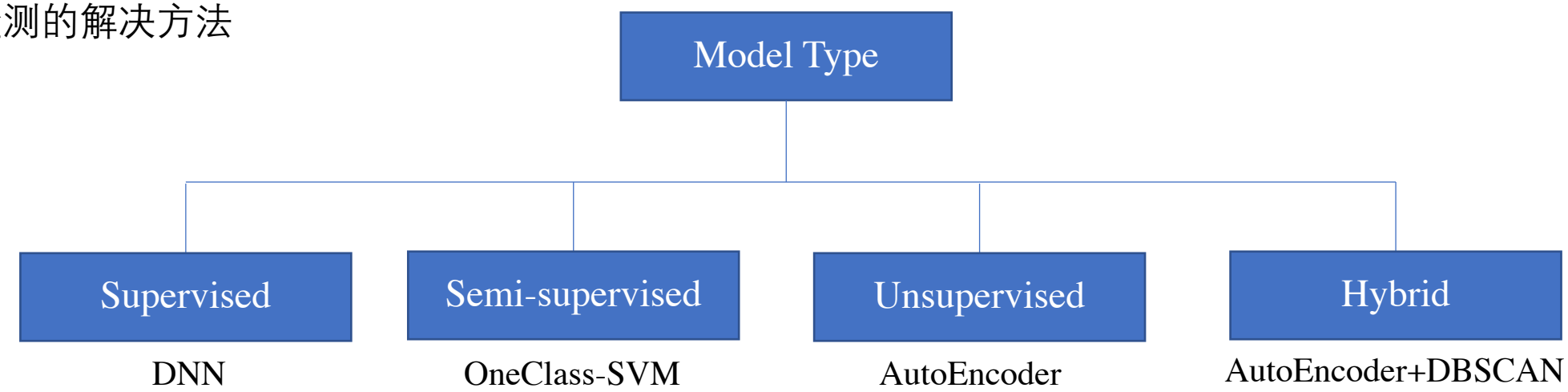
Anomaly Detection

什么是异常

异常就是偏离数据主要分布的点或者群体，导致异常的原因有很多，比如恶意行为，系统故障，蓄意欺诈等，因此，异常检测对于决策系统是必须的。

异常检测的解决方法

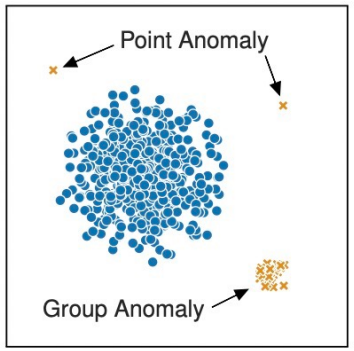


异常一般分成3个类别：

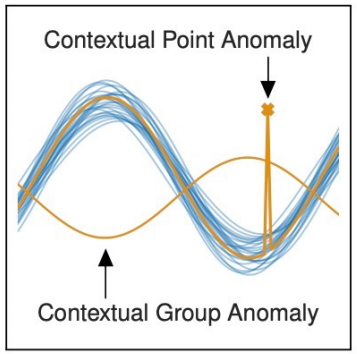
1）点异常

2）连续性异常(群组异常)

3）上下文异常(时间序列)



1)点异常和 2)连续性异常



3)上下文异常

课程目录

1. 群组异常检测

2. 离群点异常检测

3. 时间序列异常检测

4. 资金关系异常检测

FRAUDAR: Bounding Graph Fraud in the Face of Camouflage

FRAUDAR

解决的问题：在一个二部图中，找到最密集的一个子图作为最异常的群组（并且这个密集子图是抗伪装的）

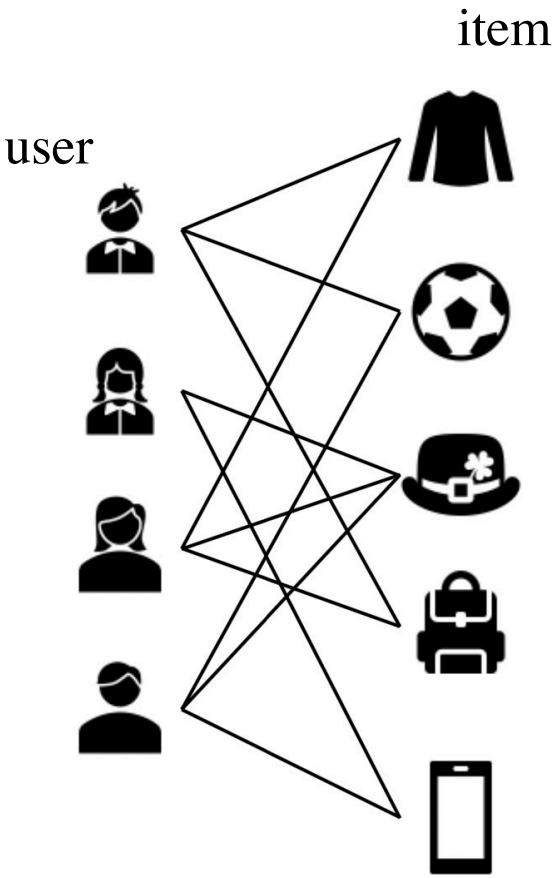
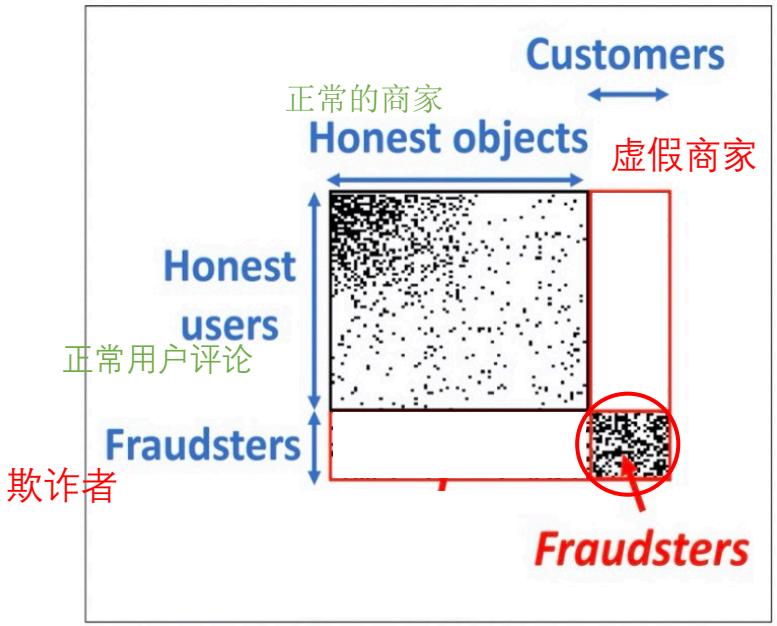


Figure 2.7 An e-commerce bipartite graph

那么，Fraudar可以解决哪些问题呢？（二部图的问题）

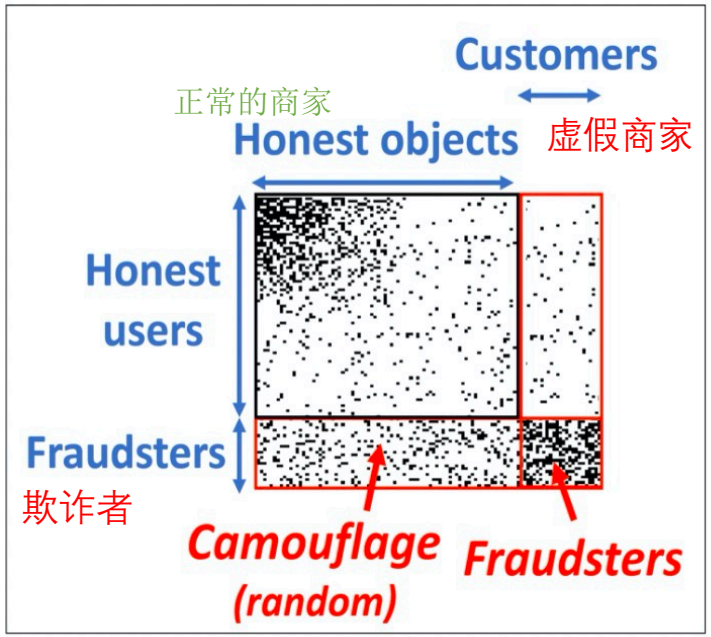
1. 用户在电商平台，购买商品
2. 用户评论电影，美食，游戏评分
3. 推特，博客关注信息
4. 支付场景，用户之间转账
5. 注册，登入场景异常聚集



商家的虚假评论

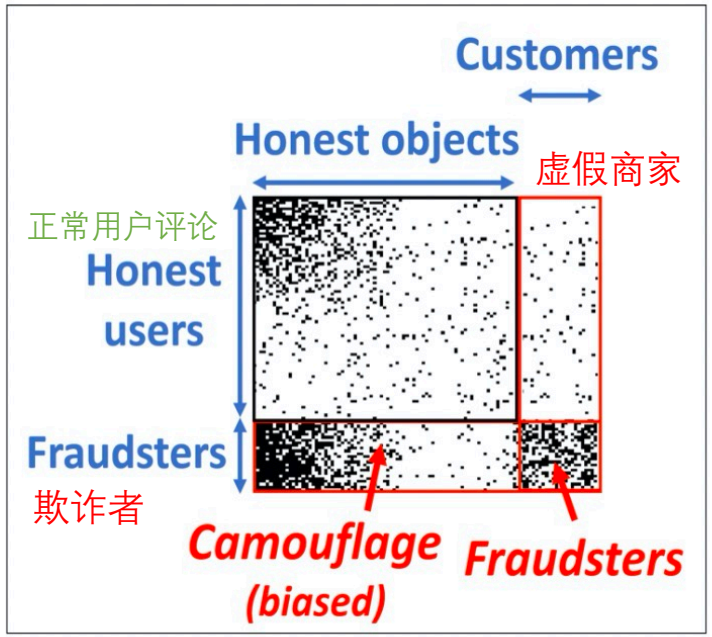
发现密集的欺诈者和欺诈虚假商家

伪装：伪装的目的是为了降低密集图的异常性，从而降低被识别出来的可能性



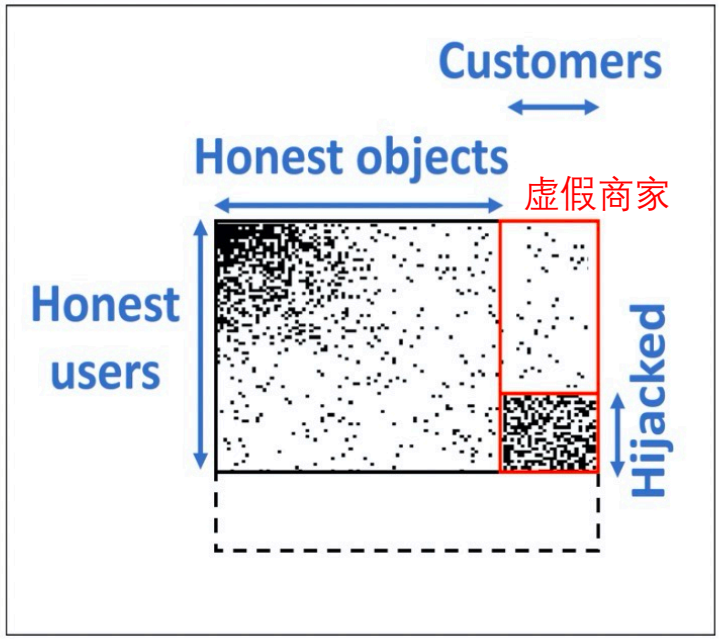
(a) Attacks with random camouflage

随机伪装



(b) Attacks with biased camouflage

有偏伪装



(c) Hijacked accounts

劫持账户（盗用）

抗伪装的定义

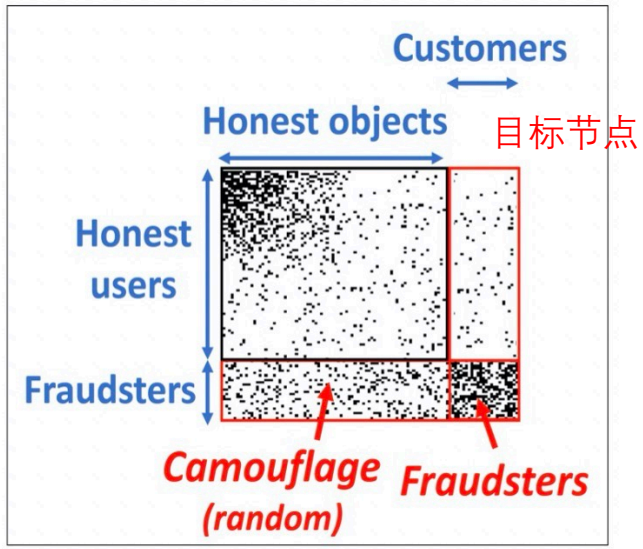
DEFINITION 1. *Let* $(\mathcal{A}, \mathcal{B})$ *be a block consisting of fraudulent users and objects. A density metric* g *is* camouflage-resistant *if when any amount of camouflage is added by the adversary,* $g(\mathcal{A} \cup \mathcal{B})$ *does not decrease.*

如果一个欺诈团伙，在增加伪装后，
其欺诈指标没有下降，就认为该方法是抗伪装。
也就是增加伪装也不会降低对他的怀疑。

我们如何去让我们的模型能够抗伪装呢？

欺诈者是容易添加伪装的，但是恶意商家是很难去让好用户去购买的

当目标节点有很高的度时，我们标记对目标节点的边一个低的权重



比如电商商户：华为，小米，苹果，大量的用户去购买商品，那么用户去购买这个商户产品时，这条边就给予一个低的边权重。反之，没有名气的商户应给予一个高的边权重。

A

B


C

D

1	C_a	C_b	
2	C_a	C_b	
3			C_c
.			
.			
n	C_a		C_c

1. 欺诈者添加正常商品边权重比较低，不会包含在密集子图中
2. 当添加边到正常商品时，没有对欺诈块添加或移除边
3. 欺诈用户和欺诈商品的边权重，没有因为添加正常商品而变化

如何通过列来重新计算边的权重C_{ij}

	A	B	C
	1	0	0
	1	1	1
	1	1	0
	1	0	1
	1	0	0
	1	0	0
	0	0	1
d_j	6	2	3

A	B	C
0.41	0	0
0.41	0.51	0.48
0.41	0.51	0
0.41	0	0.48
0.41	0	0
0.41	0	0
0	0	0.48

$$h_A(x) = \frac{1}{\log(5+6)} = 0.41$$

$$h_B(x) = \frac{1}{\log(5+2)} = 0.51$$

$$h_C(x) = \frac{1}{\log(5+3)} = 0.48$$

$$h(x) = \frac{1}{\log(c+d_j)} = 0.41$$

常数c=5

作者定义的异常指标

$$g(\mathcal{S}) = \frac{f(\mathcal{S})}{|\mathcal{S}|} \quad (1)$$

异常值 $f(\mathcal{S})$

节点数量 $|\mathcal{S}|$

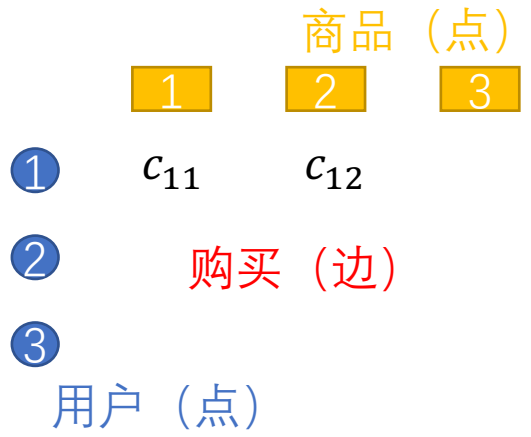
异常指标 $g(\mathcal{S})$

Let $\mathcal{A} \subseteq \mathcal{U}$ be a subset of users and $\mathcal{B} \subseteq \mathcal{W}$ be a subset of objects. Let $\mathcal{S} = \mathcal{A} \cup \mathcal{B}$, and $\mathcal{V} = \mathcal{U} \cup \mathcal{W}$. For the rest of this paper,

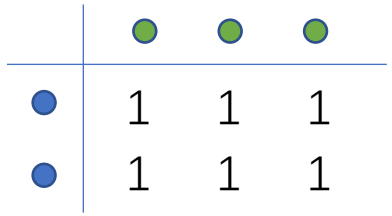
$$f(\mathcal{S}) = f_{\mathcal{V}}(\mathcal{S}) + f_{\mathcal{E}}(\mathcal{S}) = \sum_{i \in \mathcal{S}} a_i + \sum_{i,j \in \mathcal{S} \wedge (i,j) \in \mathcal{E}} c_{ij}, \quad (2)$$

点异常

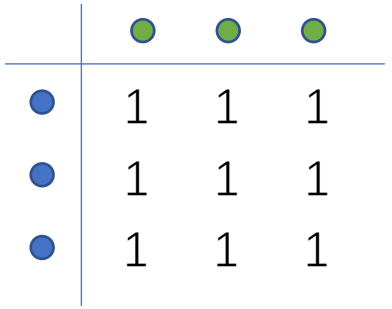
边异常：比如i购买j商品



公理1:：高危节点集合比低危节点集合更可疑
公理2：集合内增加边，将会增加集合可疑性
公理3：当点和边的权重相同时（并且在同样边的密度下），大图比小图更可疑
公理4：如果总的可疑度相同，小图比大图更可疑（密度大）



$$g(S) = \frac{5+6}{5} = 2.2$$



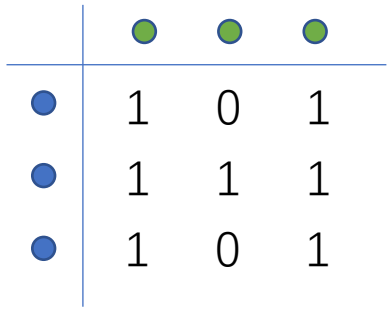
$$g(S) = \frac{6+9}{6} = 2.5$$

算法流程

1. 初始的二部图，包含所有的节点和边，计算初始密度，设置其为最优密度
2. 采用贪心方法，去掉一个可疑度最小的节点，重新计算密度
3. 如果该密度大于最优密度，则将该密度赋值为最优密度，并保留剩余子图。（如果小于，也保留剩余节点）
4. 重复执行2,3，当行或者列中某个集合为空时，停止循环。
5. 输出最优集合和密度

$$g(S) = \frac{f(S)}{|S|} \quad (1)$$

$$g'(S) = \frac{f(S) - f(i)}{|S| - 1}$$



$$f(S) = f_V(S) + f_E(S) = \sum_{i \in S} a_i + \sum_{i,j \in S \wedge (i,j) \in E} c_{ij}, \quad (2)$$

蓝1	3
蓝2	4
蓝3	3

绿1	4
绿2	2
绿3	4

1. 计算初始最优密度 $g(S) = \frac{6+7}{6} = 2.16$
2. 计算一个影响最小的节点，绿2影响最小，去掉后重新计算密度 $g'(S) = \frac{5+6}{6-1} = 2.2$
3. 新图密度大于最优密度，并同时保留子图和最新密度
4. 迭代执行

优化方法：采用贪心优化过程，不断的去掉节点，使得剩余子图具有更高的异常值

Algorithm 1 FRAUDAR, which greedily removes nodes to maximize a metric g. Line 5 and 6 run in $O(\log |\mathcal{V}|)$ time, using a data structure described in Section 4.2.

Require: Bipartite $G = (\mathcal{U} \cup \mathcal{W}, \mathcal{E})$; density metric g of the form in (1)

1: **procedure** FRAUDAR (G, g)
2: Construct priority tree T from $\mathcal{U} \cup \mathcal{W}$ 对所有节点构建优先树
3: $\mathcal{X}_0 \leftarrow \mathcal{U} \cup \mathcal{W}$
4: **for** $t = 1, \ldots, (m+n)$ **do** 去掉节点i后的最到异常团伙
5: $i^* \leftarrow \arg\max_{i \in \mathcal{X}_i} g(\mathcal{X}_i \setminus \{i\})$
6: Update priorities in T for all neighbors of i^* 更新优先树
7: $\mathcal{X}_t \leftarrow \mathcal{X}_{t-1} \setminus \{i^*\}$ 去除节点
8: **end for**
9: **return** $\arg\max_{\mathcal{X}_i \in \{\mathcal{X}_0, \ldots, \mathcal{X}_{m+n}\}} g(\mathcal{X}_i)$
10: **end procedure**

$$g(\mathcal{S}) = \frac{f(\mathcal{S})}{|\mathcal{S}|} \quad (1)$$

▷ see Section 4.2

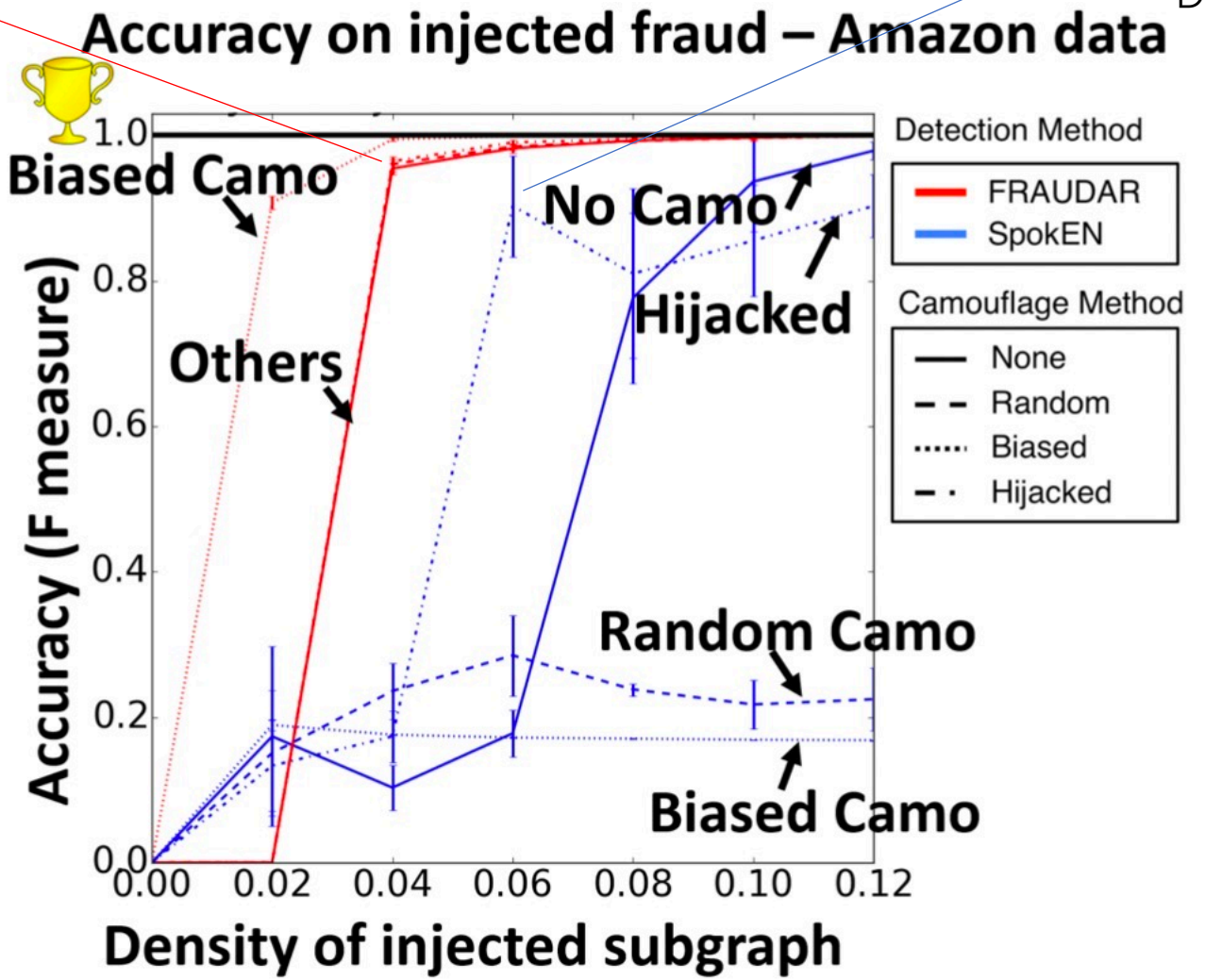
▷ suspicious set is initially the entire set of nodes $\mathcal{U} \cup \mathcal{W}$

