * env

        if self.hetero:
            y = x @ self.wxy + h @ self.why + torch.randn(n, self.dim) * env
            z = y @ self.wyz + h @ self.whz + torch.randn(n, self.dim)
        else:
            y = x @ self.wxy + h @ self.why + torch.randn(n, self.dim)
            z = y @ self.wyz + h @ self.whz + torch.randn(n, self.dim) * env

        return torch.cat((x, z), 1) @ self.scramble, y.sum(1, keepdim=True)
```

通常情况下， `__call__` 方法是一个类的特殊方法，它允许将对象实例视为函数。这意味着可以像调用函数一样调用对象实例。在这种情况下，这个对象应该是一个类的实例，并且这个类应该定义了一个 `__call__` 方法。 `@` 符号是张量的矩阵乘法操作。 `solution()` 函数给出各个变量权重的ground truth,将wxy的每一行相加并在最后加上5个0(前五个表示causal 变量的weight,后5个表示non-causal的变量的权

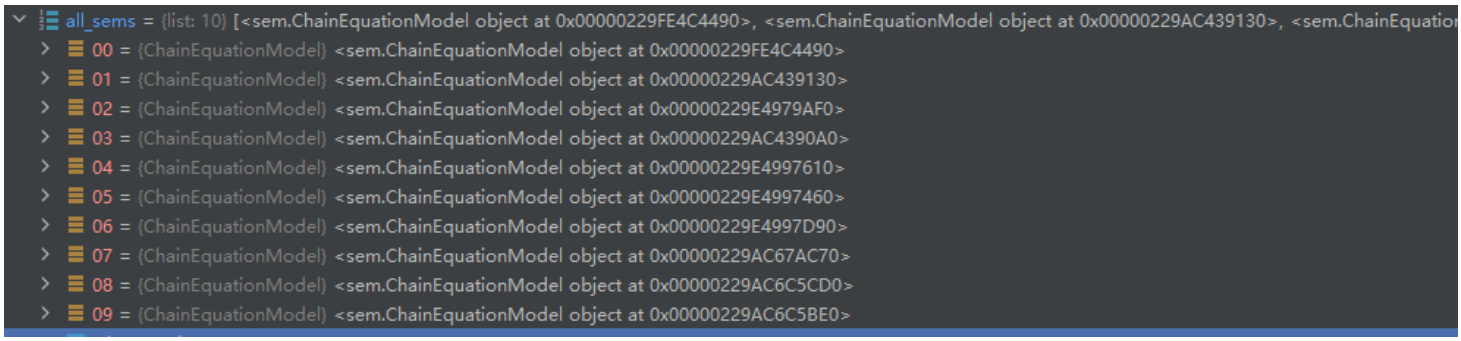
重)

这段代码的生成过程与上述的实验设置中的causal graph是对应的,用变量x表示Z_1^e,用变量z表示Z_2^e.w表示两个变量之间的权重系数.并且最终每个环境中观测到的特征是x, z经过混杂后得到的,

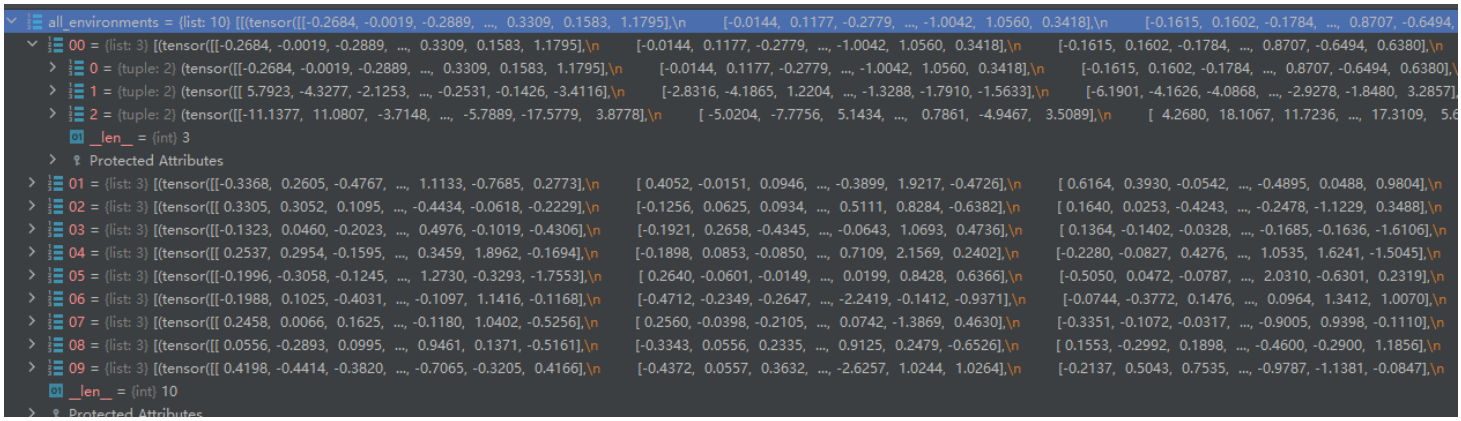
即 `torch.cat((x, z), 1) @ self.scramble, y.sum(1, keepdim=True)`

5. 最终生成的合成数据

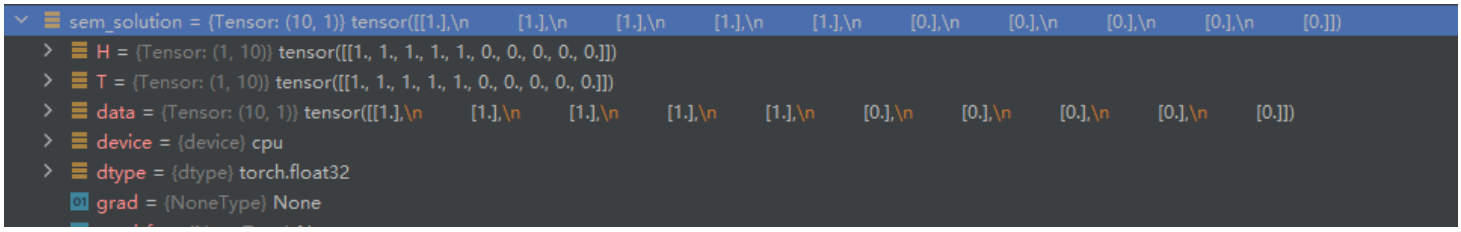
数据中包含9个SEM(结构应该是一样的),其中每个SEM中取三个不同的方差来进行采样



每个SEM对应三个environments



最终的权重系数(ground truth)为



ERM,ICP,IRM模型测试与对比

针对每个SEM存在3个environments,将这三个Environments(其中包含x,y的数据)交给ERM,ICP,IRM模型来进行训练,然后通过对比对权重系数wxy, wyz的估计来评判模型的准确性.

6. IRM的训练过程如下:

```
class InvariantRiskMinimization(object):
    def __init__(self, environments, args):
        best_reg = 0
        best_err = 1e6

        x_val = environments[-1][0]
        y_val = environments[-1][1]

        for reg in [0, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1]:
            self.train(environments[:-1], args, reg=reg)
            err = (x_val @ self.solution() - y_val).pow(2).mean().item()  # 经过训练之后模型的误差

            if args["verbose"]:
                print("IRM (reg={:.3f}) has {:.3f} validation error.".format(
                    reg, err))

            if err < best_err:
                best_err = err
                best_reg = reg
                best_phi = self.phi.clone()
        self.phi = best_phi

    def train(self, environments, args, reg=0):
        dim_x = environments[0][0].size(1)

        self.phi = torch.nn.Parameter(torch.eye(dim_x, dim_x))
        self.w = torch.ones(dim_x, 1)  # 固定这部分的参数,即固定后续分类器的参数
        self.w.requires_grad = True

        opt = torch.optim.Adam([self.phi], lr=args["lr"])  # 优化特征提取器的参数
        loss = torch.nn.MSELoss()

        for iteration in range(args["n_iterations"]):
            penalty = 0
            error = 0
            for x_e, y_e in environments:
                error_e = loss(x_e @ self.phi @ self.w, y_e)  # 计算ERM误差
                penalty += grad(error_e, self.w,
                                create_graph=True)[0].pow(2).mean()  # 计算ERM的残差对于输入的导数
                error += error_e

            opt.zero_grad()
            (reg * error + (1 - reg) * penalty).backward()  # reg值表示惩罚项的权重
            opt.step()

            if args["verbose"] and iteration % 1000 == 0:
                w_str = pretty(self.solution())
                print("{:05d} | {:.5f} | {:.5f} | {:.5f} | {}".format(iteration,
                                                                      reg,
                                                                      error,
```