▷ exonerate least suspicious node

▷ return most suspicious set $\mathcal{X}_i$

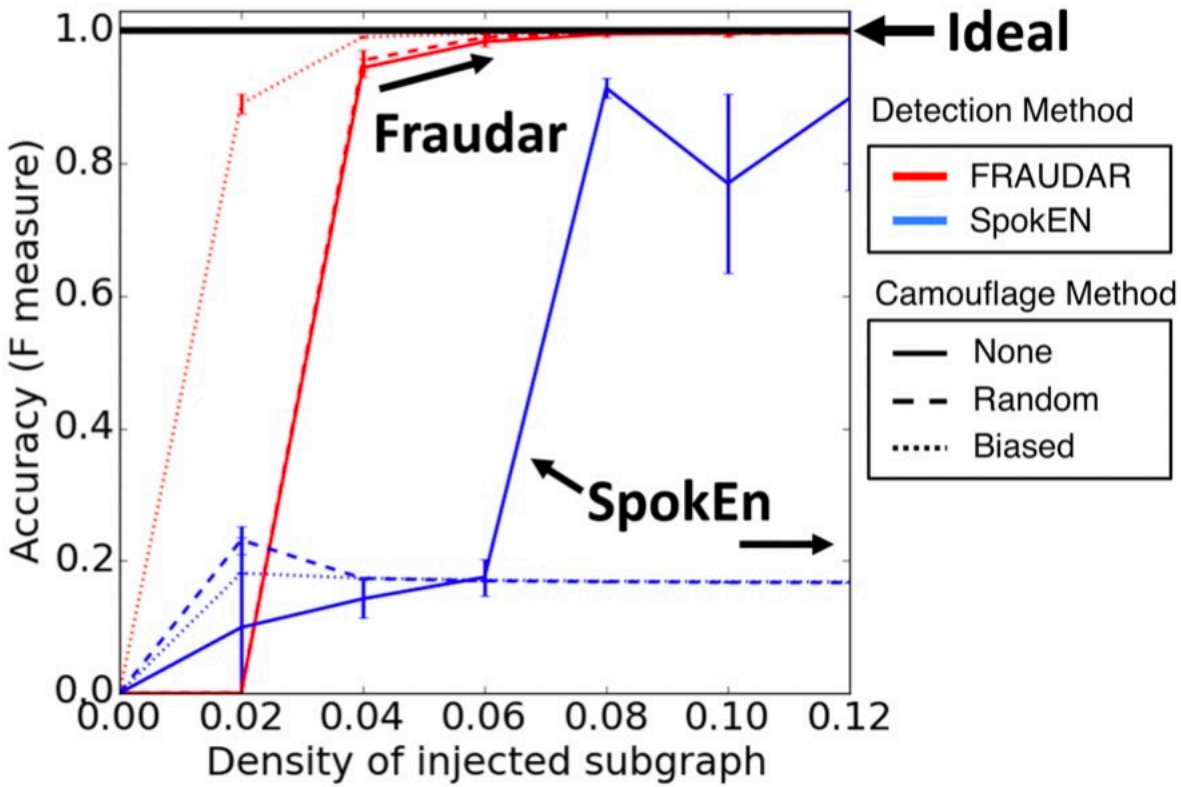
Density>=0.4下，所有场景下，F达到0.95

没有伪装下SpokEN的效果，
Density>=0.06，最大才能达到0.9

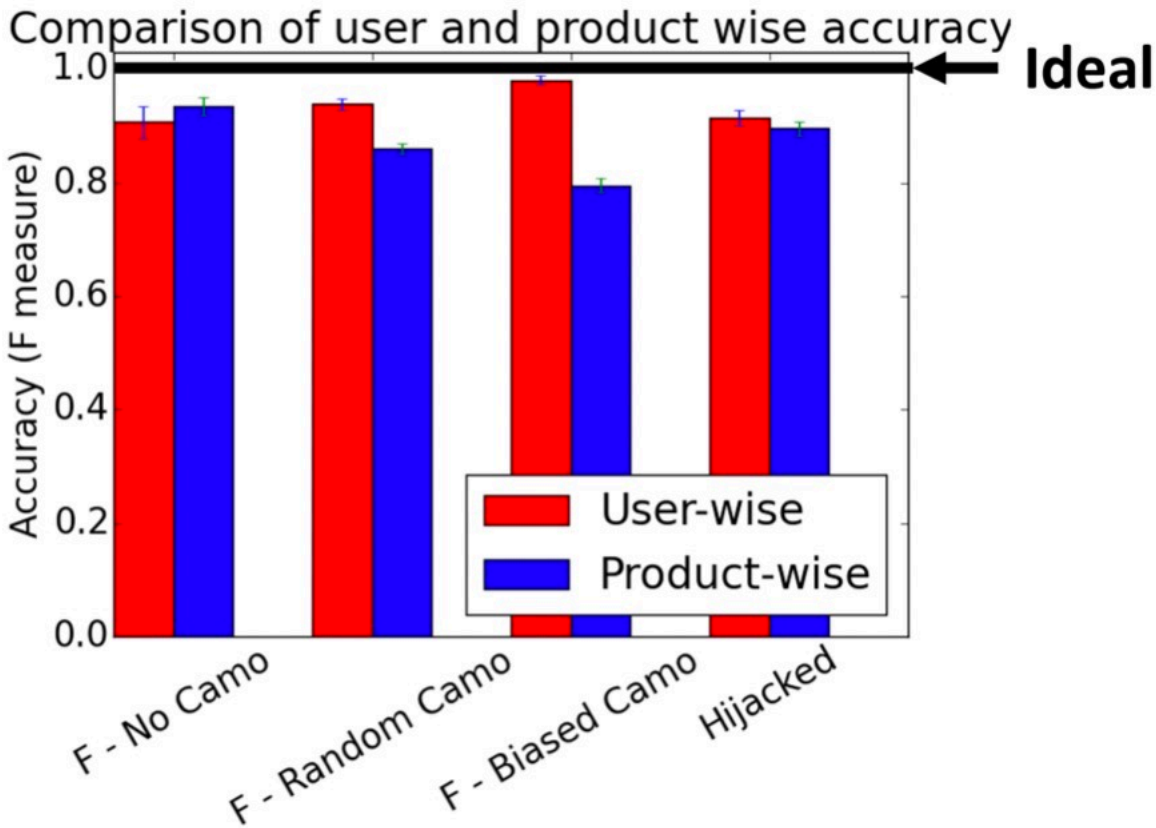


(b) FRAUDAR outperforms competitors

假设正常用户会有边添加到欺诈用户（反伪装）

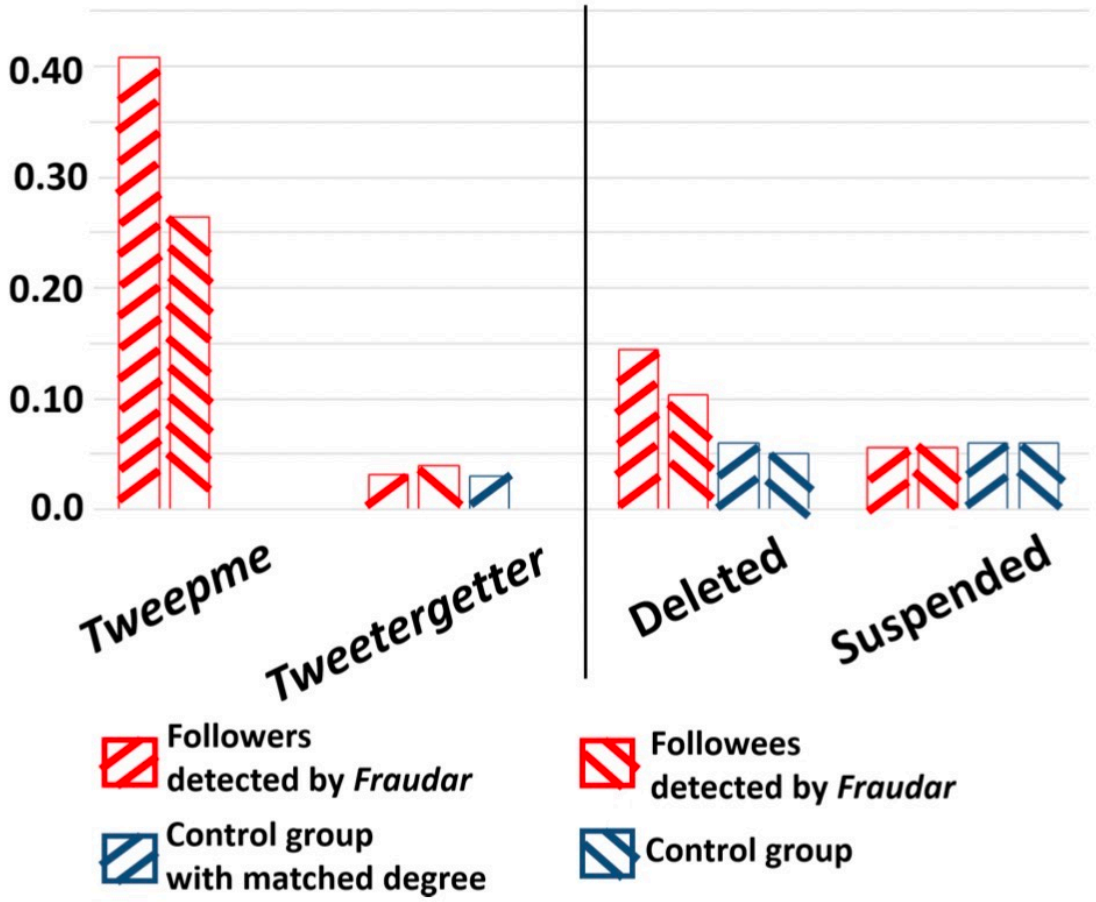


Fraudar效果依旧很好，
但是SpokEn有下降

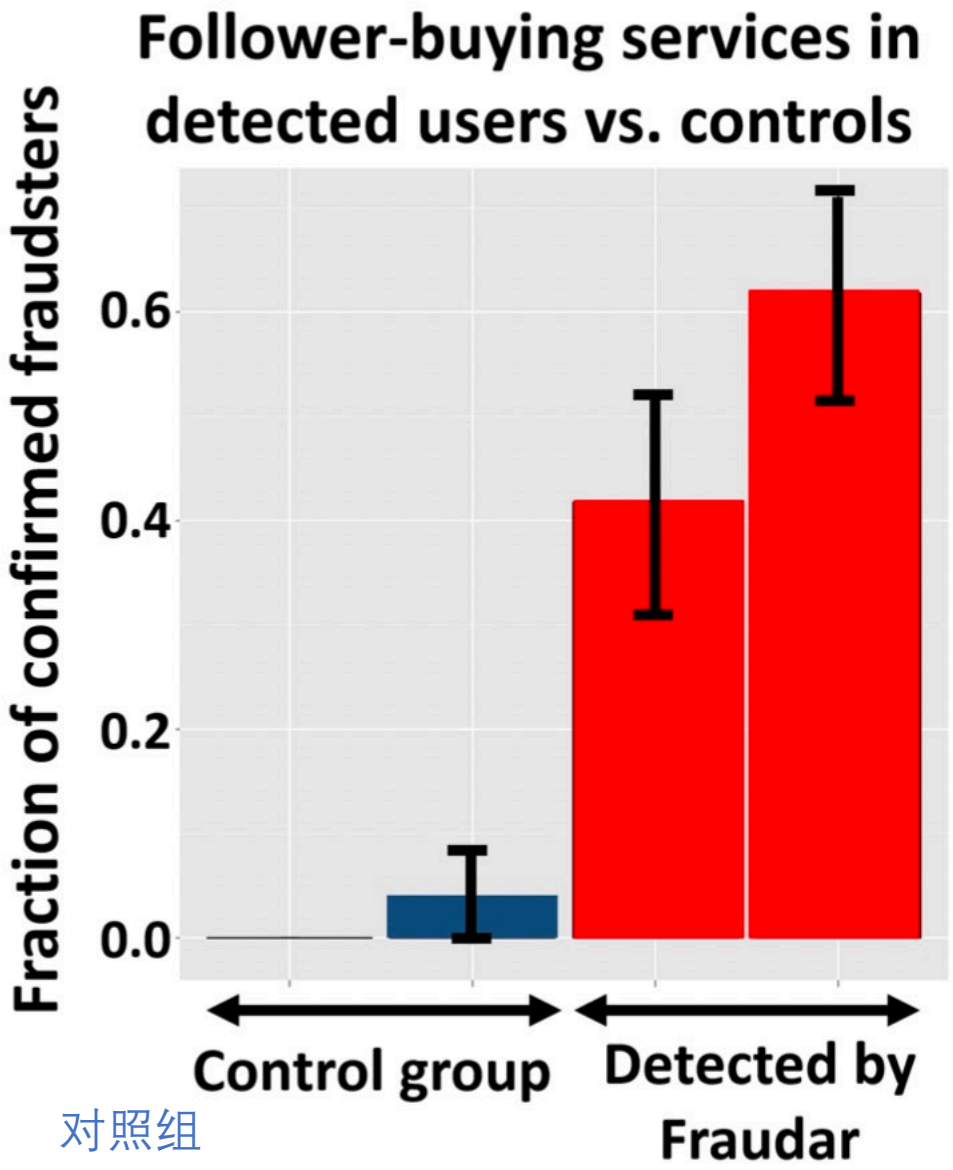


Fraudar在识别User和Product上的效果

Follower：刷单者
Followee：受益账户

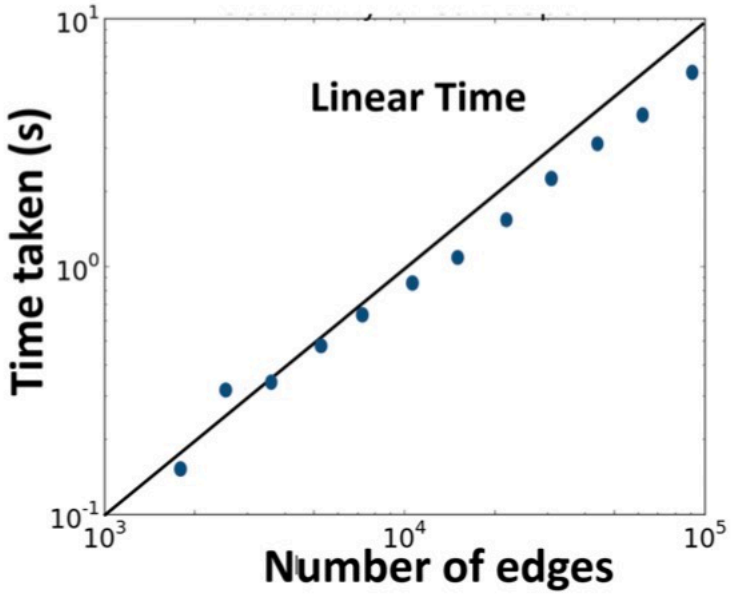


Tweepme和Tweetergetter：粉丝购买软件



对照组

(c) FRAUDAR detects many confirmed fraudsters


Linear Time
Time taken (s)
Number of edges
10^{1}
10^{0}
10^{-1}
10^{3}
10^{4}
10^{5}

时间复杂度呈线性关系

```
def logWeightedAveDegree(M, nodeSusp=None):  M:   (0, 0)\t1\n  (0, 1)\t1\n
    (m, n) = M.shape  m: 500  n: 500
    colSums = M.sum(axis=0)  # 对列进行求和  colSums: Loading timed out
    colWeights = np.squeeze(np.array(1.0 / np.log(np.squeeze(colSums) + 5)))
    colDiag = sparse.lil_matrix((n, n))  # 建立稀疏矩阵  colDiag: Loading time
    colDiag.setdiag(colWeights)  # 对角阵
    W = M * colDiag  # 将原始邻接矩阵通过colWeight赋予权重
    print("finished computing weight matrix")
    return fastGreedyDecreasing(W, colWeights, nodeSusp)  # 建立贪心算法
```

$$h(x) = \frac{1}{\log(c + d_j)}$$

```
colDiag = sparse.lil_matrix((3, 3))  # 建立稀疏矩阵
colDiag.setdiag(np.array([0.1,0.2,0.3]))
```

```
array([[0., 0., 0.],        [[0.1 0.  0. ]
       [0., 0., 0.],         [0.  0.2 0. ]
       [0., 0., 0.]])        [0.  0.  0.3]]
```

Code

```
def c2Score(M, rowSet, colSet, nodeSusp):
    suspTotal = nodeSusp[0][list(rowSet)].sum() + nodeSusp[1][list(colSet)].sum()  # 计算行列的可疑节点的异常值
    return M[list(rowSet),:][:,list(colSet)].sum(axis=None) + suspTotal
```

计算异常分值

$$f(\mathcal{S}) = f_{\mathcal{V}}(\mathcal{S}) + f_{\mathcal{E}}(\mathcal{S}) = \sum_{i \in \mathcal{S}} a_i + \sum_{i,j \in \mathcal{S} \wedge (i,j) \in \mathcal{E}} c_{ij}, \quad (2)$$

```
bestAveScore = curScore / (len(rowSet) + len(colSet))  # 计算异常密度分值
```

$$g(\mathcal{S}) = \frac{f(\mathcal{S})}{|\mathcal{S}|} \quad (1)$$

Algorithm 1 FRAUDAR, which greedily removes nodes to maximize a metric g. Line 5 and 6 run in $O(\log |\mathcal{V}|)$ time, using a data structure described in Section 4.2.

Require: Bipartite $G = (\mathcal{U} \cup \mathcal{W}, \mathcal{E})$; density metric g of the form in (1)

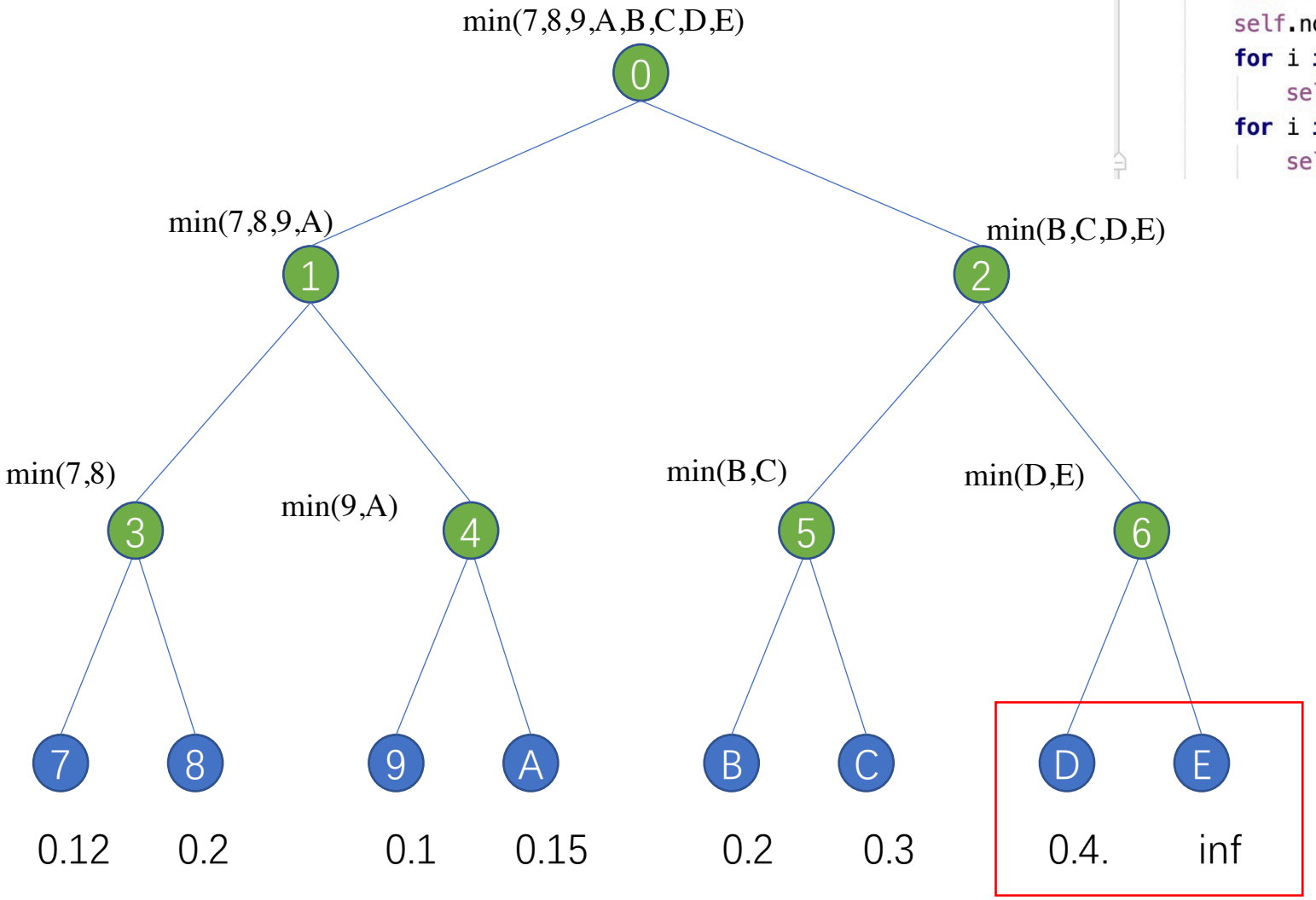
1: **procedure** FRAUDAR (G, g)
2: Construct priority tree T from $\mathcal{U} \cup \mathcal{W}$ ▷ see Section 4.2
3: $\mathcal{X}_0 \leftarrow \mathcal{U} \cup \mathcal{W}$ ▷ suspicious set is initially the entire set of nodes $\mathcal{U} \cup \mathcal{W}$
4: **for** $t = 1, \ldots, (m + n)$ **do**
5: $i^* \leftarrow \arg\max_{i \in \mathcal{X}_i} g(\mathcal{X}_i \setminus \{i\})$ ▷ exonerate least suspicious node
6: Update priorities in T for all neighbors of i^*
7: $\mathcal{X}_t \leftarrow \mathcal{X}_{t-1} \setminus \{i^*\}$
8: **end for**
9: **return** $\arg\max_{\mathcal{X}_i \in \{\mathcal{X}_0, \ldots, \mathcal{X}_{m+n}\}} g(\mathcal{X}_i)$ ▷ return most suspicious set $\mathcal{X}_i$
10: **end procedure**

```
rowTree = MinTree(rowDeltas)
colTree = MinTree(colDeltas)
```

有7个数需要比较：

degree = [0.12, 0.2, 0.1, 0.15, 0.2, 0.3, 0.4]

```python
class MinTree:
    def __init__(self, degrees):
        self.height = int(math.ceil(math.log(len(degrees), 2)))  # 建立树的深度
        self.numLeaves = 2 ** self.height  # 叶子数量
        self.numBranches = self.numLeaves - 1  # 分支数量(整个树除去叶子后的节点数
        self.n = self.numBranches + self.numLeaves  # 树的总节点数量
        self.nodes = [float('inf')] * self.n  # 节点初始值
        for i in range(len(degrees)):
            self.nodes[self.numBranches + i] = degrees[i]  # 将degree值赋值到[s
        for i in reversed(list(range(self.numBranches))):  # 索引510开始的
            self.nodes[i] = min(self.nodes[2 * i + 1], self.nodes[2 * i + 2])
```



degree = 7

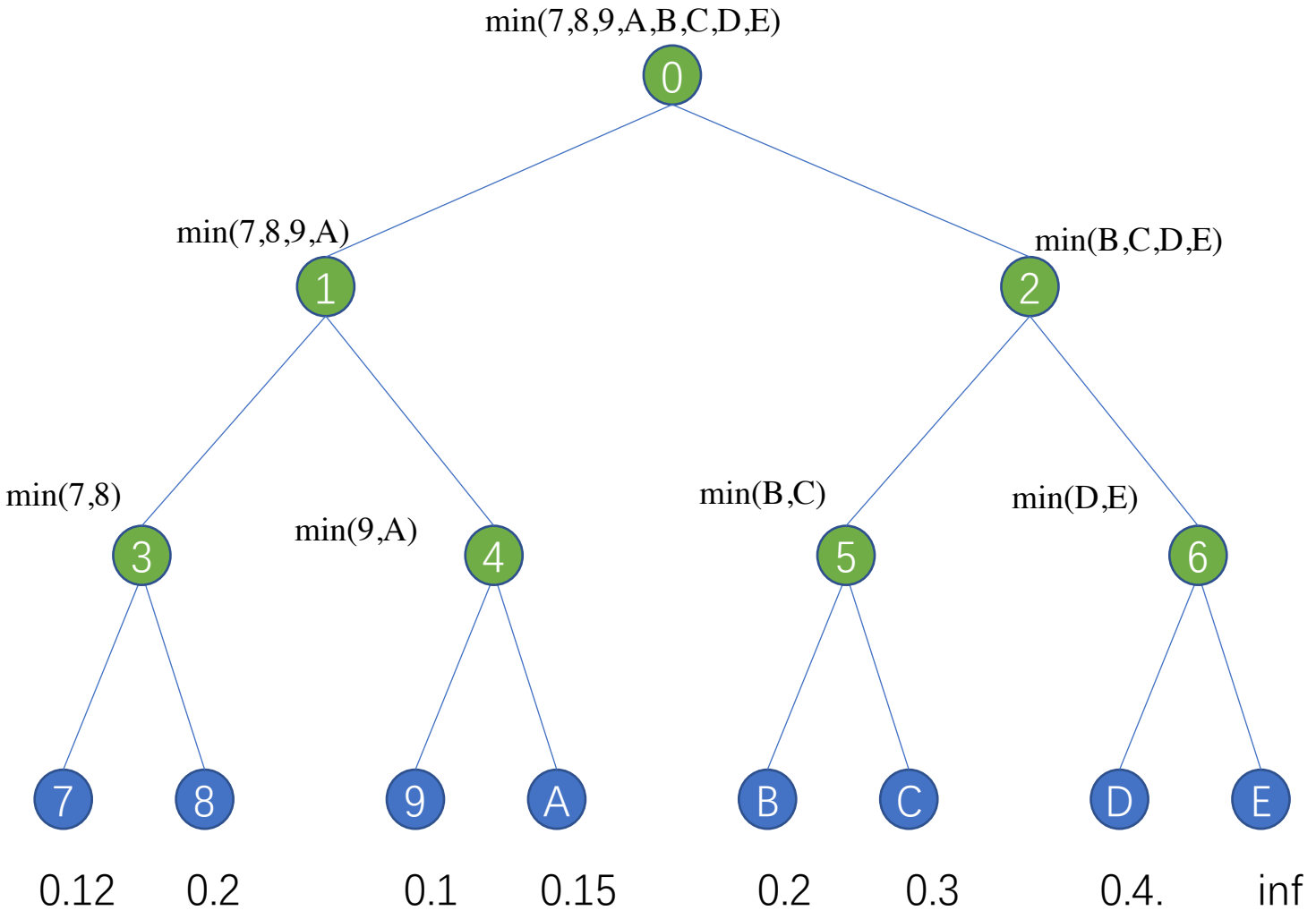
height = 3

self.numLeaves = 8

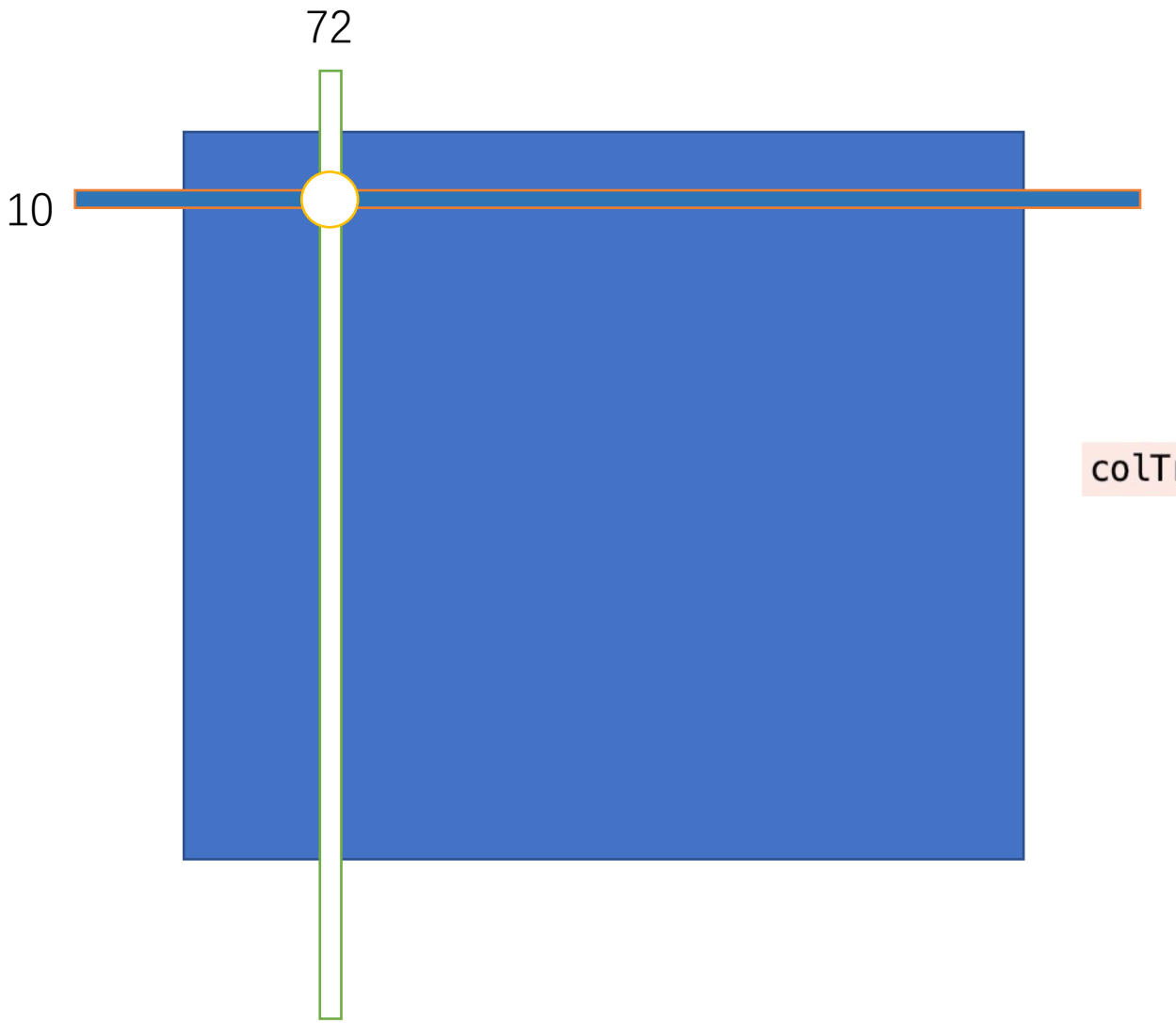
self.numBranches = 7

i = 6

[2 * i + 1], [2 * i + 2] = 13, 14




```
def getMin(self):  self: <MinTree.MinTree object at 0x7fb541762748>
    cur = 0  cur: 0
    for i in range(self.height):  i: 0
        cur = (2 * cur + 1) if self.nodes[2 * cur + 1] <= self.nodes[2 * cur + 2] else (2 * cur + 2)
    # print "found min at %d: %d" % (cur, self.nodes[cur])
    return (cur - self.numBranches, self.nodes[cur])
```

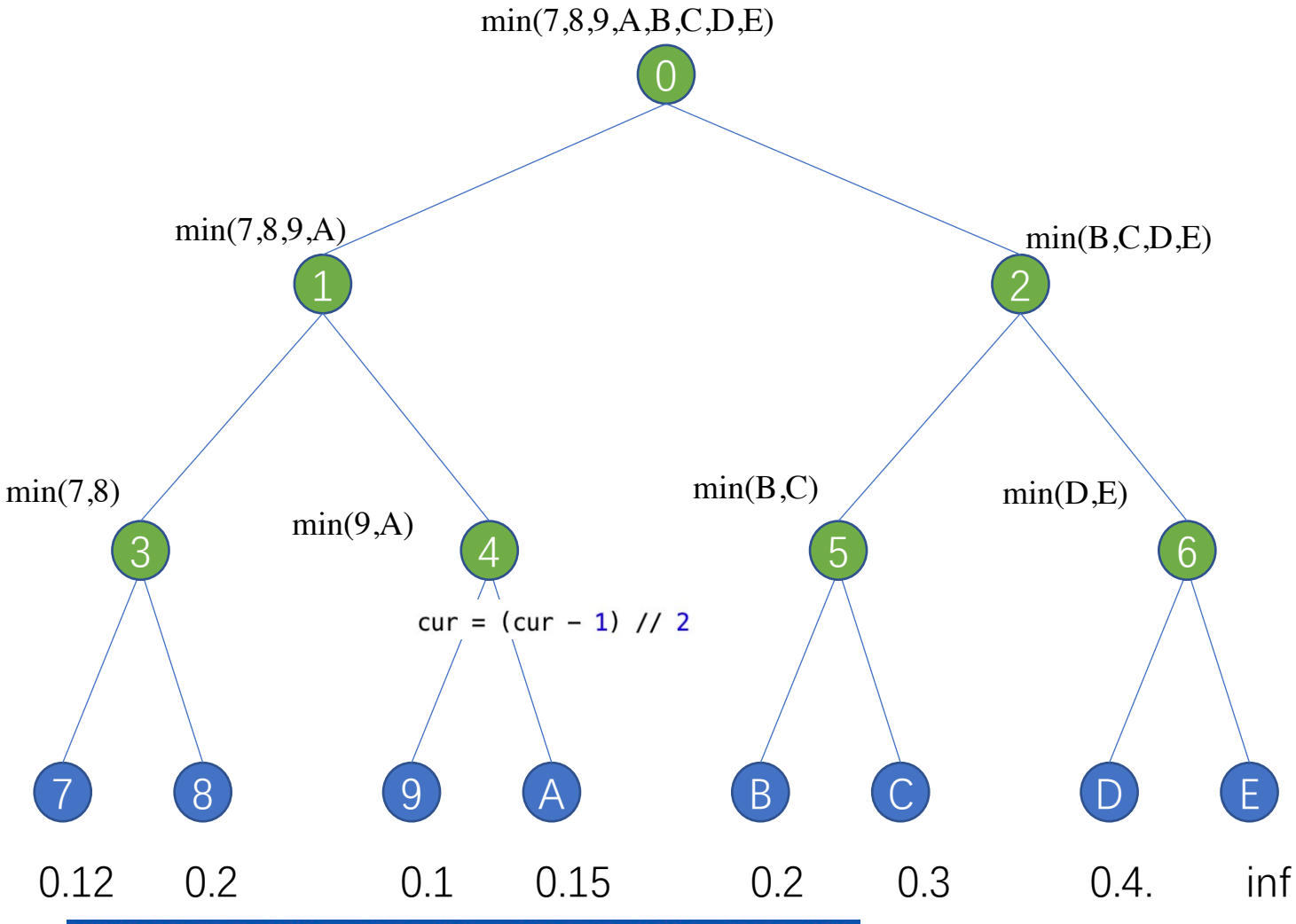


更改行和的值

```
rowTree.changeVal(i, -colWeights[nextCol])
```

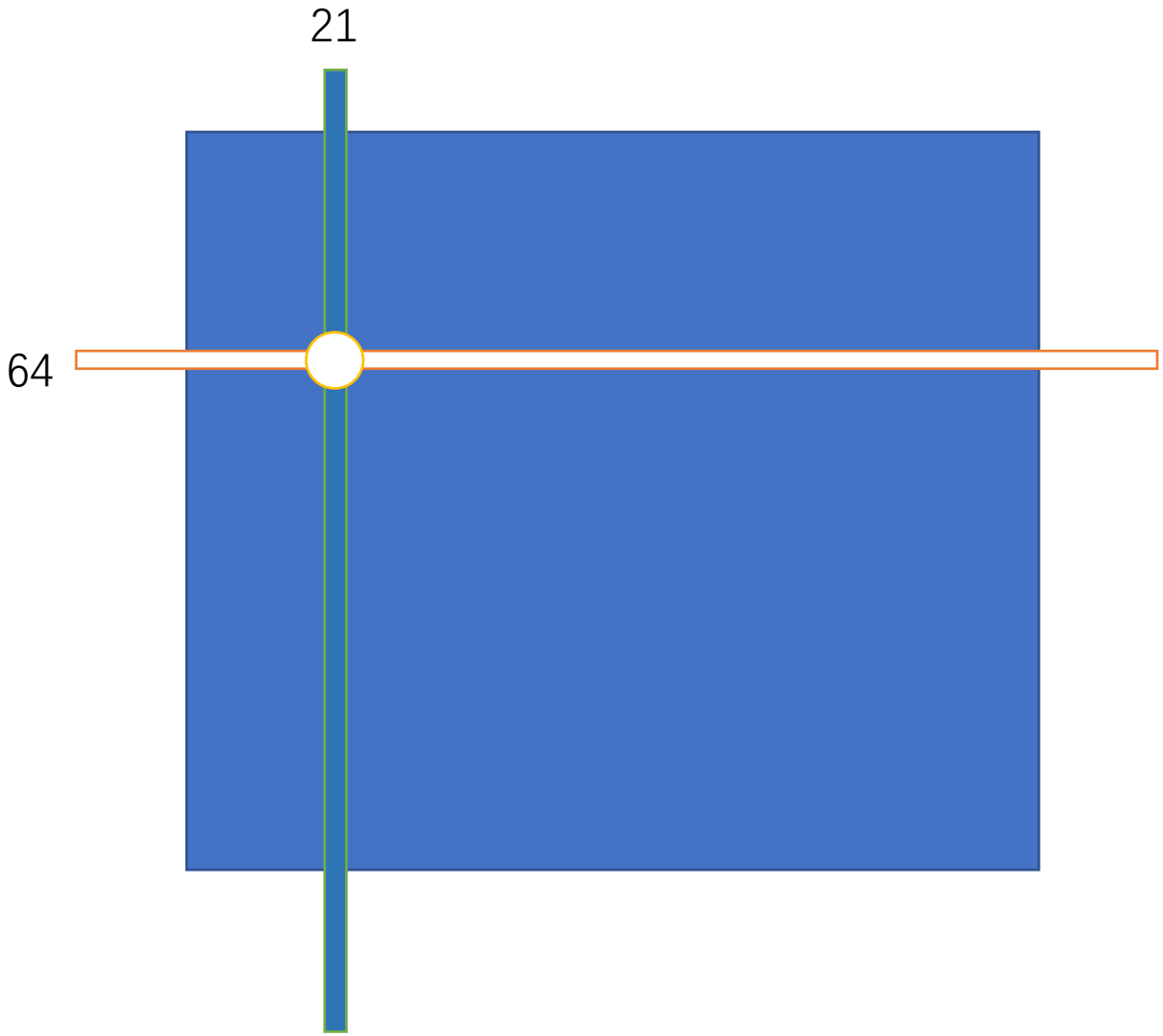
将该列的值赋值为inf

```
colTree.changeVal(nextCol, float('inf'))  # 将该节点值赋值为inf
```



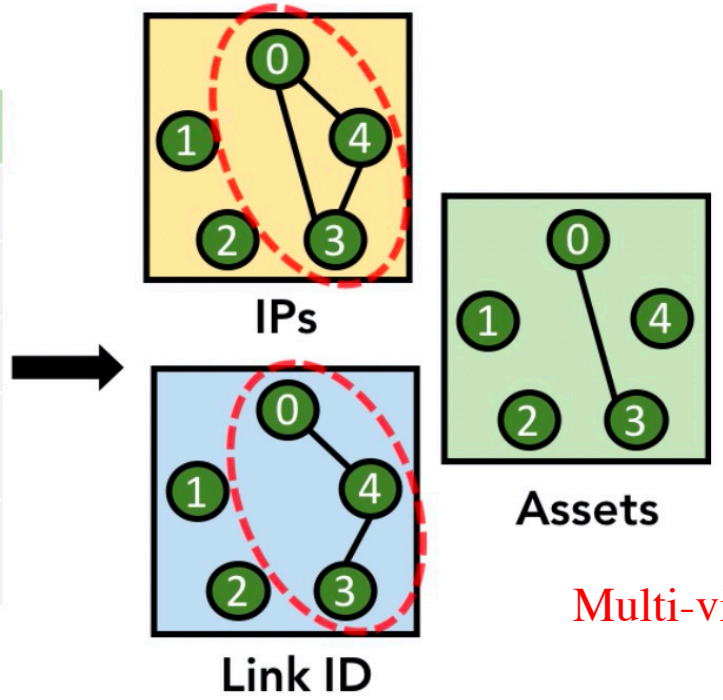

```
min(self.nodes[2 * cur + 1], self.nodes[2 * cur + 2])
```

```
def changeVal(self, idx, delta):  self: <MinTree.MinTree object at 0x7fb542
    cur = self.numBranches + idx  # 对应到的节点
    self.nodes[cur] += delta
    for i in range(self.height):
        cur = (cur - 1) // 2
        nextParent = min(self.nodes[2 * cur + 1], self.nodes[2 * cur + 2])
        if self.nodes[cur] == nextParent:
            break
        self.nodes[cur] = nextParent  # 赋值新的最小值
```


21
64

SLICENDICE: Mining Suspicious Multi-attribute Entity Groups with Multi-view Graphs

ID	IPs	Link ID	Assets
0	{1.2.3.4}	{b1c45}	{Cheap-Iphone.jpg}
1	{103.71.11.5}	{ytnw71}	{Smoothie.jpg}
2	{201.27.18.6}	{1m572d}	{main.jpg}
3	{112.11,16.1, 1.2.3.4}	{a6wu7}	{Promotion-1.jpg, Cheap-Iphone.jpg}
4	{1.2.3.4}	{a6wu7, b1c45}	{Cheap-Rolex.jpg}

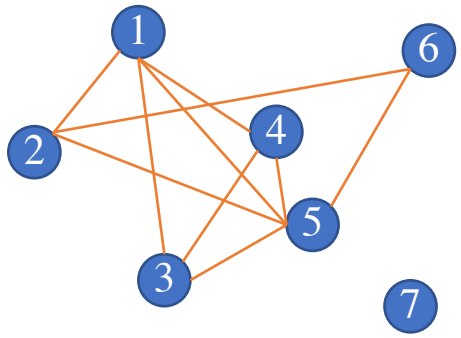


Multi-view Graphs

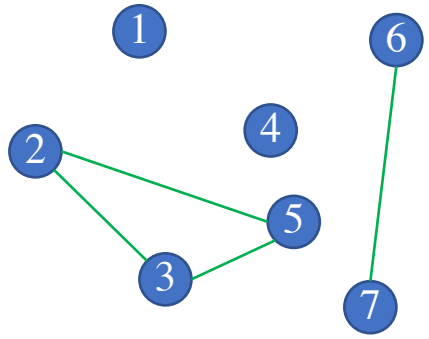
(a) Our formulation: multi-attributed entity data as a multi-view graph

Multi-attribute: 节点有多个属性（IPs, Link ID, Assets）

{"entity_id": "1", "country": "Libya,Germany", "IP":"1.2.3.1"}
{"entity_id": "2", "country": "Libya,USA", "IP":"1.2.3.4"}
{"entity_id": "3", "country": "Germany", "IP":"1.2.3.4"}
{"entity_id": "4", "country": "Germany", "IP":"1.2.3.2"}
{"entity_id": "5", "country": "USA,Germany", "IP":"1.2.3.4"}
{"entity_id": "6", "country": "USA", "IP":"1.2.3.5"}
{"entity_id": "7", "country": "Iran", "IP":"1.2.3.5"}



Country



IP

衡量单个view上的异常性

$mass\ \text{of}\ \mathcal{X}_i\ \text{as}\ c_i = \sum_{(a,b)\in\mathcal{X}^2} w_i^{(a,b)}$

质量

$volume\ \text{of}\ \mathcal{X}_i = n(n-1)/2$

体积

节点之间全连接对应到的边数量

$\rho_i = c_i / v_i$

密度

$m_{country} = 10$

$m_{IP} = 4$

$\omega_{country}^{(1,3)}$: 1,3节点连接的边权重

$\omega_{country}^{(2,3)}$: 2,3节点连接的边权重

$$v_{country} = 7 * \frac{7-1}{2} = 21$$

$$v_{IP} = 7 * \frac{7-1}{2} = 21$$

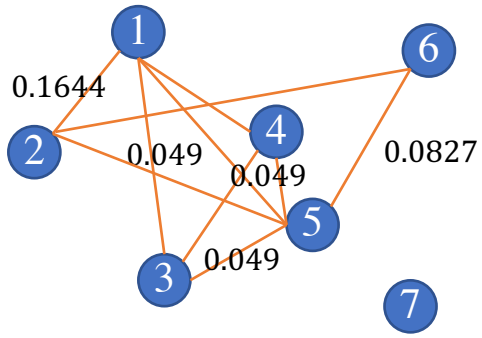
因为总节点数量相同，所以所有view中的体积均相同

$\rho_{country} = 10/21 = 0.476$

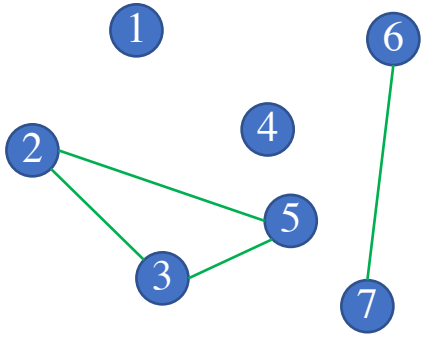
$\rho_{IP} = 4/21 = 0.1905$

根据以上的结果可以看出，Country的集合比较密集，虽然Country，但是实际上真的是这样吗？

{"entity_id": "1", "country": "Libya,Germany", "IP":"1.2.3.1"}
{"entity_id": "2", "country": "Libya,USA", "IP":"1.2.3.4"}
{"entity_id": "3", "country": "Germany", "IP":"1.2.3.4"}
{"entity_id": "4", "country": "Germany", "IP":"1.2.3.2"}
{"entity_id": "5", "country": "USA,Germany", "IP":"1.2.3.4"}
{"entity_id": "6", "country": "USA", "IP":"1.2.3.5"}
{"entity_id": "7", "country": "Iran", "IP":"1.2.3.5"}



Country



IP

Country聚集的风险度，要明显小于IP属性的聚集，所以我们要给予不同属性图以不同边的权重

Country: {'Germany': 4, 'USA': 3, 'Libya': 2, 'Iran': 1}

计算idf值

paper

$$ief(v_i) = \left(N / \log\left(1 + |\mathcal{A}_i^{-1}(v_i)|\right)\right)^2$$

'Germany' $= (10/\log(1+\frac{1}{4}))^2 = 2008$

'USA': $= (10/\log(1+\frac{1}{3}))^2 = 1208$

'Libya': $= (10/\log(1+\frac{1}{2}))^2 = 608$

'Iran': $= (10/\log(1+\frac{1}{1}))^2 = 208$

Code $ief(v_i) = \left(\log\left(1 + |A_i^{-1}(v_i)|\right)\right) ** 2$

'Germany' $= (\log(1+\frac{1}{4}))^2 = 0.049$

' USA ': $= (\log(1+\frac{1}{3}))^2 = 0.0827$

' Libya': $= (\log(1+\frac{1}{2}))^2 = 0.1644$

'Iran': $= (\log(1+\frac{1}{1}))^2 = 0.4805$

IDF

$$\mathrm{idf_i} = \lg \frac{|D|}{|\{j : t_i \in d_j\}|}$$

总文件数量

包含该词文件数

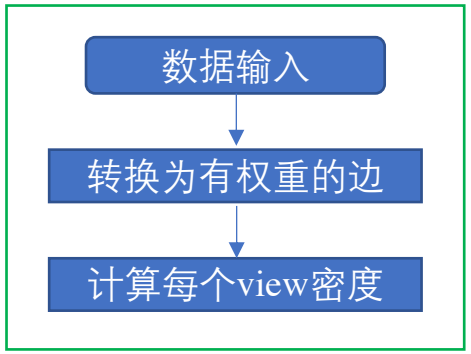
'Germany' $= \log(10/4) = 0.916$

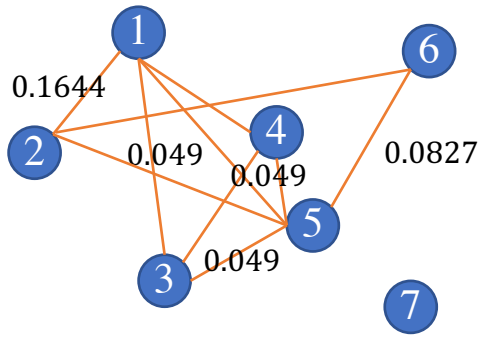
'USA': $= \log(10/3) = 1.203$

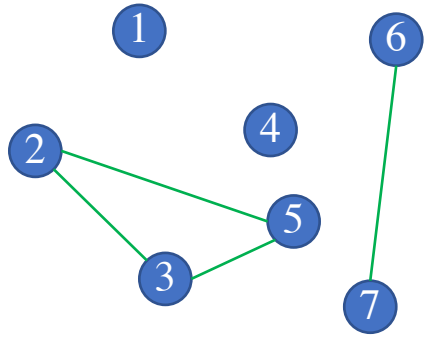
'Libya': $= \log(10/2) = 1.609$

'Iran': $= \log(10/1) = 2.303$

算法流程







数据预处理工作

SliceNDice算法也是使用贪心法来筛选异常群组，但不同与Fraudar，该算法是先选择种子节点，再不断完善子图

我们在选择种子节点之前，要先确定选择哪些视图。

View Choose

每个view中涉及到的属性的99.5%分位数对应值的倒数，进行概率采样

Country: {'Germany': 40, 'USA': 30, 'Libya': 20, 'Iran': 10}
IP: {'1.2.3.4': 10, '1.2.3.5': 10, '1.3.4.5':8 ,,, '1.3.4.6':5 ,,, '1.8.8.8':1}