```
                                                                penalty,
                                                                w_str))

    def solution(self):
        return (self.phi @ self.w).view(-1, 1)
```

其中reg参数代表惩罚项前的系数值,作者在这里尝试了多个系数.并且以三个环境中的最后一个environment作为验证集: `x_val = environments[-1][0] y_val = environments[-1][1]` 并记录最佳的reg,最小的误差以及最佳的Φ.训练过程中确定特征提取模块以及最终的分类器部分均为线性的,并且只对特征提取模块Φ进行更新:

```
self.phi = torch.nn.Parameter(torch.eye(dim_x, dim_x))
self.w = torch.ones(dim_x, 1)  # 固定这部分的参数,即固定后续分类器的参数
self.w.requires_grad = True

opt = torch.optim.Adam([self.phi], lr=args["lr"])  # 优化特征提取器的参数
loss = torch.nn.MSELoss()
```

模型各项损失的计算如下:

```
for x_e, y_e in environments:
    error_e = loss(x_e @ self.phi @ self.w, y_e)  # 计算ERM误差
    penalty += grad(error_e, self.w,
                    create_graph=True)[0].pow(2).mean()  # 计算ERM的残差对于输入的导数
    error += error_e

opt.zero_grad()
(reg * error + (1 - reg) * penalty).backward()  # reg值表示惩罚项的权重
```

7. ICP模型寻找Invariant set的过程如下,ICP模型是首先选出合适的变量集合,接着在变量集合上使用线性拟合得到权重系数再进行指标量的计算

将多个环境的数据整合并标记环境index

```
for e, (x, y) in enumerate(environments):
    x_all.append(x.numpy())
    y_all.append(y.numpy())
    e_all.append(np.full(x.shape[0], e))

x_all = np.vstack(x_all)
y_all = np.vstack(y_all)  # 把三个维度的list直接合成一个维度的
e_all = np.hstack(e_all)
```

候选特征集合的迭代:

```
def powerset(self, s):
    return chain.from_iterable(combinations(s, r) for r in range(len(s) + 1))
```

t检验与f检验结果

```
def mean_var_test(self, x, y):
    pvalue_mean = ttest_ind(x, y, equal_var=False).pvalue
    pvalue_var1 = 1 - fdist.cdf(np.var(x, ddof=1) / np.var(y, ddof=1),
                                x.shape[0] - 1,
                                y.shape[0] - 1)

    pvalue_var2 = 2 * min(pvalue_var1, 1 - pvalue_var1)
```

一些想法

1. 不使用原观测特征而使用原特征映射得到的特征,有两方面的原因: 像素之间存在causal graph的这个假设很不现实,或者说物理意义不明显;另一方面使用原特征映射得到的特征是前一种方法的扩展.(但是在电网之中使用原特征进行操作还是有很强的物理意义的)
2. OOD问题的直接目标是最小化在所有环境中最大的误差,如(1)所示,那么这个问题与robust learning的目标(4)其实是比较一致的,当baseline完全取为0时是完全一致的.那么IRM要在这个基础上保证权重系数ω在所有环境下都能达到最小,并且要求所有环境中的总误差最小.
3. IRM主要解决的是$P(Y|X)$在变化的问题,重点不是$P(X)$在变化的问题,虽然IRM支持representation在不同的环境下分布不同,但这其实不是他解决的重点,他的重点是X与Y的作用关系中的变与不变并且加以利用
4. 使用(11)作为penalty可能会存在Φ并没有取到最合适的值,但是通过无限减小γ来获得最优解
5. 这篇文章关于penalty项的合理性以及取$\omega = 1$之类的操作其实是**局限于线性假设的**