99.5%百分位数: 39.85
99.5%百分位数:10

Sample proba:
0.025
0.1

说明了，越不容易聚集的属性，被当作候选view的概率越大（这些view的聚集更能反映异常）

比如我们有图中有10种属性，我们只选择其中5个属性进行异常聚类

Entity Choose

这个视图中的所有连接节点

{'1.2.3.4': {2, 3, 5}, '1.2.3.5': {6, 7}}

随机选择一个候选view — IP

随机选择两个相连接的节点 — {2, 3, 5} -> {3, 5}　因为两个节点连接，则说明他们密度一定大于0，要比随机选择两个节点密度期望更大

计算两个节点子块的密度 — {3, 5}子图在所有view下密度(Country, IP)

每个view全体节点的密度

遍历每个view

子图密度>全图密度

No

遍历每个不满足要求view

该view连接节点 — {'Libya': {1, 2}, 'Germany': {1, 3, 4, 5}, 'USA': {2, 5, 6}}

遍历候选节点{3, 5}

节点 in 连接集合

No

重新Entity Choose

{3}, {5}是否在{1, 2}, {1, 3, 4, 5}, {2, 5, 6}中

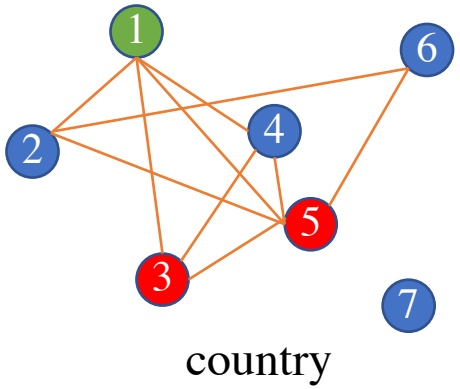
Yes

随机添加一个节点 — {1}

{3, 5, 1}

Yes

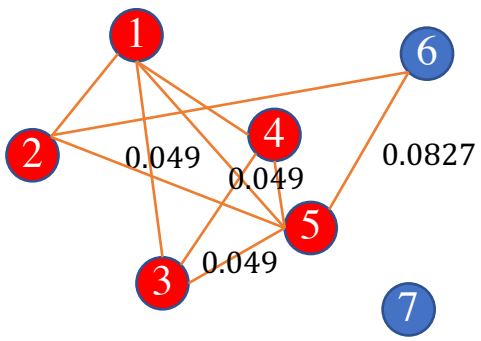
最终Entity Set



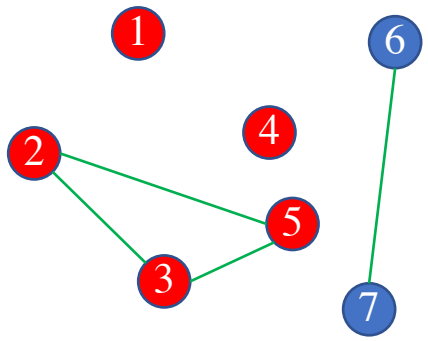
已经得到了选择的view和Entity Set

因为在该view上不满足密度要求，那么在添加一个连接的边的节点，则大概率会增大密度，以达到密度要求

假如我们现在已经找到了密集子图的节点集合和对应到的view，那么我们如何估计这个多属性子图的可疑性呢？



Country



IP

一种比较直观的想法是，直接求各个view上的密度的和或者均值，来表示多个属性子图的可疑度。
但是，这里作者采用了伽马分布的似然函数当作可疑度的衡量。

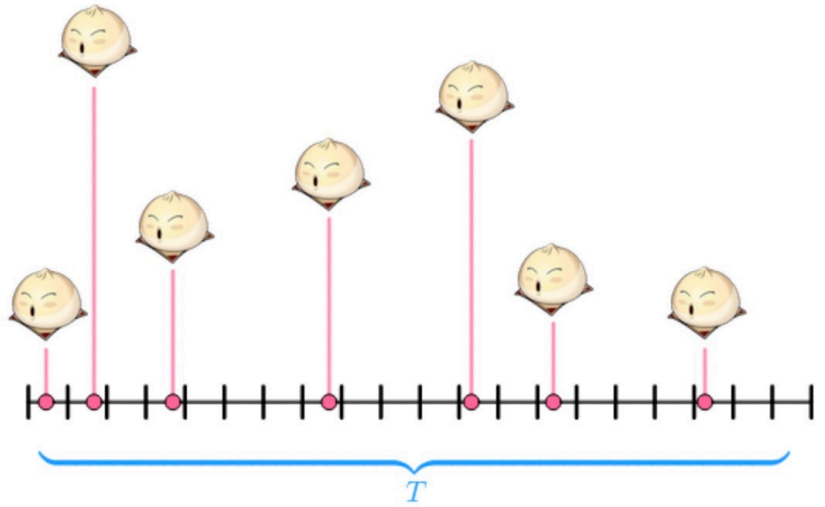
接下来，我们先简单介绍下几个常见的分布：

贝努利实验：投硬币，正面反面概率分别为p，1-p。
二项分布：n次贝努利实验，正面出现的次数。
泊松分布：特定时间内，发生n个事件的概率。一个小时内，卖出馒头的数量为10,20,30……个的概率。
指数分布：指数分布表示两次事件发生间隔为t的概率分布。1,2,5,10分钟内，没有卖出馒头的概率。
伽玛分布：如果指定事件出现N次，需要等待的时间。卖出10个馒头，需要的一小时时间。

泊松分布

某一天卖出的馒头数量，T表示是这一天的时间

	销售
周一	3
周二	7
周三	4
周四	6
周五	5



周二

$$\overline{X} = \frac{3+7+4+6+5}{5} = 5$$

现在是将T时间分割成20份，但是如果卖出的馒头数量超过20个，则这个划分就不够了，所以将T划分成无限大份。

$p = \frac{\mu}{n}$ μ表示平均T时间内卖出的馒头数量，n表示将时间T划分成n份，那么$\frac{\mu}{n}$则表示了每份时间中，卖出的概率

k表示T时间内卖出的馒头数量

$$\lim_{n\to\infty}\binom{n}{k}p^k(1-p)^{n-k} = \lim_{n\to\infty}\binom{n}{k}\left(\frac{\mu}{n}\right)^k(1-\frac{\mu}{n})^{n-k}$$

$$\lim_{n\to\infty}\binom{n}{k}\left(\frac{\mu}{n}\right)^k(1-\frac{\mu}{n})^{n-k} = \frac{\mu^k}{k!}e^{-\mu}$$

λ: T时间内平均卖出的馒头数量

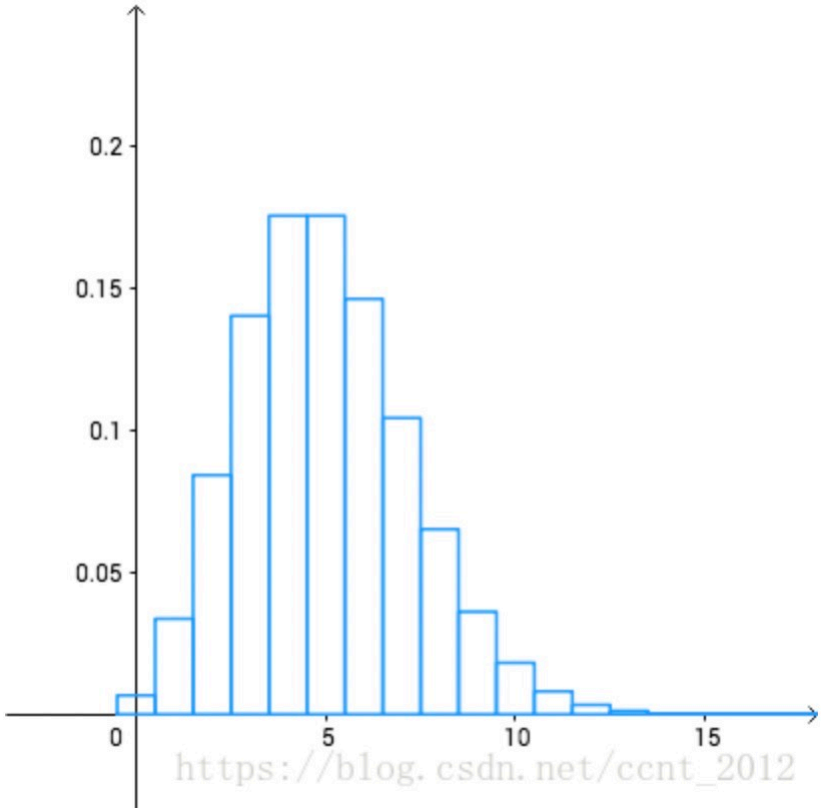
$$P(X=k) = \frac{\lambda^k}{k!}e^{-\lambda}$$

泊松分布的最终概率分布函数

$$P(X=k)=\frac{\lambda^k}{k!}e^{-\lambda}$$

根据公式计算出来的在T时间内卖出的馒头数量

$$\bar{x}=\lambda=5$$



https://blog.csdn.net/ccnt_2012

$$P(X=1)=\frac{\lambda^k}{k!}e^{-\lambda}=\frac{5^1}{1!}e^{-5}=0.034$$

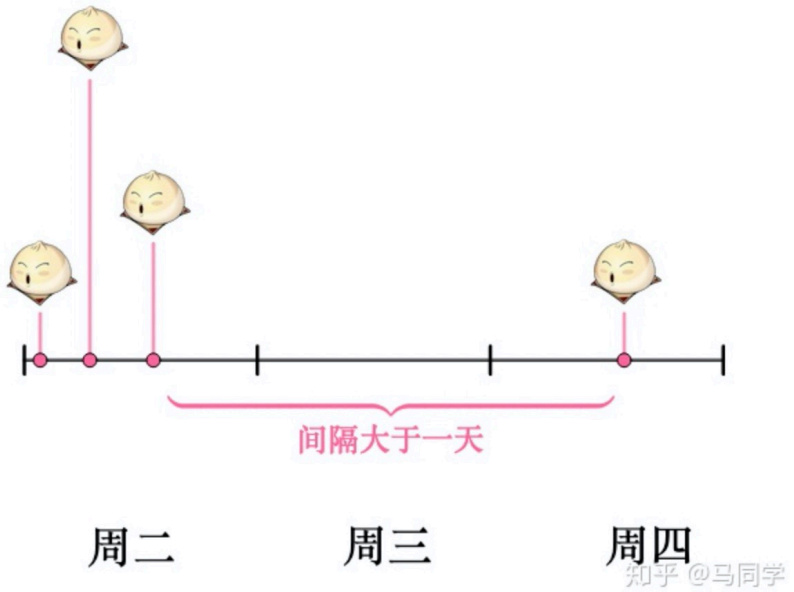
$$P(X=2)=\frac{\lambda^k}{k!}e^{-\lambda}=\frac{5^2}{2!}e^{-5}=0.084$$

0 : 0.007
1 : 0.034
2 : 0.084
3 : 0.14
4 : 0.175
5 : 0.175
6 : 0.146
7 : 0.104
8 : 0.065
9 : 0.036

https://blog.csdn.net/ccnt_2012/article/details/81114920

指数分布

之前我们已将讨论了每天卖出馒头的数量，即泊松分布。
接下来，我们讨论馒头和馒头卖出的时间间隔问题。



假如周三没卖出馒头，这意味着，周二最后卖出的馒头，和周四最早卖出的馒头中间间隔了一天

某一天没有卖出馒头，根据泊松分布，可得： $P(X=0)=\frac{\lambda^0}{0!}e^{-\lambda}=e^{-\lambda}$

$Y=$ 卖出两个馒头之间的时间间隔为1天　$P(Y>1)=P(X=0)=e^{-\lambda}$

59.6小时发生一次事故$\lambda = 1/59.6$
一天内卖出5个馒头 $\lambda = 5/1$

一个设备出现多次故障的时间间隔记录如下：

23, 261, 87, 7, 120, 14, 62, 47, 225, 71, 246, 21, 42, 20, 5, 12, 120, 11, 3, 14, 71, 11, 14, 11, 16, 90, 1, 16, 52, 95

根据上面数据，我们可以计算得到该设备发生故障的平均时间是59.6小时，即单位小时时间内发生故障事件的次数为λ=1/59.6=0.0168。那么该设备在3天（72小时)内出现故障的概率是多大呢？即求P(x<72)，这就需要计算指数分布的累积分布函数：

$$p(y) = \begin{cases} \lambda e^{-\lambda y}, & y \geq 0 \\ 0, & y < 0 \end{cases} \qquad P(X < 72) = \int_0^{72} \lambda e^{-\lambda x} dx = 1 - e^{-\lambda(72)} = 1 - e^{-0.0168*72} = 0.7017$$

5*90分钟内进一个球$\lambda = 1/450$

也即该设备3天内出现故障的概率大于70%。

再举个国足的例子，假设国足参加了2018年世界杯，国足面对世界强队比赛，平均5场比赛进一个球，即单位分钟时间内发生进球事件的次数为λ=1/（5*90）=0.002222。那么小组赛3场比赛，至少能进一个球的概率是多少？

3场小组赛，3*90分钟=270分钟。

$$P(X < 270) = \int_0^{270} \lambda e^{-\lambda x} dx = 1 - e^{-\lambda(270)} = 1 - e^{-0.002222*270} = 0.4512$$

也就是说3场小组赛一个球不进的概率是1-0.4512=54.88%

https://blog.csdn.net/saltriver/article/details/53363888

伽玛分布

指数分布是描述泊松分布中事件发生时间间隔的概率分布

*意义来看：*指数分布解决的问题是 “要等到一个随机事件发生，需要经历多久时间”，
伽玛分布解决的问题是 “要等到n个随机事件都发生，需要经历多久时间”

X~Gamma(α,λ)，概率公式如下

随机事件发生α次

$$f_X(x) = \begin{cases} \frac{\lambda^{\alpha} x^{\alpha-1} e^{-\lambda x}}{\Gamma(\alpha)} & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

λ: 分布参数(一天内卖出100个馒头)

alpha代表上述的n, 当alpha=1时，就变成了指数分布：

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

C. Mining Suspicious Groups 群组异常分值

$$\rho_i = c_i / v_i$$

体积

$$\lambda_i = N(N-1)/(2C_i) = V/C_i = P_i^{-1}$$

质量

λ：密度的倒数

Lemma 1 (MVSG subgraph mass). *The mass M_i of a MVERE-distributed subgraph of $Exp(\lambda_i)$ follows $M_i \sim Gamma(v, P_i^{-1})$*

一小时卖出100个馒头：$\lambda = 100/1$

泊松分布：一小时卖出50个馒头的概率？

指数分布：30分钟卖出馒头的概率？

伽马分布：卖出200个馒头需要的时间为一小时概率？

$$M_i \sim Gamma(v, P_i^{-1})$$

质量为C下，体积为V $\lambda = \frac{V}{C}$

求在体积V下，该质量C的概率？（概率大，越真实）

$$f\left(n, \vec{c}, N, \vec{C}\right) = -\log\left(\prod_{i=1}^{K} Pr\left(M_i = c_i\right)\right)$$

loss函数

α: 随机事件发生次数(体积V)

$$f(x, \lambda, \alpha) = \frac{\lambda^{\alpha}\, x^{\alpha-1}\, e^{-\lambda x}}{\mathcal{T}(\alpha)}$$

伽马分布概率密度函数　λ: 单位时间内均值

c_i:子图mass

$$\sum_{i=1}^{K} -\log\left(\frac{(P_i^{-1})^{v}\, c_i^{v-1}\, e^{(-P_i^{-1})*c_i}}{\mathcal{T}(v)}\right)$$

$$f\left(x, P_i^{-1}, v\right) = \frac{(P_i^{-1})^{v}\, X^{v-1}\, e^{(-P_i^{-1})*X}}{\mathcal{T}(v)}$$

$$= \sum_{i=1}^{K} -v \log\left(\frac{V}{C_i}\right) + \boxed{\log \Gamma(v)} - (v-1)\log c_i + \frac{V c_i}{C_i}$$

C_i: 全图所有节点mass

极大似然估计，越大越好
如果值较小，表示异常。
现在负对数似然，越大，表示异常性越高

$$= \sum_{i=1}^{K} v \log \frac{C_i}{V} + \boxed{v \log v - v - \log v} - v \log c_i + \log c_i + \frac{V c_i}{C_i}$$

极大似然估计

若总体 X 属离散型，其分布律 $P\{X=x=p(x;\theta)$，$\theta \in \Theta$ 的形式已知，θ 为待估参数，Θ 是 θ 的可能取值范围。设 $X_1, X_2, \ldots, X_n$ 是来自 X 的样本，则 $X_1, X_2, \ldots, X_n$ 的联合概率分布为

$$\prod_{i=1}^{n} p(x_i;\theta)$$

设 $x_1, x_2, \ldots, x_n$ 是相应于样本 $X_1, X_2, \ldots, X_n$ 的一个样本值。则样本 $X_1, X_2, \ldots, X_n$ 取到观察值 $x_1, x_2, \ldots, x_n$ 的概率，也就是事件 $\{X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n\}$ 发生的概率为

$$L(\theta) = L(x_1, x_2, \ldots, x_n; \theta) = \prod_{i=1}^{n} p(x_i;\theta), \quad \theta \in \Theta$$

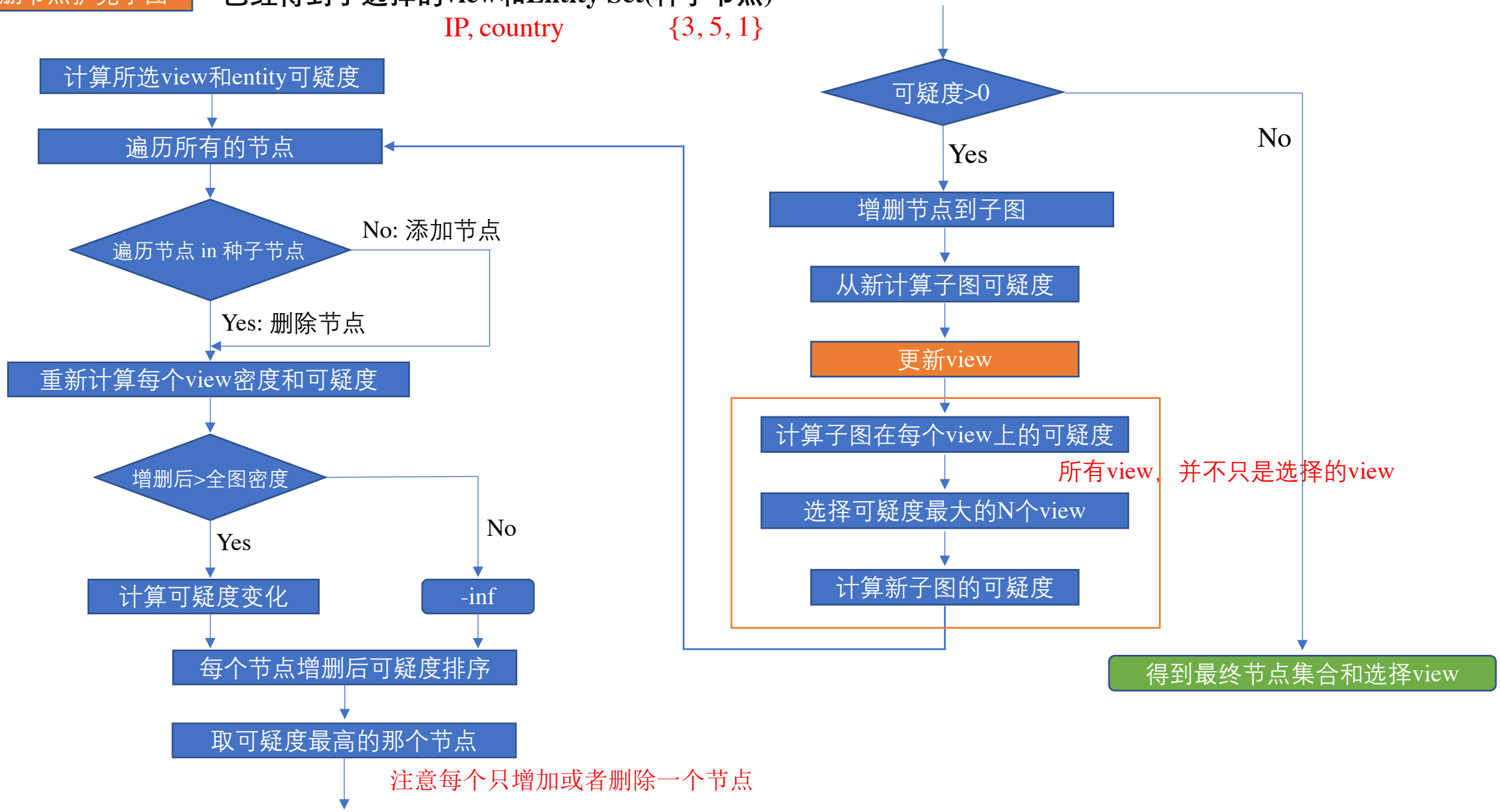
逻辑回归的损失函数：$\mathrm{P(y|x;\theta)} = h_\theta(x)^y * (1-h_\theta(x))^{1-y}$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} log h_\theta(x^{(i)}) + (1-y^{(i)}) log(1-h_\theta(x^{(i)}))]$$

增删节点扩充子图

已经得到了选择的view和Entity Set(种子节点)

IP, country {3, 5, 1}



Algorithm 1 SLICENDICE

Require: MVG $\mathcal{G}$ (N nodes, K views, $\vec{C}$ masses), constraint $z \leq K$ 视图数量限制

$\vec{k} \leftarrow \text{SEEDVIEWS}(\mathcal{G}, z)$ z个视图 ▷ choose z views

$\mathcal{X}_{\vec{k}} \leftarrow \text{SEEDNODES}(\mathcal{G}, \vec{k})$ n个节点 ▷ n nodes, $\vec{c}$ masses

$S \leftarrow f\left(n, \vec{c}, N, \vec{C}\right)$ ▷ compute suspiciousness metric

do ▷ alternating optimization

  $S' \leftarrow S$

  $\vec{k} \leftarrow \text{UPDATEVIEWS}(\mathcal{G}, \mathcal{X})$ 修改视图集合 ▷ revise view set

  $\mathcal{X}_{\vec{k}} \leftarrow \text{UPDATENODES}(\mathcal{G}, \mathcal{X}_{\vec{k}})$ 修改节点集合 ▷ revise node set

  $S \leftarrow f\left(n, \vec{c}, N, \vec{C}\right)$ 重新计算异常值S

while $S > S'$ ▷ repeat until S converges

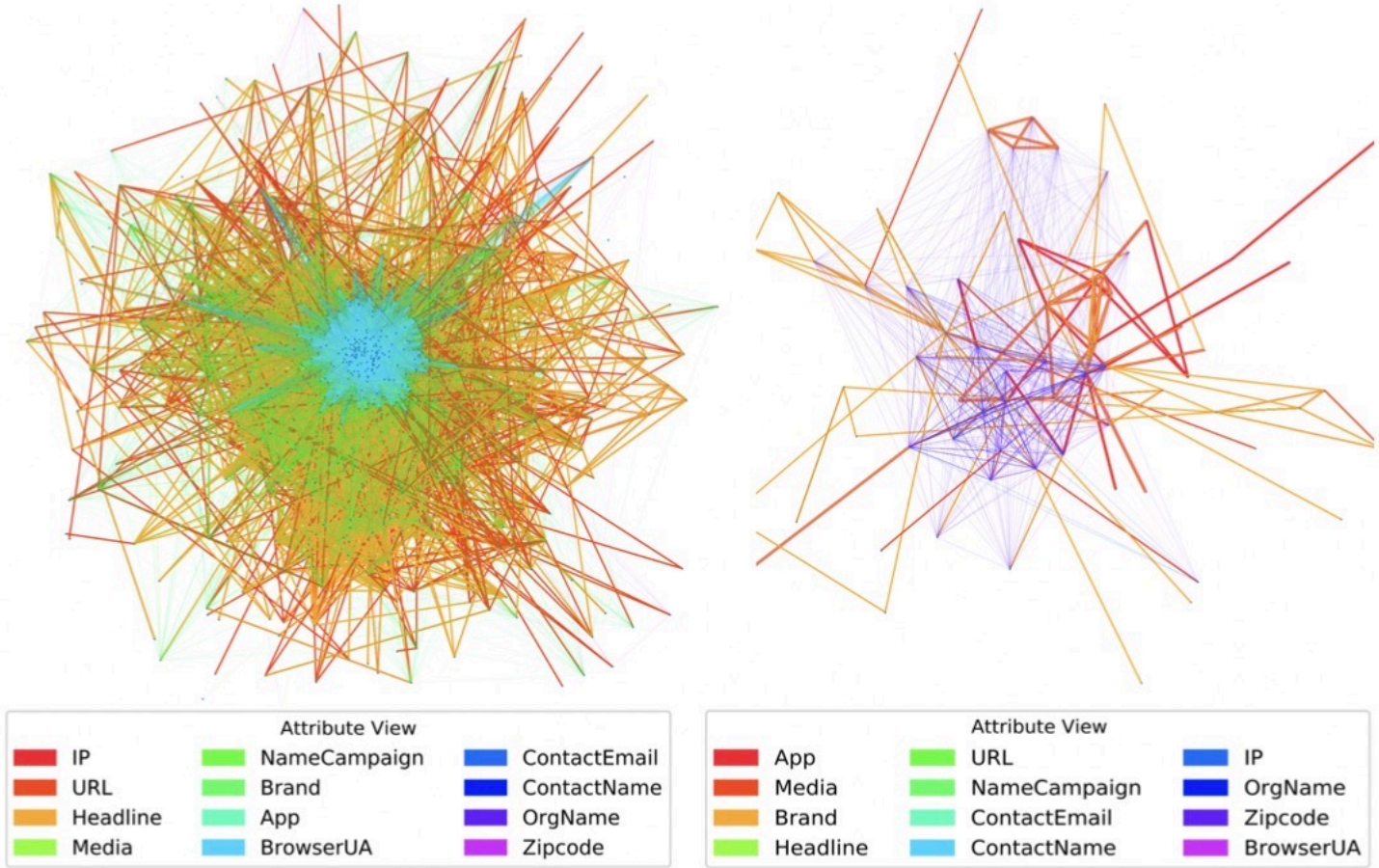
return $\left(\mathcal{X}_{\vec{k}}, S\right)$

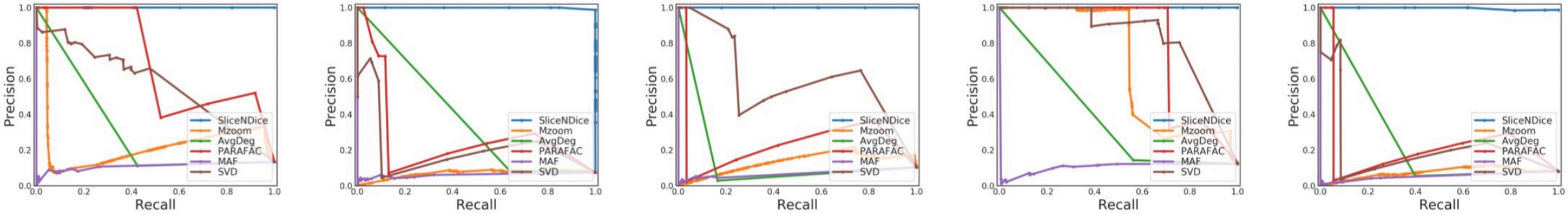
贪心法选择节点和视图

Algorithm 2 GREEDYSEED

Require: MVG $\mathcal{G}$ (N nodes, K views, P dens.), views $\vec{k}$

define SHUFFLE(S): return S in random order
define CHOOSE(S, r): return r random elements from S
$\mathcal{V} \leftarrow \{i \mid \mathbb{1}(k_i)\}$ ▷ chosen view set
$\mathcal{H}_i^{ve}(a) \leftarrow a \Rightarrow \mathcal{A}_i^{-1}(a)\ \forall a \in \mathcal{A}, i \in \mathcal{V}$ ▷ value-to-entity hashmap
$\mathcal{H}_i^{ev}(a) \leftarrow a \Rightarrow \mathcal{A}_i(a)\ \forall a \in \mathcal{G},\ i \in \mathcal{V}$ ▷ entity-to-value hashmap
$i \leftarrow$ CHOOSE($\mathcal{V}, 1$) ▷ choose a view
$a \leftarrow$ CHOOSE($\{a \mid |\mathcal{H}_i^{ve}(a) \mid \geq 2\}, 1$) ▷ choose a shared value
$\mathcal{X} \leftarrow$ CHOOSE($\mathcal{H}_i^{ve}(a),\ 2$) ▷ initialize seed with similar entities
for view $i \in$ SHUFFLE($\mathcal{V}$) **do**
    $t \leftarrow 0$ ▷ keep track of attempts
    while ($\rho_i < P_i$ and t ¡ 20) **do** ▷ try to satisfy constraint
        $e_1 \in$ SHUFFLE($\mathcal{X}, 1$) ▷ choose entity already in $\mathcal{X}$
        $a \leftarrow$ CHOOSE($\mathcal{H}_i^{ev}(e_1), 1$) ▷ choose a shared value
        $e_2 \leftarrow$ CHOOSE($\mathcal{H}_i^{ve}(a), 1$) ▷ choose a similar entity
        $\mathcal{X} \leftarrow \mathcal{X} \cup e_2$ ▷ grow seed
        $t \leftarrow t + 1$
    end while
    if ($\rho_i < P_i$) **then**
        go to 6 ▷ start fresh if constraint not yet met
    end if
end for
return $\mathcal{X}_{\vec{k}}$

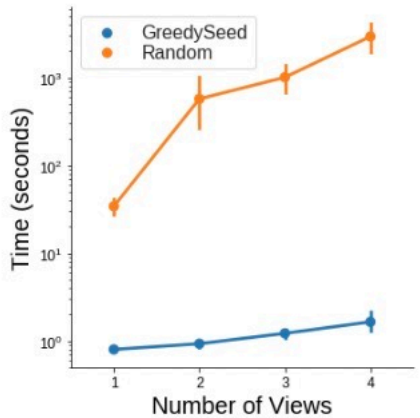

Attribute View
IP
URL
Headline
Media
NameCampaign
Brand
App
BrowserUA
ContactEmail
ContactName
OrgName
Zipcode
Attribute View
App
Media
Brand
Headline
URL
NameCampaign
ContactEmail
ContactName
IP
OrgName
Zipcode
BrowserUA

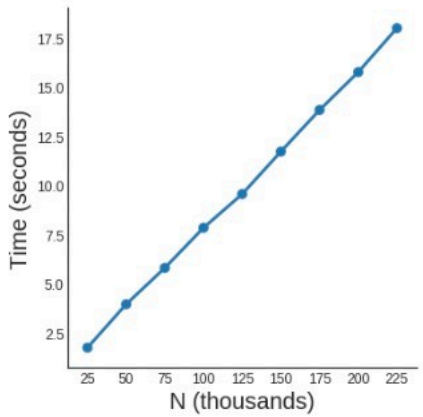


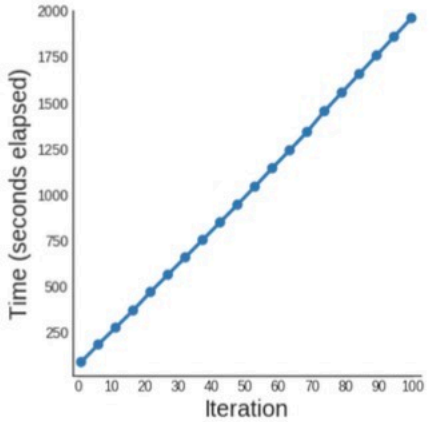
(a) High attack sync. (b) Low attack sync. (c) High-signal attacks (d) Low-signal attacks (e) High-dim attacks

1) **High attacker synchrony:** Default settings; attacks sample attributes from $1/10$th of associated attribute spaces, making them much denser than the norm.
2) **Low attacker synchrony:** $\tau = 2$. Attribute spaces are restricted to $1/2$ and thus much harder to detect.
3) **High-signal attribute attacks:** Attack views are sampled with weights $\propto \vec{u}$ (more likely to land in sparse views).
4) **Low-signal attribute attacks:** Attack views are sampled with weights $\propto 1/\vec{u}$ (more likely to land in dense views).
5) **Attacks in high dimensionality:** $K = 30$. Attacks are highly distributed in $3\times$ higher dimensionality.

不同攻击下算法的效果



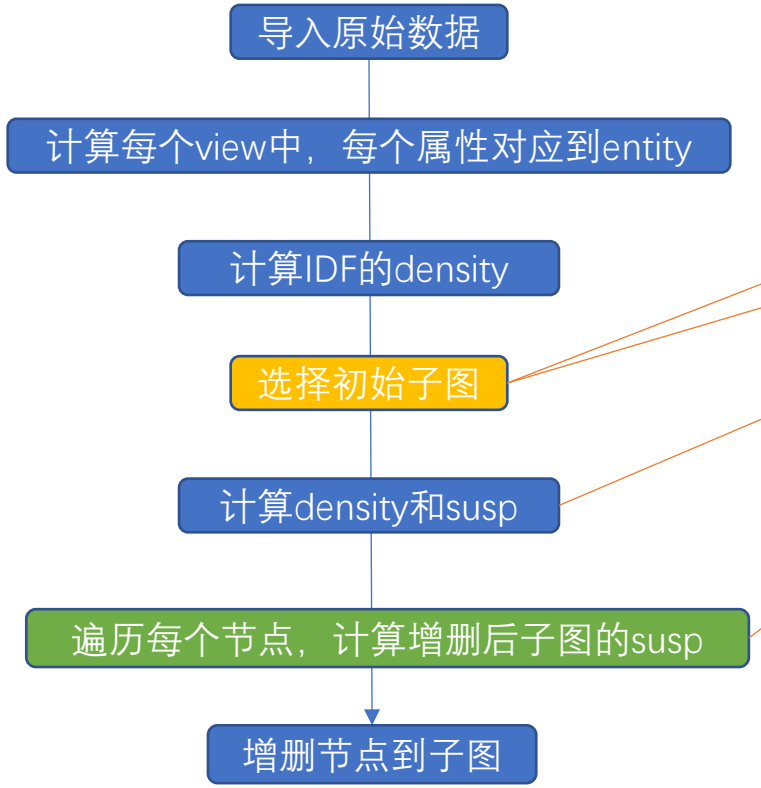




(a) Time to seed (b) Time vs. # entities (c) Time vs. # iterations

运行时间

Code



1: $\vec{k} \leftarrow \text{SEEDVIEWS}(\mathcal{G}, z)$ z个视图
2: $\mathcal{X}_{\vec{k}} \leftarrow \text{SEEDNODES}(\mathcal{G}, \vec{k})$ n个节点
3: $S \leftarrow f\left(n, \vec{c}, N, \vec{C}\right)$
4: **do**
5: $S' \leftarrow S$
6: $\vec{k} \leftarrow \text{UPDATEVIEWS}(\mathcal{G}, \mathcal{X})$ 修改视图集合
7: $\mathcal{X}_{\vec{k}} \leftarrow \text{UPDATENODES}(\mathcal{G}, \mathcal{X}_{\vec{k}})$ 修改节点集合
8: $S \leftarrow f\left(n, \vec{c}, N, \vec{C}\right)$ 重新计算异常值S
9: **while** $S > S'$
10: **return** $\left(\mathcal{X}_{\vec{k}}, S\right)$

Code

```
for view in self.job_params.ALL_VIEWS:  # 遍历视图  view: 'country'
    counts = {}  # count number of recurring values across every sing
    attr_to_entity_map = {}  # mapping from attribute values to all e

    for row in chosen_entity_rows:  # 遍历实体  row: <class 'dict'>: {
        attribute_set = row[view]  # 每个实体中，该视图对应的属性  attrib
        entity_id = row[self.job_params.ENTITY_ID_FIELD]  # 实体id  en

        for element in attribute_set:  # 遍历属性  element: 'Iran'
            if is_valid_attr_value(element):  # 属性值为str或数值，非空
                if element not in counts:
                    counts[element] = 0  # 记录这个属性包含节点数量
                    if return_attr_to_entity_map:  # 是否返回关联映射
                        attr_to_entity_map[element] = set()
                counts[element] += 1  # 记录该视图属性中包含的节点数量

                if return_attr_to_entity_map:  # 返回属性对应到的id
                    attr_to_entity_map[element].add(entity_id)
```

原始数据

{"entity_id": "1", "country": "Libya,Germany", "IP":"1.2.3.1"}
{"entity_id": "2", "country": "Libya", "IP":"1.2.3.4"}
{"entity_id": "3", "country": "Germany", "IP":"1.2.3.4"}
{"entity_id": "4", "country": "Germany", "IP":"1.2.3.2"}
{"entity_id": "5", "country": "USA,Germany", "IP":"1.2.3.4"}
{"entity_id": "6", "country": "USA", "IP":"1.2.3.5"}
{"entity_id": "7", "country": "Iran", "IP":"1.2.3.5"}

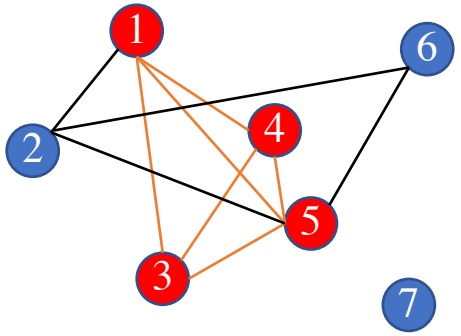
每个属性包含的entity数量

```
In[24]: counts
Out[24]: {'Germany': 4, 'Libya': 2, 'USA': 3, 'Iran': 1}
```

每个属性包含的entity集合

```
In[25]: attr_to_entity_map
Out[25]: {'Germany': {1, 3, 4, 5}, 'Libya': {1, 2}, 'USA': {2, 5, 6}, 'Iran': {7}}
```

```
In[25]: attr_to_entity_map
Out[25]: {'Germany': {1, 3, 4, 5}, 'Libya': {1, 2}, 'USA': {2, 5, 6}, 'Iran': {7}}
```



$$c_i = \sum_{v_i \in A_i} ief(v_i)(\mathcal{J}_i(v_i)^2 - \mathcal{J}_i(v_i))/2$$

完全图边计算公式

N*(N-1)

```
mass = sum(int(value_count) ** 2 - int(value_count) for value_count in counts.values())

volume = ((size * (size - 1)) / 2.0)  # 体积的计算
```

$$V = N(N-1)/2$$

```
In[37]: block_metadata
Out[37]:
{'country': (5.0,
  7.0,
  0.23809523809523808,
  {'Libya': 2, 'Germany': 3, 'USA': 2, 'Iran': 1},
  {'Libya': {1, 2}, 'Germany': {1, 3, 4}, 'USA': {5, 6}, 'Iran': {7}})}
```

mass

size

density

counts

attr_to_entity_map

各个视图的信息

Country, IP

```
In[38]: block_metadata
Out[38]:
{'country': (5.0,
  7.0,
  0.23809523809523808,
  {'Libya': 2, 'Germany': 3, 'USA': 2, 'Iran': 1},
  {'Libya': {1, 2}, 'Germany': {1, 3, 4}, 'USA': {5, 6}, 'Iran': {7}}),
 'IP': (4.0,
  7.0,
  0.19047619047619047,
  {'1.2.3.1': 1, '1.2.3.4': 3, '1.2.3.2': 1, '1.2.3.5': 2},
  {'1.2.3.1': {1}, '1.2.3.4': {2, 3, 5}, '1.2.3.2': {4}, '1.2.3.5': {6, 7}})}
```

mass

size

counts

attr_to_entity_map

```
term_idf_score_map[term] = np.log(1 + (1.0 / float(term_freq))) ** 2  # 计算idf值
```

$$\log(1 + 1/A)^2$$

每个视图中各个属性的idf值

```
In[45]: self.term_idf
Out[45]:
{'country': {'Libya': 0.16440195389316542,
  'Germany': 0.08276097481015168,
  'USA': 0.16440195389316542,
  'Iran': 0.4804530139182014},
 'IP': {'1.2.3.1': 0.4804530139182014,
  '1.2.3.4': 0.08276097481015168,
  '1.2.3.2': 0.4804530139182014,
  '1.2.3.5': 0.16440195389316542}}
```

```
self.tensor_stats = self.compute_block_metadata(self.all_entities, disable_idf=False)  # 根据IDF再次计算mass
```

self.tensor_stats_noidf: 没有idf各个视图属性
self.tensor_stats: 使用idf各个视图属性

```
In[32]: self.tensor_stats
Out[32]:
{'country': (0.7114431452823247,
  7.0,
  0.03387824501344403,
  {'Germany': 4, 'Libya': 2, 'USA': 3, 'Iran': 1}),
 'IP': (0.41268487832362044,
  7.0,
  0.019651660872553354,
  {'1.2.3.1': 1, '1.2.3.4': 3, '1.2.3.2': 1, '1.2.3.5': 2})}
```

mass
size
density
counts

```
In[33]: self.tensor_stats_noidf
Out[33]:
{'country': (10.0,
  7.0,
  0.476190476190476 16,
  {'Germany': 4, 'Libya': 2, 'USA': 3, 'Iran': 1},
  {'Germany': {1, 3, 4, 5}, 'Libya': {1, 2}, 'USA': {2, 5, 6}, 'Iran': {7}}),
 'IP': (4.0,
  7.0,
  0.19047619047619047,
  {'1.2.3.1': 1, '1.2.3.4': 3, '1.2.3.2': 1, '1.2.3.5': 2},
  {'1.2.3.1': {1}, '1.2.3.4': {2, 3, 5}, '1.2.3.2': {4}, '1.2.3.5': {6, 7}})}
```

mass
size
density
counts

```
self.cull_attr_to_entity_map()
```

```
In[35]: self.tensor_stats_noidf
Out[35]:
{'country': (10.0, mass
  7.0, size
  0.47619047619047616, density
  {'Germany': 4, 'Libya': 2, 'USA': 3, 'Iran': 1}, counts
  {'Germany': {1, 3, 4, 5}, 'Libya': {1, 2}, 'USA': {2, 5, 6}, 'Iran': {7}},
  {'Germany': {1, 3, 4, 5}, 'Libya': {1, 2}, 'USA': {2, 5, 6}}),
 'IP': (4.0,
  7.0,
  0.19047619047619047,
  {'1.2.3.1': 1, '1.2.3.4': 3, '1.2.3.2': 1, '1.2.3.5': 2},
  {'1.2.3.1': {1}, '1.2.3.4': {2, 3, 5}, '1.2.3.2': {4}, '1.2.3.5': {6, 7}},
  {'1.2.3.4': {2, 3, 5}, '1.2.3.5': {6, 7}})}
```

新添加属性和节点之间的连接情况

选择初始子图

1. Choose view
2. Choose entities

Choose view

1. 计算每个view中，属性对应到的实体
2. 计算实体数量的99.5%分位数的值
3. 分位数值的倒数当作是选择view的概率
4. 选择k个view

Country:[100,10,10]

IP：[10,10,10,10,10,,,1,,,]

```
In[67]: np.percentile([100,10,10],percentile)
Out[67]: 99.1
In[68]: np.percentile([100,]+[1]*20,percentile)
Out[68]: 90.09999999999985
In[69]: np.percentile([10]*10+[1]*20,percentile)
Out[69]: 10.0
```

views_to_sort = [0.01, 0.1] IP被选择的概率大

属性集中性低，被选中的概率就大

Entity Choose

这个视图中的所有连接节点

{'1.2.3.4': {2, 3, 5}, '1.2.3.5': {6, 7}}

随机选择一个候选view — IP

随机选择两个相连接的节点 — {2, 3, 5} -> {3, 5}　因为两个节点连接，则说明他们密度一定大于0，要比随机选择两个节点密度期望更大

计算两个节点子块的密度 — {3, 5}子图在所有view下密度(Country, IP)

每个view全体节点的密度

遍历每个view

子图密度>全图密度

No

遍历每个不满足要求view

该view连接节点 — {'Libya': {1, 2}, 'Germany': {1, 3, 4, 5}, 'USA': {2, 5, 6}}

遍历候选节点{3, 5}

节点 in 连接集合

No

重新Entity Choose

{3}, {5}是否在{1, 2}, {1, 3, 4, 5}, {2, 5, 6}中

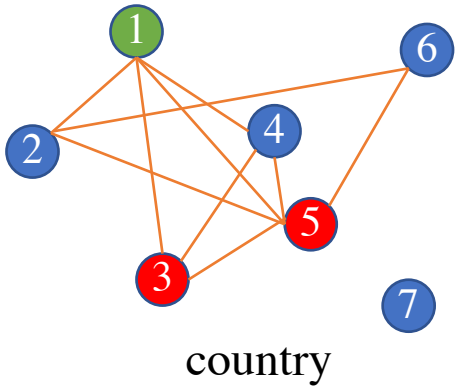
Yes

随机添加一个节点 — {1}

{3, 5, 1}

Yes

最终Entity Set



已经得到了选择的view和Entity Set

因为在该view上不满足密度要求，那么在添加一个连接的边的节点，则大概率会增大密度，以达到密度要求

计算可疑度

```
volume = (num_nodes_in_block * (num_nodes_in_block - 1)) / 2.0
susp = (volume * np.log(background_view_density)) + (volume * np.log(volume)) - volume - np.log(volume) - (
        volume * np.log(mass_in_block)) + np.log(mass_in_block) + (
            mass_in_block * (1.0 / background_view_density))
```

$$= \sum_{i=1}^{K} v \log \frac{C_i}{V} + v \log v - v - \log v - v \log c_i + \log c_i + \frac{V c_i}{C_i}$$

C_i: i视图的全体节点mass
V: 体积(所有view体积一致)

c_i: i视图的子图节点mass
v: 子图体积

volumne: 体积 v

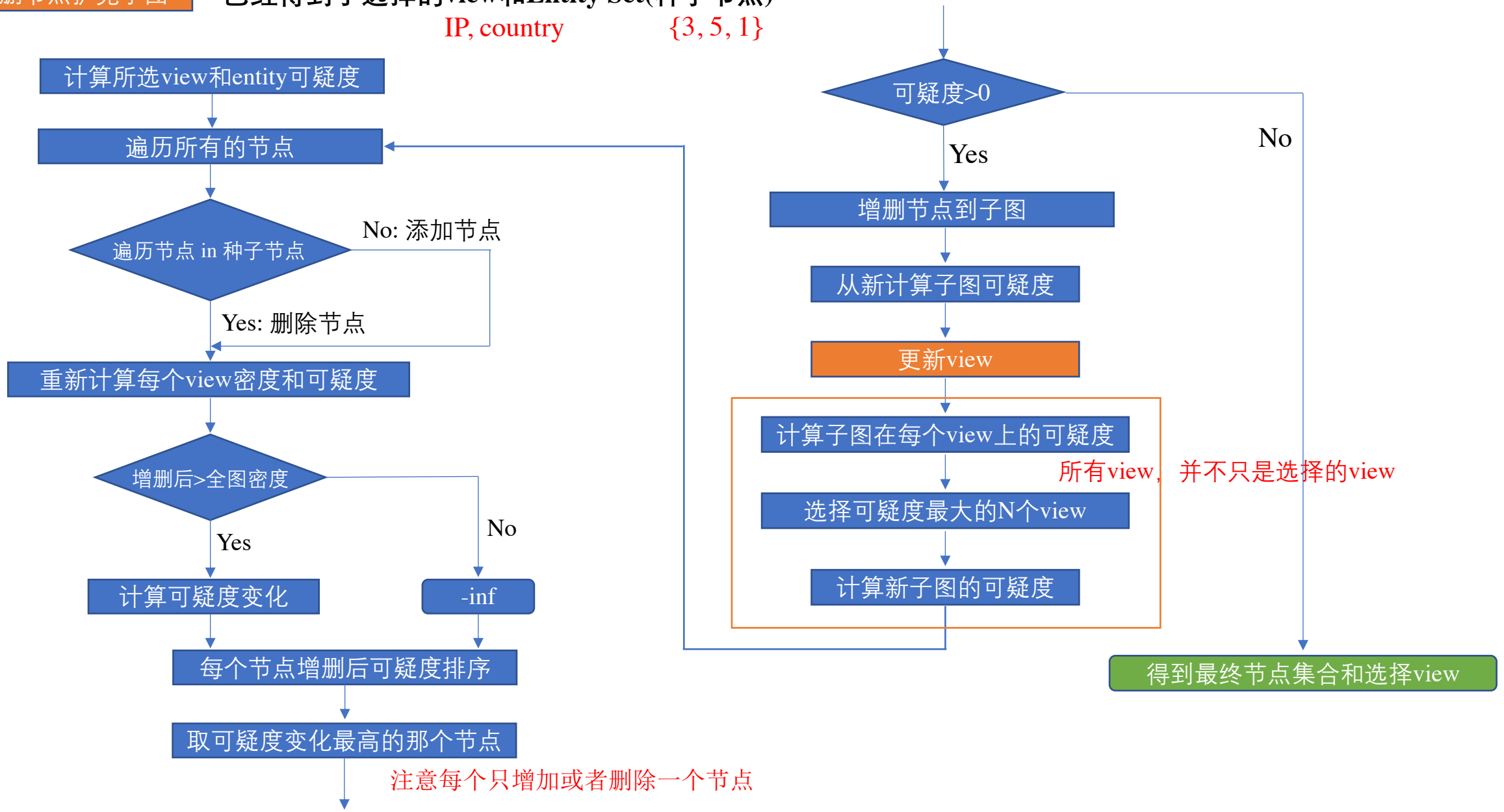
background_view_density: 总体密度 $\frac{C_i}{V}$

mass_in_block: 块质量 c_i

增删节点扩充子图

已经得到了选择的view和Entity Set(种子节点)

IP, country {3, 5, 1}



Deep Structure Learning for Fraud Detection (SDNE)

使用场景：二部图（电商，评论等等）

解决问题：同时保留了二部图的结构信息和用户的行为信息，将其压缩为embedding向量

算法思路：欺诈用户会同时购买某些商品，必然会产生聚集性。而正常用户在购买行为会趋于均匀分布。

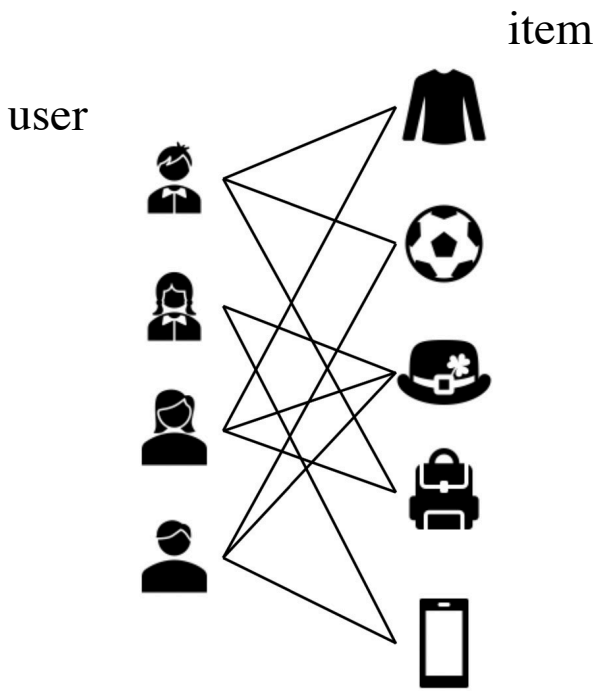


Figure 2.7 An e-commerce bipartite graph

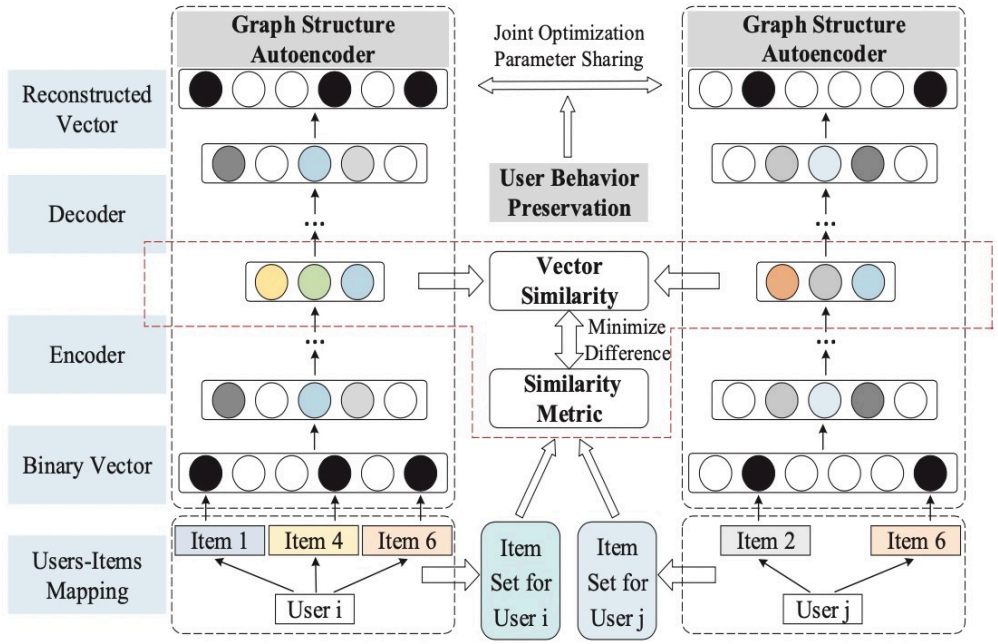


Figure 2. The deep embedding framework of DeepFD model

III. Our Solution: DeepFD Model

preserve global graph structure information and user behavior characteristics simultaneously

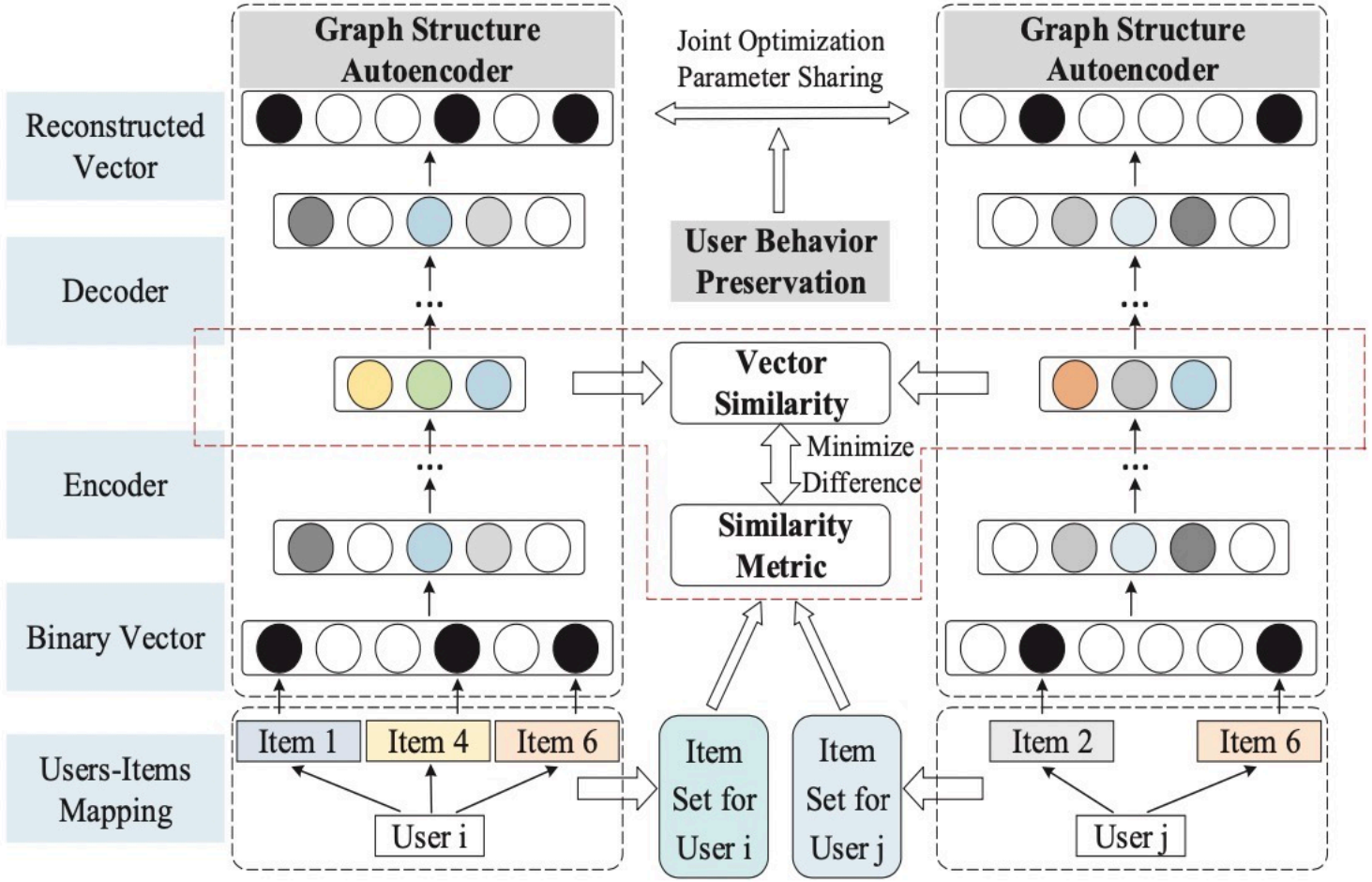


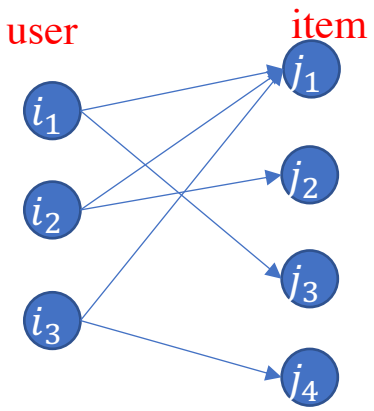
Figure 2. The deep embedding framework of DeepFD model

A. Graph Structure Reconstruction

B. User Behavior Preservation

A. *Graph Structure Reconstruction*

训练数据的构造



$$S = \{s_1, s_2, \ldots, s_m\}$$

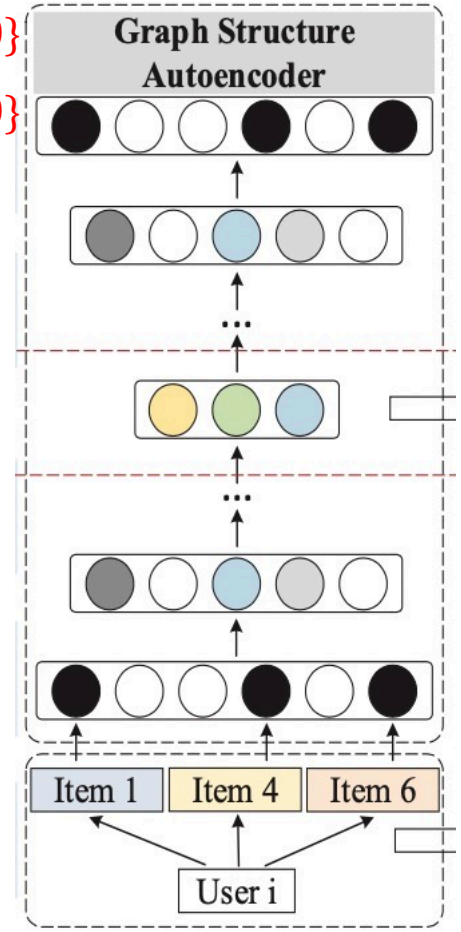
$$s_i = \{s_{ij}\}_{j=1}^{n}$$

$S_{ij} = 1$ 用户i购买物品j

$$S_1 = \{S_{11}, S_{13}\} = \{1,0,1,0\}$$
$$S_2 = \{S_{21}, S_{22}\} = \{1,1,0,0\}$$
$$S_3 = \{S_{11}, S_{13}\} = \{1,0,0,1\}$$

$\hat{S}_1 = \{1,0,0,0\}$

$\hat{S}_2 = \{1,1,1,0\}$



$S_1 = \{1,0,1,0\}$

$S_2 = \{1,1,0,0\}$

使用一个AutoEncoder构建重构误差

loss

$$\mathcal{L}_{tmp} = \sum_{i=1}^{m} ||\hat{s}_i - s_i||_2^2 \qquad (2)$$

user和item的连接是稀疏的，
loss会倾向于所有输出均为0

$$\mathcal{L}_{recon} = \sum_{i=1}^{m} ||(\hat{s}_i - s_i) \odot h_i||_2^2 = ||(\hat{S} - S) \odot H||_2^2 \qquad (3)$$

对原始input非0项添加一个大权重

$S_1 = \{1,0,1,0\}$ $H_1 = \{10,1,10,1\}$

$\hat{S}_1 = \{1,0,0,0\}$

$loss_1 = [(\{1,0,1,0\} - \{1,0,0,0\}) * (\{10,1,10,1\})]^2$

1判断成0了，给予更大的惩罚

B. User Behavior Preservation

同一个欺诈团伙的欺诈行为具有相似性，但是正常用户，他们的行为具有差异性

$$dis_{ij} = ||(g_i^{(K)} - g_j^{(K)})||_2^2 \quad (4)$$

i, j两个节点

$$\widehat{sim}_{ij} = \exp\left(-\lambda \cdot dis_{ij}\right) \quad (5)$$

距离越小，相似性越大

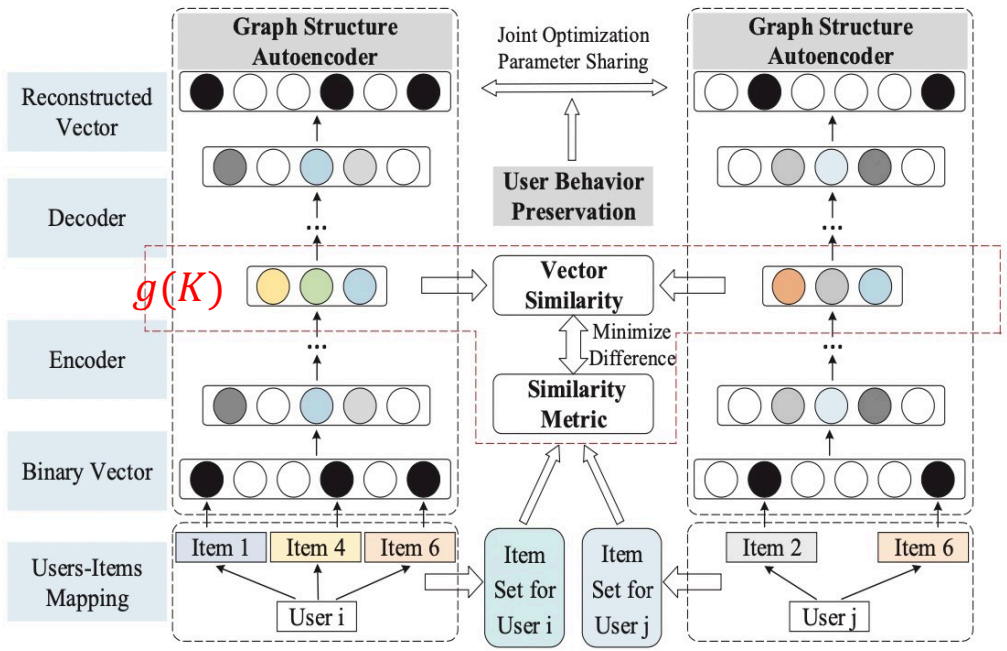
$$\mathcal{L}'_{sim} = \sum_{i,j=1}^{m} ||\widehat{sim}_{ij} - sim_{ij}||_2^2 \quad (6)$$

预先定义的相似度

$$sim_{ij} = \begin{cases} \frac{|N_i \cap N_j|+1}{|N_i \cup N_j|+n} & |N_i \cap N_j| = \varnothing, \\ \frac{|N_i \cap N_j|+(n-1)}{|N_i \cup N_j|+n} & |N_i \cap N_j| = |N_i \cup N_j|), \\ \frac{|N_i \cap N_j|}{|N_i \cup N_j|} & otherwise. \end{cases}$$

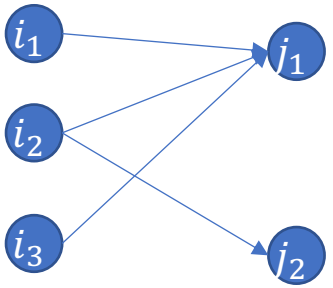
一般情况下，一个群组中的相似的行为意味着存在欺诈的可能性会比较大。所以我们给予相似用户以更大惩罚权重

$$\mathcal{L}_{sim} = \sum_{i,j=1}^{m} sim_{ij} \cdot ||\widehat{sim}_{ij} - sim_{ij}||_2^2 \quad (7)$$



相似性的度量：

$$sim_{ij} = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}, \text{ where } N_i := \{y_j \in Y : e_{ij} = 1\}$$



如果两个用户共享很多item，则两个用户具有高的相似性

$$N_1 = \{j_1\} \quad N_2 = \{j_1, j_2\}$$

$$sim_{1,2} = \frac{|j_1|}{|j_1 + j_2|} = 0.5$$

可能存在$|N_i \cap N_j| = 0$和$|N_i \cap N_j| = |N_i \cup N_j|$的情况，我们对相似度度量加入平滑项

$$sim_{ij} = \begin{cases} \frac{|N_i \cap N_j|+1}{|N_i \cup N_j|+n} & |N_i \cap N_j| = \varnothing, \\ \frac{|N_i \cap N_j|+(n-1)}{|N_i \cup N_j|+n} & |N_i \cap N_j| = |N_i \cup N_j|), \\ \frac{|N_i \cap N_j|}{|N_i \cup N_j|} & otherwise. \end{cases}$$

n is the total number of the item nodes

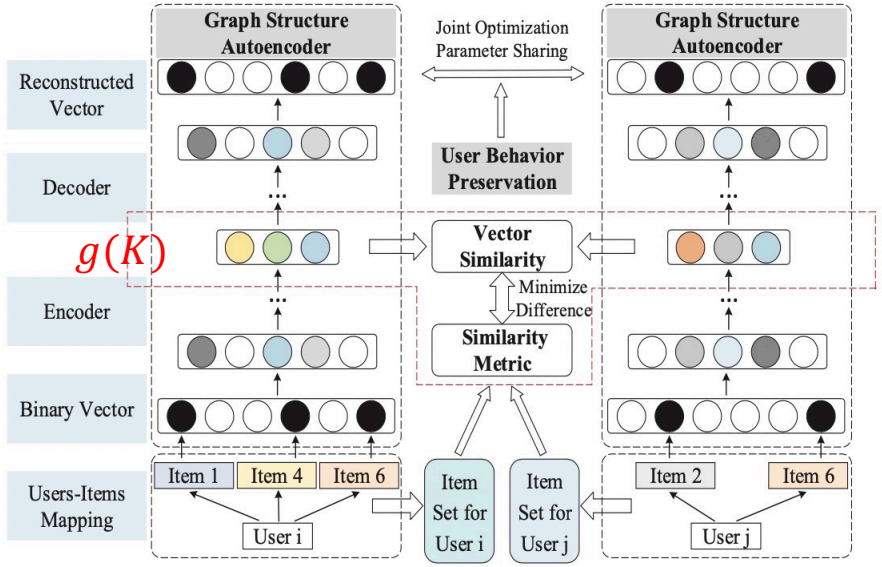
C. Model Summary and Optimization

联合优化目标函数

$$\mathcal{L} = \mathcal{L}_{recon} + \alpha\mathcal{L}_{sim} + \gamma\mathcal{L}_{reg} \qquad (8)$$

$$\mathcal{L}_{reg} = \frac{1}{2}\sum_{l=1}^{K}(||W^{(l)}||_2^2 + ||\hat{W}^{(l)}||_2^2 + ||b^{(l)}||_2^2 + ||\hat{b}^{(l)}||_2^2) \quad (9)$$ L2-norm

在优化节点之间的相似度时，需要对两两节点进行采样，计算量很大，这里采用负采样方式降低计算量。



负采样方法：比如对节点i进行计算

1. 第一步，采样N个和节点i至少共享一个item的节点j，计算他们的相似度loss
2. 第二步，采样N个和节点i没有任何共享item的节点集合，计算他们的相似度loss

D. Deep Structure Learning for Fraud Detection

得到节点embedding$g(K)$后，采用DBSCAN进行聚类，输出异常群组

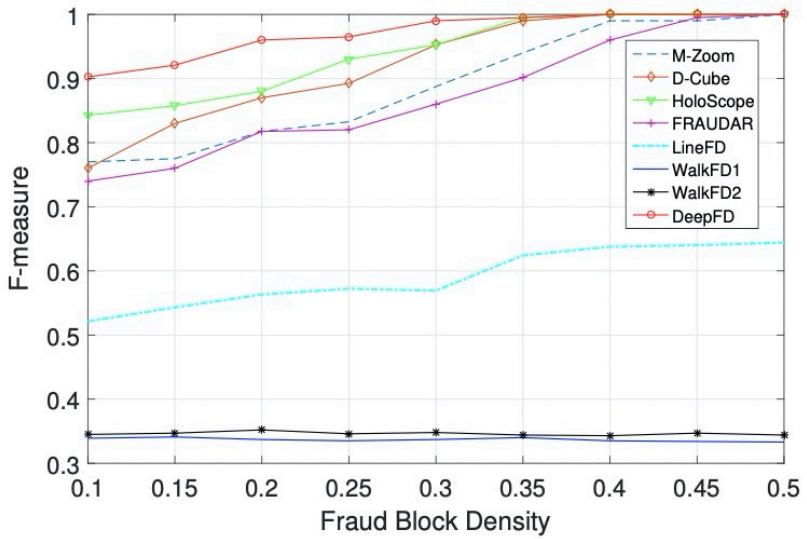
Algorithm 1 DeepFD model

Input: The interaction information graph $G = (X, Y, E)$, The hyper-parameters α and γ, Batch size p, Embedding size d, Number of iterations I

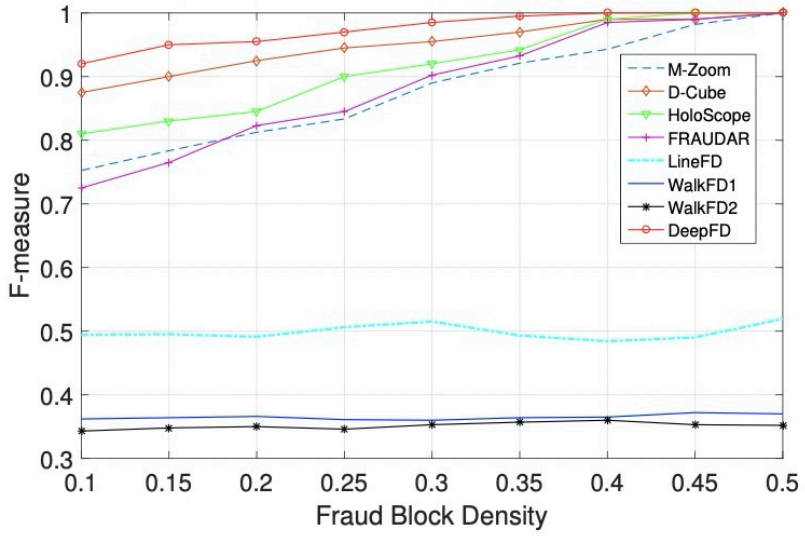
Output: Fraudsters for all fraud blocks $fraudblock$

Initialize the parameter set $\{\hat{W}^{(l)}, W^{(l)}, \hat{b}^{(l)}, b^{(l)}\}_{l=1}^{K}$ by Deep Belief Network;
sim = GetSimliariyByNegativeSampling(G);
for $i = 0$ to I **do**
    sim' = Shuffle(sim);
    while True **do**
        $minibatch$ = Sample(sim', p);
        $nodeset$ = ConstructUserNodes($minibatch$);
        $loss$ = Optimize(α, γ, $nodeset$, $minibatch$);
        if Batch sampling is end **then**
            break;
        end if
    end while
end for
f = GetEmbeddingResults(X);
$fraudblock$ = FindFraudblockByDBSCAN(f);
return $fraudblock$;

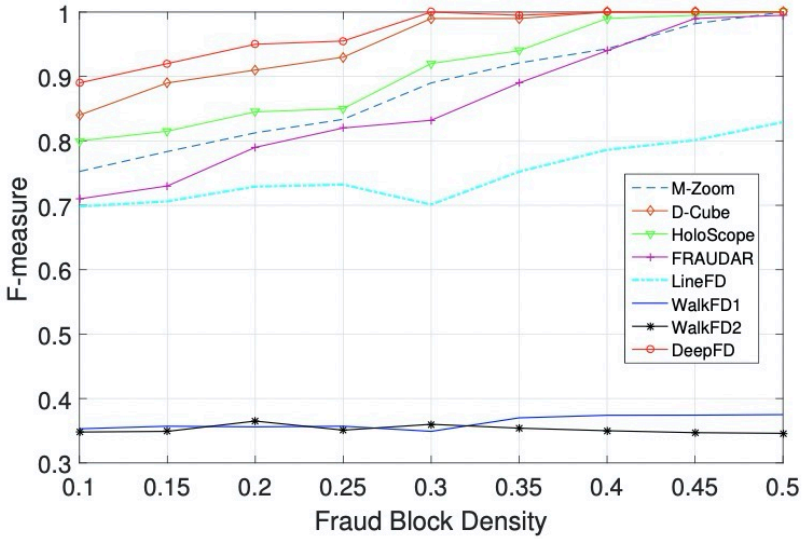
$$\mathcal{L} = \mathcal{L}_{recon} + \alpha \mathcal{L}_{sim} + \gamma \mathcal{L}_{reg} \tag{8}$$



(a) Yelp



(b) Amazon Instrument



(c) Amazon Movie

各种方法的欺诈检测效果

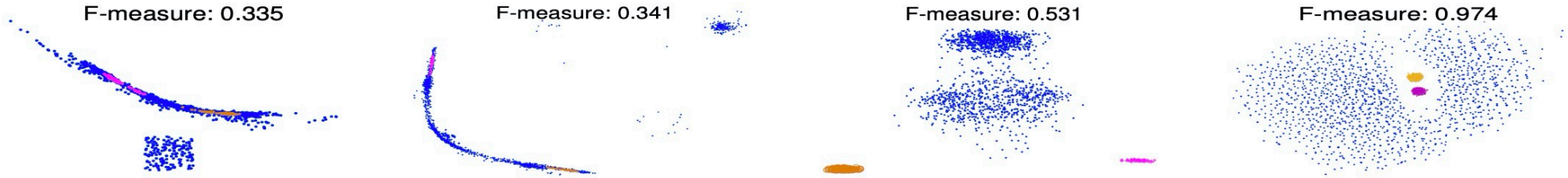
其他的异常检测方法

- M-Zoom
- D-Cube
- HoloScope
- FRAUDAR

Graphembedding方法

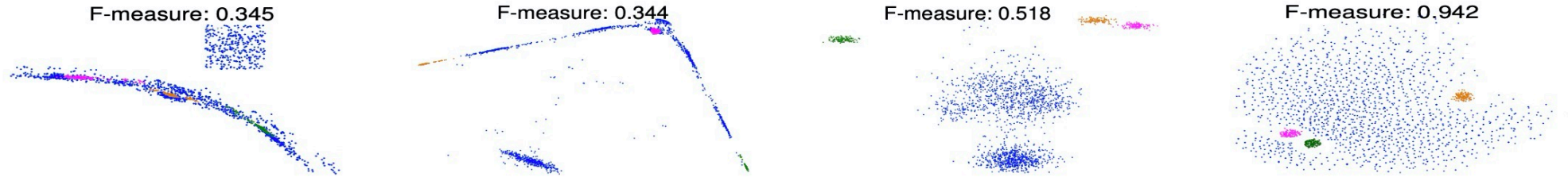
- WalkFD1
- WalkFD2
- LineFD

多个欺诈块情况



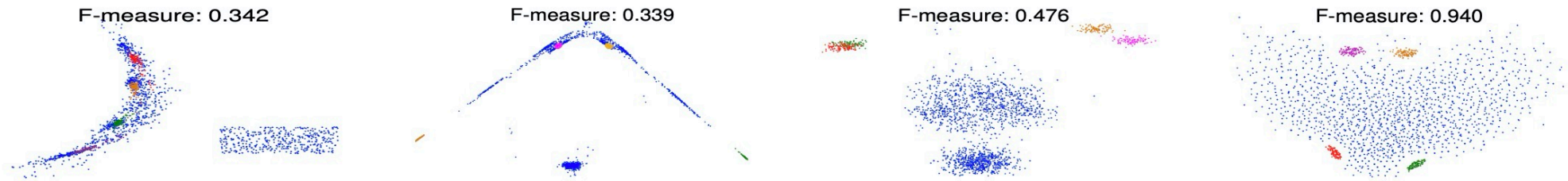
(a) WalkFD1 (b) WalkFD2 (c) LineFD (d) DeepFD

Figure 4. F-measure comparison for different fraud detection methods and visualization for embedding results (**2 fraud blocks injected**). Blue color represents normal users, and other colors represent different fraud blocks.



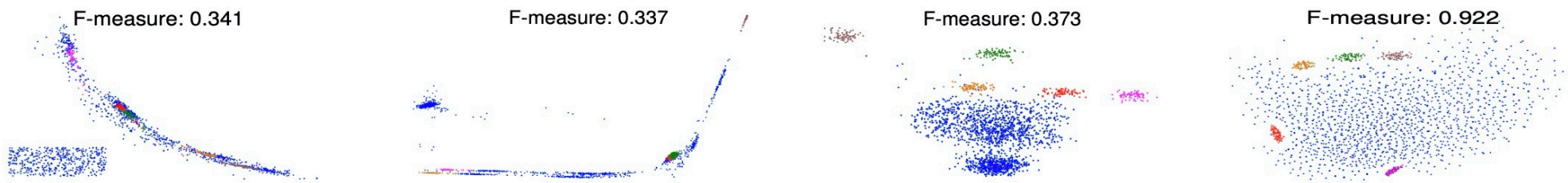
(a) WalkFD1 (b) WalkFD2 (c) LineFD (d) DeepFD

Figure 5. F-measure comparison for different fraud detection methods and visualization for embedding results (**3 fraud blocks injected**). Blue color represents normal users, and other colors represent different fraud blocks.



(a) WalkFD1 (b) WalkFD2 (c) LineFD (d) DeepFD

Figure 6. F-measure comparison for different fraud detection methods and visualization for embedding results (**4 fraud blocks injected**). Blue color represents normal users, and other colors represent different fraud blocks.



(a) WalkFD1 (b) WalkFD2 (c) LineFD (d) DeepFD

Figure 7. F-measure comparison for different fraud detection methods and visualization for embedding results (**5 fraud blocks injected**). Blue color represents normal users, and other colors represent different fraud blocks.

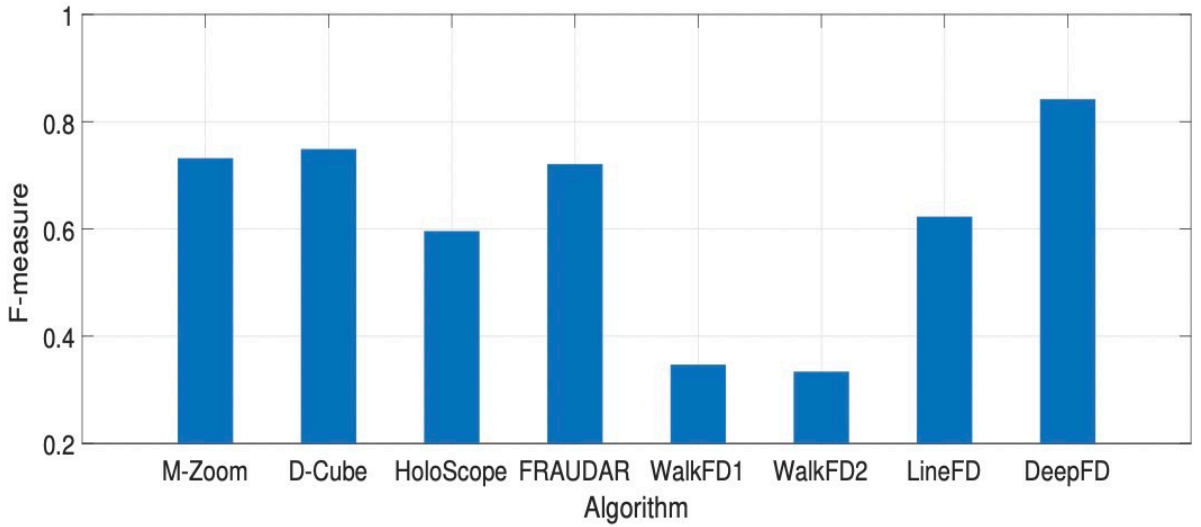
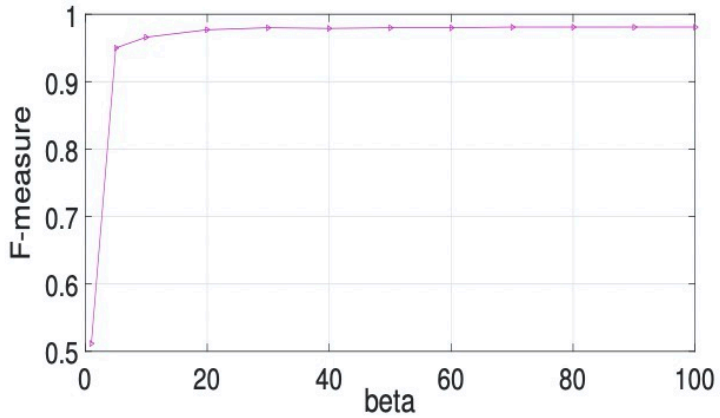
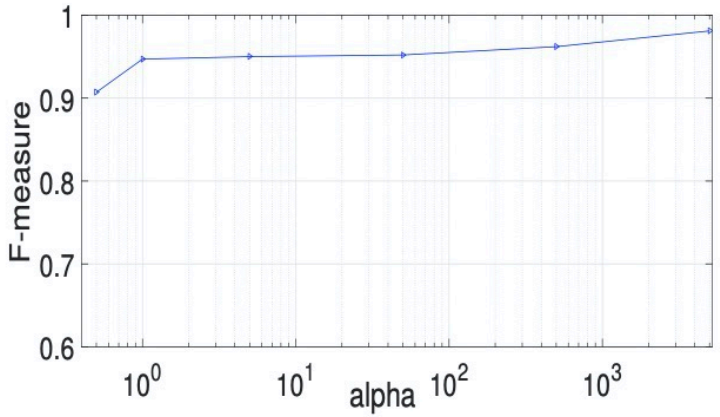


Figure 8. Experimental results on real-world DDos attack dataset

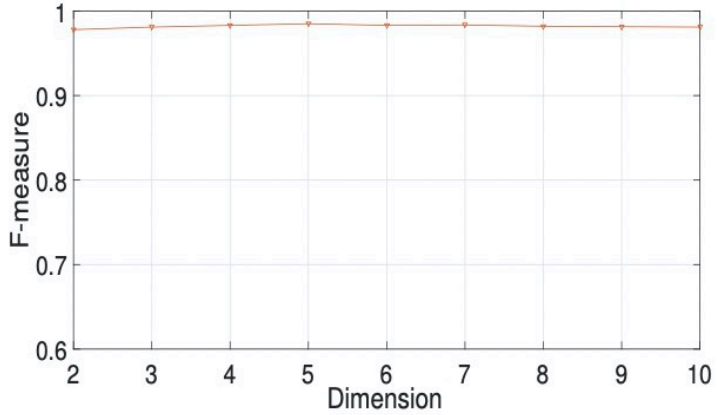
真实数据集上的效果



(a) β



(b) α



(c) Dimension

Figure 9. Parameter sensitivity

$||(\hat{S} - S) \odot H||_2^2$

β autoencoder中非0的惩罚项

$$\mathcal{L} = \mathcal{L}_{recon} + \alpha\mathcal{L}_{sim} + \gamma\mathcal{L}_{reg} \quad (8)$$

参数选择对模型不敏感

Code

```
nz_entries = []  nz_entries: <class 'list'>: []
for i in range(np.shape(graph_u2u)[0]):  # 遍历user  i: 0
    for j in range(i+1, np.shape(graph_u2u)[0]):
        nz_entries.append([i, j])
```

遍历所有user组合

(user, user)　　邻接矩阵

```
results = pool.map(get_simi_single_iter, [(entries_batch, graph_u2p) for entries_batch in batches])
```

```
In[16]: batches[0]
Out[16]:
array([[   0,    1],
       [   0,    2],
       [   0,    3],
       ...,
       [   0,  998],
       [   0,  999],
       [   0, 1000]])
```

```
In[17]: graph_u2p.A
Out[17]:
array([[0., 1., 0., ..., 0., 1., 0.],
       [1., 0., 1., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [1., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 1., 0.]])
```

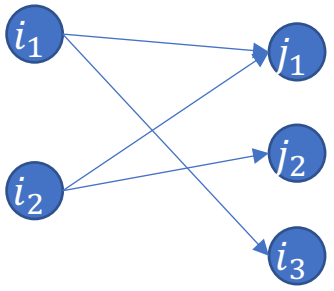
(user, user)　　邻接矩阵

```
results = pool.map(get_simi_single_iter, [(entries_batch, graph_u2p) for entries_batch in batches])
```

```
def get_simi_single_iter(params):
    entries_batch, feats = params
    ii, jj = entries_batch.T  # user1集合, user2集合
    simi = []
    for x in range(len(ii)):  # user
        simi.append(get_simi(feats[ii[x]].toarray(), feats[jj[x]].toarray()))
    simi = np.asarray(simi)
    assert np.shape(ii) == np.shape(jj) == np.shape(simi)
    return ii, jj, simi
```

user　　User所连接的节点

$$sim_{ij} = \begin{cases} \frac{|N_i \cap N_j|+1}{|N_i \cup N_j|+n} & |N_i \cap N_j| = \varnothing, \\ \frac{|N_i \cap N_j|+(n-1)}{|N_i \cup N_j|+n} & |N_i \cap N_j| = |N_i \cup N_j|), \\ \frac{|N_i \cap N_j|}{|N_i \cup N_j|} & otherwise. \end{cases}$$

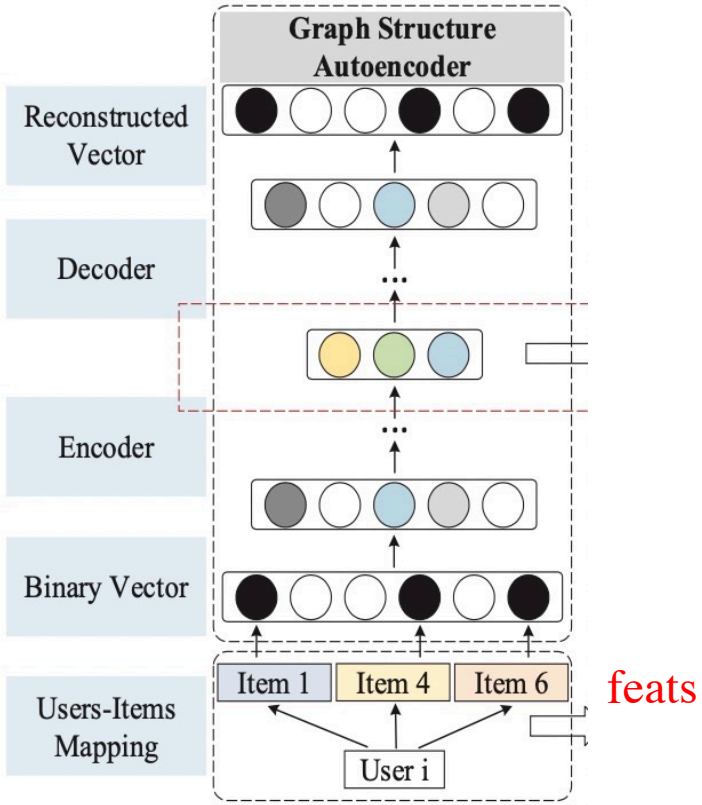



```
def get_simi(u1, u2):  u1: [[0. 0. 0. ... 0. 0. 0.]]  u2: [[0. 0. 0. ... 0. 0. 0
    nz_u1 = u1.nonzero()[1]   [ 6, 52, 57, 81, 109, 110, 411, 519, 753, 900, 1197, 1579, 3649, 3728]
    nz_u2 = u2.nonzero()[1]   [1599]
    nz_inter = np.array(list(set(nz_u1) & set(nz_u2)))
    nz_union = np.array(list(set(nz_u1) | set(nz_u2)))
    if len(nz_inter) == 0:
        simi_score = 1 / (len(nz_union) + len(u1))
    elif len(nz_inter) == len(nz_union):
        simi_score = (len(nz_union) + len(u1) - 1) / (len(nz_union) + len(u1))
    else:
        simi_score = len(nz_inter) / len(nz_union)
    return float(simi_score)
```

Return ii, jj, simi

```
results = pool.map(get_simi_single_iter, [(entries_batch, graph_u2p) for entries_batch in batches])
results = list(zip(*results))  # 所有节点，每个之间的相似度
row = np.concatenate(results[0])
col = np.concatenate(results[1])
dat = np.concatenate(results[2])
```

```
def forward(self, nodes_batch):  self:
    feats = self.features[nodes_batch]
    x_en = F.relu_(self.fc1(feats))
    embs = F.relu_(self.fc2(x_en))
    x_de = F.relu_(self.fc3(embs))
    recon = F.relu_(self.fc4(x_de))
    return embs, recon
```



feats

Loss

```
def get_loss_recon(self, nodes_batch, recon_batch):  self: <src.models
    feats_batch = self.features[nodes_batch]  # 原始节                    _batc
    H_batch = (feats_batch * (self.beta - 1)) + 1
    assert feats_batch.size() == recon_batch.size() == H_batch.size()
    L = ((recon_batch - feats_batch) * H_batch) ** 2
    return L.mean()
```

$h_{ij} = \beta > 1.$

$$\mathcal{L}_{recon} = \sum_{i=1}^{m} ||(\hat{s}_i - s_i) \odot h_i||_2^2 = ||(\hat{S} - S) \odot H||_2^2 \tag{3}$$

```
def get_loss_simi(self, embs_batch):  self: <src.models.Loss_DeepFD object at 0x7ffc24445320>  embs_batch: tenso
    node2index = {n:i for i,n in enumerate(self.extended_nodes_batch)}  # 涉及到的节点; node:index  node2index: <c
    simi_feat = []  simi_feat: <class 'list'>: [tensor([0.0051])]
    simi_embs = []  simi_embs: <class 'list'>: []
    for node, cps in self.node_pairs.items():  # 节点连接到的采样节点  node: 2766  cps: <class 'list'>: [(2766, 14)
        for i, j in cps:  # 采样节点(user, user)  j: 14
            simi_feat.append(torch.FloatTensor([self.graph_simi[i, j]]))  # 节点之间的原始相似度
            dis_ij = (embs_batch[node2index[i]] - embs_batch[node2index[j]]) ** 2  # 节点之间中间embedding的相似度
            dis_ij = torch.exp(-dis_ij.sum())
            simi_embs.append(dis_ij.view(1))
    simi_feat = torch.cat(simi_feat, 0).to(self.device)
    simi_embs = torch.cat(simi_embs, 0)
    L = simi_feat * ((simi_embs - simi_feat) ** 2)
    return L.mean()
```

$$dis_{ij} = ||(g_i^{(K)} - g_j^{(K)})||_2^2$$

$$\widehat{sim}_{ij} = \exp(-\lambda \cdot dis_{ij})$$

$$\mathcal{L}_{sim} = \sum_{i,j=1}^{m} sim_{ij} \cdot ||\widehat{sim}_{ij} - sim_{ij}||_2^2 \tag{7}$$

Online E-Commerce Fraud: A Large-scale Detection and Analysis

电商领域中，搜索引擎成为的流量的主要入口，也是商家最为主要的盈利来源。于是恶意商户就通过虚假访问，浏览，购买等方法来提升自己的搜索结果

AnTi-Fraud system (ATF)

three components:
preprocessor
Graph-Based Detection module (GBD)
Time Series based Detection module (TSD).

GBD emphasizes detecting fraud items that are promoted using similar strategies with known patterns,
while TSD focuses on detecting new fraud items and those are generated following new promotion strategies

GBD

Seed Identification.

The first step of GBD is to identify a small set of dishonest users

Fraud Propagation.

$p(I_i)$ item的欺诈评分

$p(U_i)$ user的不诚实评分

(1) $\forall U_i \in S, p(U_i) = 1$

(2) $\forall U_i \in U \setminus S, p(U_i) = 0$

(3) $\forall I_i \in I, p(I_i) = 0$

欺诈用户评分为1，其余用户评分为0
所有item评分为0

根据二部图，迭代的计算item得分(HITS)算法。

$$p(I_j) = \frac{\sum_{E_{ij} \in E} W_{ij}^U \times p(U_i)}{\sum_{E_{ij} \in E} W_{ij}^U}, \quad (1)$$

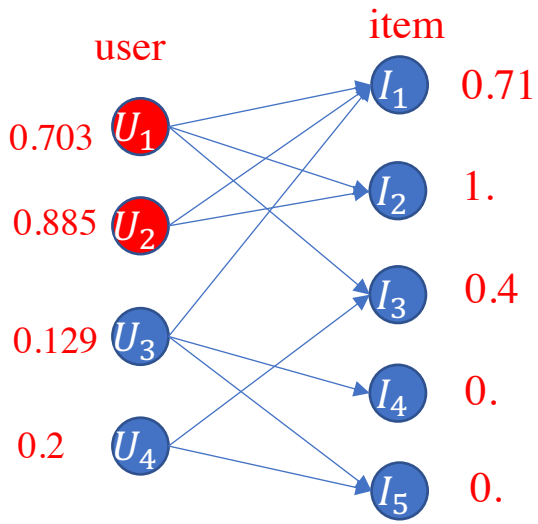
$$p(U_i) = \frac{\sum_{E_{ij} \in E} W_{ij}^I \times p(I_j)}{\sum_{E_{ij} \in E} W_{ij}^I}, \quad (2)$$

更新规则

$W_{11}^I = 3/4$

$W_{21}^I = 2/4$

$$W_{ij}^U = \frac{1}{\sum_{I_j \in I} \delta((U_i, I_j) \in E)}, \quad W_{ij}^I = \frac{1}{\sum_{U_i \in U} \delta((U_i, I_j) \in E)},$$



$U_1 = 0$ $I_1 = 0$ $W_{11}^U = 1/3$ $W_{11}^I = 1/3$

$U_2 = 1$ $I_2 = 0$ $W_{12}^U = 1/3$ $W_{21}^I = 1/3$

$U_3 = 0$ $\cdots$ $W_{14}^U = 0$ $W_{31}^I = 1/3$

$U_4 = 1$ $I_5 = 0$ $W_{15}^U = 0$ $W_{41}^I = 0$

更新$P(I_1)$

$$P(I_1) = \frac{W_{11}^U * P(U_1) + \cdots + W_{41}^U * P(U_4)}{W_{11}^U + W_{21}^U + W_{31}^U + W_{41}^U}$$

0.71

$$= \frac{1/3 * 1 + 0.5 * 1 + 1/3 * 0 + 0 * 0}{1/3 + 0.5 + 1/3 + 0}$$

更新$P(U_1)$

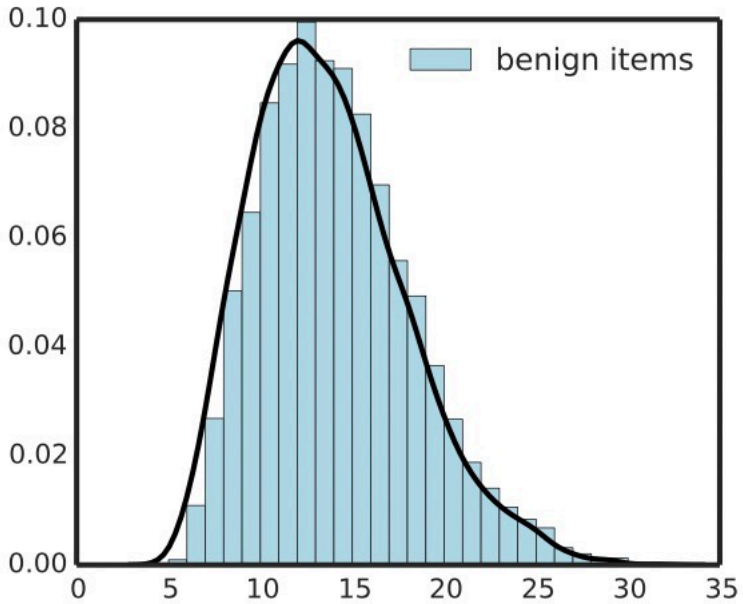
$$P(U_1) = \frac{W_{11}^U * P(I_1) + \cdots + W_{51}^U * P(I_5)}{W_{11}^U + W_{21}^U + W_{31}^U + W_{41}^U + W_{51}^U}$$

0.703

$$= \frac{1/3 * 0.71 + 0.5 * 1 + 0.5 * 0.4 + 0 * 0 + 0 * 0}{1/3 + 0.5 + 0.5 + 0 + 0}$$

3) Time Series based Detection (TSD):

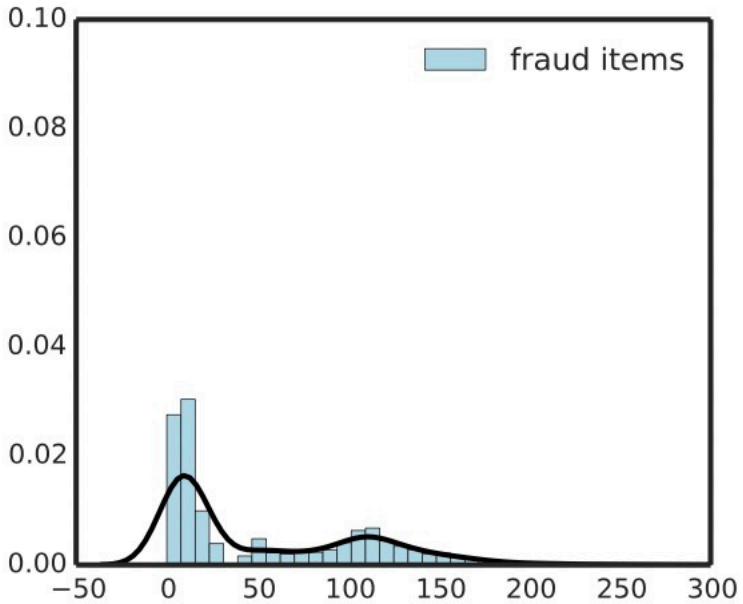
当某个item发生欺诈时，作者认为其时间序列分布会和正常用户有所区别。



(a) T^b's distribution

正常item的点击时间序列分布

服从一个泊松分布



(b) T^f's distribution

欺诈item的点击时间序列分布

欺诈item包含正常用户和欺诈用户，所以点击行为服从两个泊松分布

Traffic Time Series Modeling.

first make a hypothesis that T_i *follows the mixture of two Poisson distributions* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *with parameters* $(\lambda_1, \lambda_2, \pi_1, \pi_2)$, *where* λ_1 *and* λ_2 *are the mean values of* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *respectively,* π_1 *and* π_2 *indicate the mixture ratios of* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *respectively, and* $\pi_1 + \pi_2 = 1$. Now, for I_i, we define a *latent*

$Z_i = \{Z_i^1, Z_i^2, \cdots, Z_i^n\}$ $Z_i^j \in \{1, 2\}$ $\mathcal{P}_1$ *and* $\mathcal{P}_2$

$$L(\lambda_1, \lambda_2 | T_i) = \sum_{V_i^j(t) \in T_i} \log \sum_{k=1}^{2} \pi_k p(V_i^j(t) | \lambda_k), \quad (3)$$

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, k = 0, 1, \cdots$$

概率密度函数

$$L(\lambda_1, \lambda_2 | T_i) = \sum_{V_i^j(t) \in T_i} \log \sum_{k=1}^{2} \pi_k \frac{\lambda_k^{V_i^j(t)}}{V_i^j(t)!} e^{-\lambda_k}. \quad (4)$$

Our goal here is to find such $(\lambda_1^\star, \lambda_2^\star, \pi_1^\star, \pi_2^\star)$

$$L(\lambda_1, \lambda_2 | T_i) = \sum_{V_i^j(t) \in T_i} \log \sum_{k=1}^{2} \pi_k \frac{\lambda_k^{V_i^j(t)}}{V_i^j(t)!} e^{-\lambda_k}. \quad (4)$$

$$L(\lambda_1, \lambda_2 | T_i) \geqslant \sum_{V_i^j(t) \in T_i} \sum_{k=1}^{2} Q_i(Z_i^j = k) \log \frac{\pi_k \frac{\lambda_k^{V_i^j(t)}}{V_i^j(t)!} e^{-\lambda_k}}{Q_i(Z_i^j = k)},$$

Our goal here is to find such $(\lambda_1^\star, \lambda_2^\star, \pi_1^\star, \pi_2^\star)$

使用EM算法进行参数评估：

1）初始化$\lambda_1, \lambda_2, \pi_1, \pi_2$

a) E-step.

i来自于k的可能性

$$Q_i^m(Z_i^j = k) = \frac{\pi_k^{m-1} \frac{\lambda_k^{m-1}}{V_i^j(t)!} e^{-\lambda_k^{m-1}}}{\sum_{k=1}^{2} \pi_k^{m-1} \frac{\lambda_k^{m-1}}{V_i^j(t)!} e^{-\lambda_k^{m-1}}}. \quad (6)$$

b) *M-step*. Update π_k and λ_k as

k类别

$$\pi_k^m = \frac{\sum_{V_i^j(t) \in T_i} Q_i^m(Z_i^j = k)}{n}, \quad (7)$$

$$\lambda_k^m = \frac{\sum_{V_i^j(t) \in T_i} Q_i^m(Z_i^j = k) \times V_i^j(t)}{\sum_{V_i^j(t) \in T_i} Q_i^m(Z_i^j = k)}. \quad (8)$$

最终异常得分评估

abnormal score of I_i as $\left| \frac{\lambda_2^\star - \lambda_1^\star}{\max\{\lambda_1^\star, \lambda_2^\star\}} \right|$

TABLE II
PERFORMANCE OF ATF.

	Precision	Recall	F-score
D_1	0.9764	0.9785	0.9774
D_2	0.9749	0.9872	0.9810

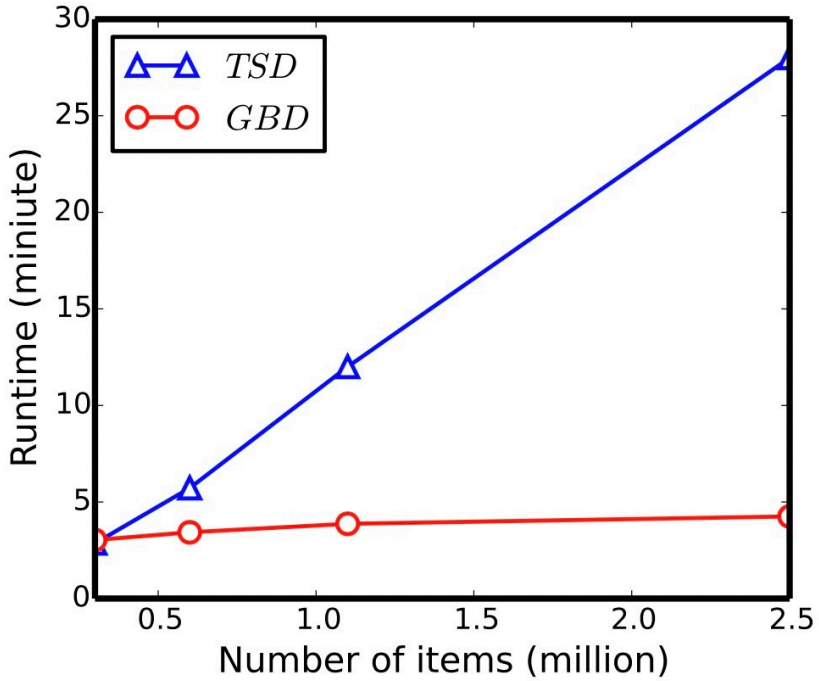


Fig. 4. Runtime v.s. number of items.

训练时间

TABLE III
RUNNING RESULTS OF ATF ON TAOBAO

Category	GBD (%)	TSD (%)	overlap (%)
men's clothing	92.26%	9.99%	2.26%
women's clothing	74.84%	27.49%	2.33%
men's shoes	92.6%	9.41%	2.01%
women's shoes	77.94%	25.12%	3.06%
computer & office	97.66%	3.12%	0.7%
phone & accessories	88.91%	12.68%	1.60%
food & grocery	77.57%	23.56%	1.14%
sports & outdoors	89.08%	12.68%	1.76%

Code

```
def update_PI(self):
    # get PI  = (WU*PU) / WI : matrix multiplication for all value of PI
    # import pdb; pdb.set_trace()
    numerator = np.asarray(np.dot(self.get_matrix_WU(), self.PU)) # (I*U) * (U*1) = (I*1)
    denominator = sum(self.get_matrix_WU().T).T # I*1
    self.PI = np.asmatrix(np.where(denominator == 0, 0, numerator / denominator))
    return # I*1 matrix
```

$\sum W_{ij}^{U} * p(U_i)$

Item可疑程度

$$p(I_j) = \frac{\sum_{E_{ij} \in E} W_{ij}^{U} \times p(U_i)}{\sum_{E_{ij} \in E} W_{ij}^{U}}, \qquad (1)$$

$$W_{ij}^{U} = \frac{1}{\sum_{I_i \in I} \delta((U_i, I_j) \in E)},$$

```
def get_matrix_WU(self):    self: <__main__.Bipartite_Graph object at 0x7f9effb1e0b8>
    return np.asmatrix(np.asarray(np.where(sum(self.M.T) ==0, sum(self.M.T), 1/sum(self.M.T)))*np.asarray(self.M.T))
```

```
In[20]: self.M.T
Out[20]: user
matrix([[1., 1., 1., 0.],
        [1., 1., 0., 0.],
        [1., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 1., 1.]])
```

item

WU:

```
matrix([[0.33333333, 0.5       , 0.33333333, 0.        ],
        [0.33333333, 0.5       , 0.        , 0.        ],
        [0.33333333, 0.        , 0.        , 0.5       ],
        [0.        , 0.        , 0.33333333, 0.        ],
        [0.        , 0.        , 0.33333333, 0.5       ]])
```

$$W_{ij}^{I} = \frac{1}{\sum_{U_i \in U} \delta((U_i, I_j) \in E)},$$ 这个item被user购买的情况

```
def get_matrix_WI(self):  self: <__main__.Bipartite_Graph object at 0x7f9effb1e0b8>
    return np.asmatrix(np.asarray(np.where(sum(self.M) ==0, sum(self.M), 1/sum(self.M)))*np.asarray(self.M))
```

```
In[18]: self.M
Out[18]: item
matrix([[1., 1., 1., 0., 0.],
   user   [1., 1., 0., 0., 0.],
          [1., 0., 0., 1., 1.],
          [0., 0., 1., 0., 1.]])
```

WI 对item进行归一化

```
matrix([[0.33333333, 0.5       , 0.5       , 0.        , 0.        ],
        [0.33333333, 0.5       , 0.        , 0.        , 0.        ],
        [0.33333333, 0.        , 0.        , 1.        , 0.5       ],
        [0.        , 0.        , 0.5       , 0.        , 0.5       ]])
```

```
In[20]: self.M.T
Out[20]: user
matrix([[1., 1., 1., 0.],
   item    [1., 1., 0., 0.],
           [1., 0., 0., 1.],
           [0., 0., 1., 0.],
           [0., 0., 1., 1.]])
```

WU: 对user进行归一化

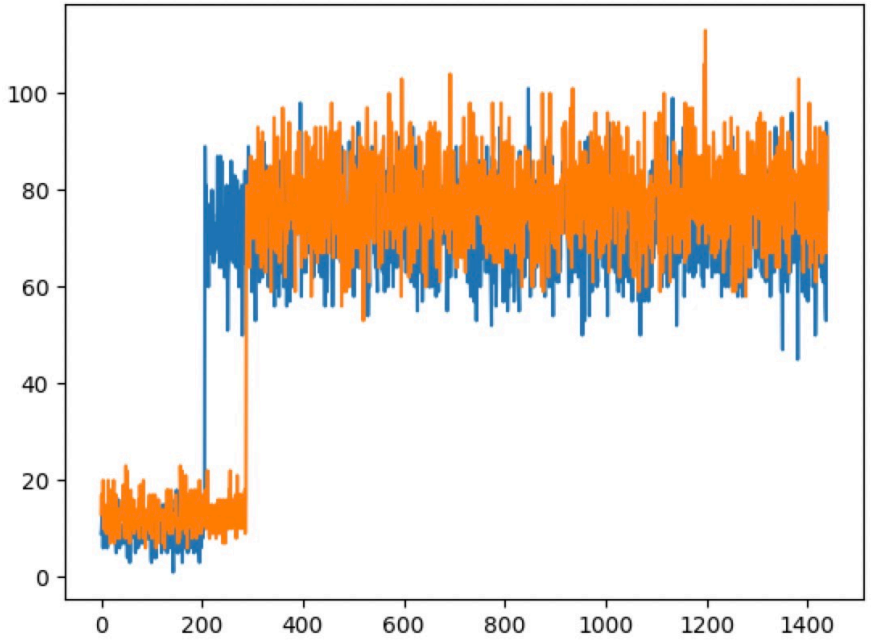
```
matrix([[0.33333333, 0.5       , 0.33333333, 0.        ],
        [0.33333333, 0.5       , 0.        , 0.        ],
        [0.33333333, 0.        , 0.        , 0.5       ],
        [0.        , 0.        , 0.33333333, 0.        ],
        [0.        , 0.        , 0.33333333, 0.5       ]])
```

```
def update_PU(self):  self:
    # import pdb; pdb.set_trace()
    numerator = np.asarray(np.dot(self.get_matrix_WI(), self.PI)).T  # (U*I) * (I*1)
    denominator = sum(self.get_matrix_WI().T)  # 行求和; 用户求和
    self.PU = np.asmatrix(np.where(denominator == 0, 0, numerator / denominator).T)
    return #U*1 matrix
```

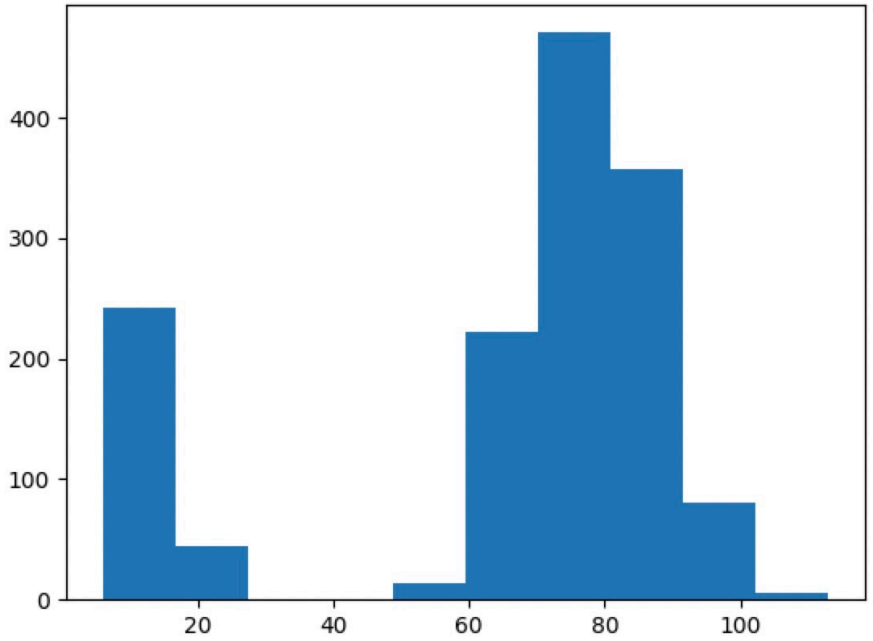
$$\sum W_{ij}^{I} * p(I_j)$$

每种颜色表示是每个item的点击时间序列，点击时间序列服从两个泊松分布

```
In[2]: T[0]                                    1440个时间点
Out[2]: array([18, 20, 12, ..., 60, 46, 46])
In[3]: T[1]
Out[3]: array([11, 15, 11, ..., 27, 28, 30])
```



两个item，随时间的点击次数



每个时间点击次数分布情况

计算每个item属于每个泊松分布的概率

```
def cal_Q1(T,lambda_1,pi_1,lambda_2,pi_2):  T:    lambda_1: [13 62 78 31 93 76 23 44 86 65 82 20 76 7
    return (cal_pdf_1(T,lambda_1,pi_1)) / ((cal_pdf_1(T,lambda_1,pi_1) + cal_pdf_2(T,lambda_2,pi_2)))
```

计算泊松分布

```
def cal_pdf_1(T,lambda_1, pi_1):  T:    la
    return poission_pdf(T,lambda_1, pi_1)
```

$$Q_i^m(Z_i^j = k) = \frac{\pi_k^{m-1} \frac{\lambda_k^{m-1}}{V_i^j(t)!} e^{-\lambda_k^{m-1}}}{\sum_{k=1}^{2} \pi_k^{m-1} \frac{\lambda_k^{m-1}}{V_i^j(t)!} e^{-\lambda_k^{m-1}}}. \qquad (6)$$

```
def poission_pdf(T, lambda_k, pi_k):  T: [[12  7  5 ... 31 3
    s1 = np.power(lambda_k.reshape(1000,1),T)/factorials(T)
    s2 = math.e**(-lambda_k).reshape(1000,1)  # e-lambda
    return s1*s2
```

$$\frac{\lambda_k^{m-1}}{V_i^j(t)!}$$

每个商户对应到的商品点击次数，都服从两个泊松分布，并使用EM算法进行参数估计。如果估计出来的λ_1和λ_2的差距比较大，则说明该商品在购买上出现异常的波动情况。（个人认为也可以作为异常检测方法）

初始的lambda和pi随机定义的

```
lambda_1 = np.asarray([rd.randrange(1,100) for i in range(number_fraud_item)])  lambda_1: [42 87 59
lambda_2 = np.asarray([rd.randrange(1,100) for i in range(number_fraud_item)])  lambda_2: [47 24 25
pi_1 = np.asarray([rd.uniform(0,1) for i in range(number_fraud_item)]).reshape(number_fraud_item,1)
pi_2 = np.asarray([1 for i in range(number_fraud_item)]).reshape(number_fraud_item,1) - pi_1  # 1-p
```

泊松分布只有一个参数λ，但是我们是两个泊松分布，所以还有另外一个参数π

```
def poission_pdf(T, lambda_k, pi_k):  T: [[ 4  8 10 ... 30 18
    s1 = lambda_k.reshape(1000,1)/factorials(T)  # 阶乘; V(t)!
    s2 = math.e**(-lambda_k).reshape(1000,1)  # e-lambda  s2:
    return s1*s2
```

$$P(X=k)=\frac{\lambda^k}{k!}e^{-\lambda},k=0,1,\cdots$$

λ: 历史均值(参数)

$$P(X=2)=\frac{\lambda^k}{k!}e^{-\lambda}=\frac{5^2}{2!}e^{-5}=0.084$$

K:本次观测值

$$Q_i^m(Z_i^j=k)=\frac{\pi_k^{m-1}\frac{\lambda_k^{m-1}}{V_i^j(t)!}e^{-\lambda_k^{m-1}}}{\sum_{k=1}^{2}\pi_k^{m-1}\frac{\lambda_k^{m-1}}{V_i^j(t)!}e^{-\lambda_k^{m-1}}}. \quad (6)$$

```
def cal_Q1(T,lambda_1,pi_1,lambda_2,pi_2):
    return (cal_pdf_1(T,lambda_1,pi_1) * pi_1) / ((cal_pdf_1(T,lambda_1,pi_1) * pi_1 + cal_pdf_2(T,lambda_2,pi_2) * pi_2))
```

分子

分母

```
for i in range(10000):
    count+=1
    pi_1 = cal_pi_1(Q1_temp)
    pi_2 = cal_pi_2(Q2_temp)
    lambda_1 = cal_lambda_1(Q1_temp)
    lambda_2 = cal_lambda_2(Q2_temp)
    Q_1 = cal_Q1(T,lambda_1,pi_1,lambda_2,pi_2)
    Q_2 = cal_Q2(T,lambda_1,pi_1,lambda_2,pi_2)
```

循环迭代

1000个序列更新后的pi值

```
def cal_pi_1(Q_1):  Q_1: [[9.62141549e-31 9.62141549e-31 9.621
    return (sum(Q_1.T) / sum(Q_1.T).shape[0]).reshape(1000,1)
```

(1000, 1440)

1000序列，每个序列1440个时间点

$$\pi_k^m = \frac{\sum_{V_i^j(t)\in T_i} Q_i^m(Z_i^j = k)}{n}, \tag{7}$$

```
def cal_lambda_1(Q_1):  Q_1: [[9.6214
    return sum((Q_1*T).T)/sum(Q_1.T)
```

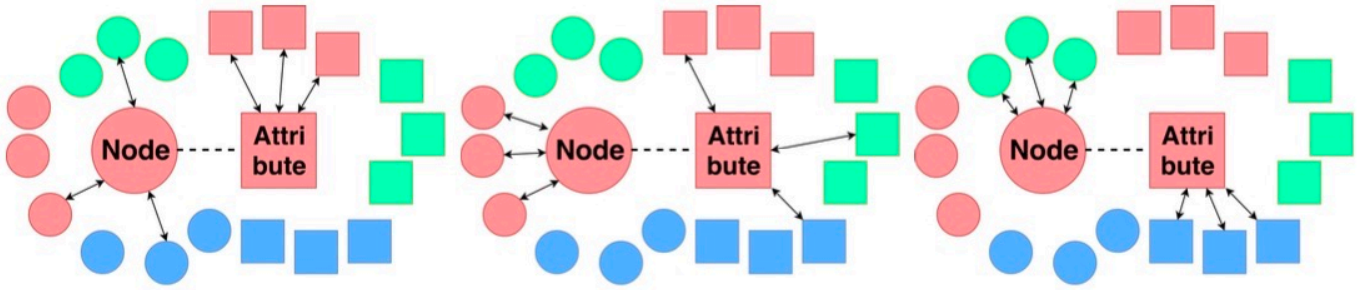
$$\lambda_k^m = \frac{\sum_{V_i^j(t)\in T_i} Q_i^m(Z_i^j = k) \times V_i^j(t)}{\sum_{V_i^j(t)\in T_i} Q_i^m(Z_i^j = k)}. \tag{8}$$

Outlier Aware Network Embedding for Attributed Networks

当图网络结构和属性一致，现有的属性化网络方法可以很好地工作。但现实世界的网络往往有异常节点，
这些异常的节点属性和结构之间存在异常，因此，导致了下游的网络挖掘任务都使用这些离群值生成的embedding。
作者在生成节点embedding的同时，也将正常和异常的进行了区分，以保证正常点embedding不受异常点的影响。

异常Outlier类别

大的圆和正方形为异常



(a) Structural Outlier

节点连接着其余类别的节点

节点的属性是红色方块，
但其连接着其余颜色的节点

(b) Attribute Outlier

节点属于一类社区，
但是属性属于多个类别

节点属于红色圆形节点，
但其属性却属于多个类别

(c) Combined Outlier

节点属于某个社区内，但是属性属于另一个社区

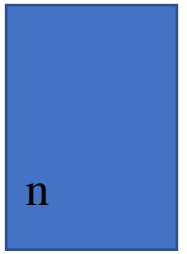
节点所属类别和属性不一致

4 Solution Approach: ONE

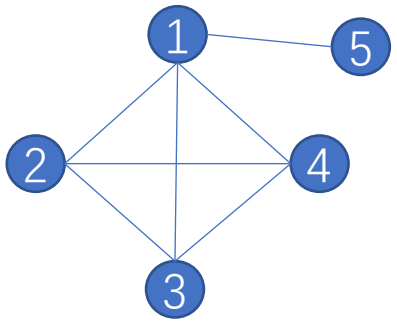
4.1 Learning from the Link Structure

初始的G,H向量 $A \approx GH$

Link Structure就是考虑节点和节点之间如果连接比较紧密，则应该有相似的embedding。
如果一个节点属于结构上的异常节点，那么在进行矩阵分解后并还原后，这个点相比较于原始结构的差距比较大。

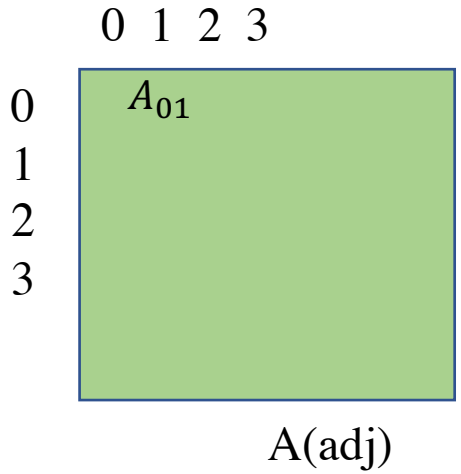
$G_{i.} =$ 

$H_{.j} =$ 



A = [[0 1 1 1 1]
[1 0 1 1 0]
[1 1 0 1 0]
[1 1 1 0 0]
[1 0 0 0 0]]

$G \cdot H =$ [[0. 1.11 1.04 1.04 0.7]
[1.01 0. 1. 1. 0.]
[1.04 0.88 0. 0.96 0.32]
[1.04 0.88 0.95 0. 0.32]
[0.89 0.3 0.05 0.05 0.]]



4 Solution Approach: ONE

4.1 Learning from the Link Structure

$$\sum_{i=1}^{N}\sum_{j=1}^{N}(A_{ij}-G_{i\cdot}\cdot H_{\cdot j})^2,$$

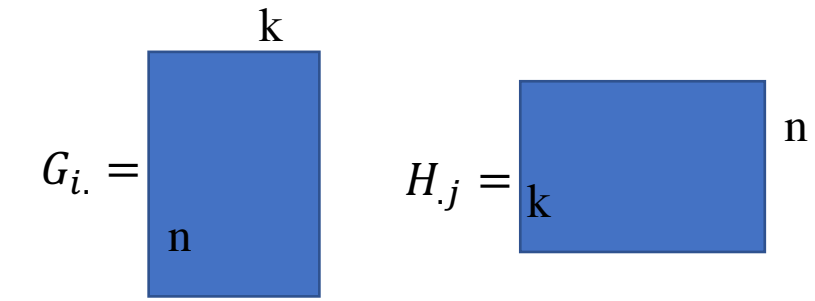
两个向量的内积

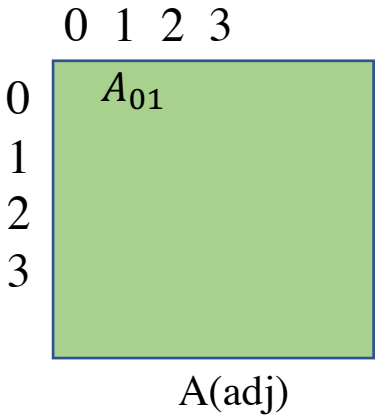
邻接矩阵 $A_{01}=1$

节点i的行向量 $G_{0\cdot}=[[0.2,0.3,1]]$

j节点列向量 $H_{\cdot 1}=[[1,2,0.1]]^T$

初始的G,H向量 $A \approx GH$





如果单纯这么做的话，异常节点和正常节点之间会趋于中间值，那么不满足寻找异常的需求

O_{1i}: 节点i的结构异常值，该值越大，对loss影响越小，说明异常性越明显

$$\mathcal{L}_{str}=\sum_{i=1}^{N}\sum_{j=1}^{N}\log\left(\frac{1}{O_{1i}}\right)(A_{ij}-G_{i\cdot}\cdot H_{\cdot j})^2 \qquad (1)$$

$$0<O_{1i}\leq 1$$

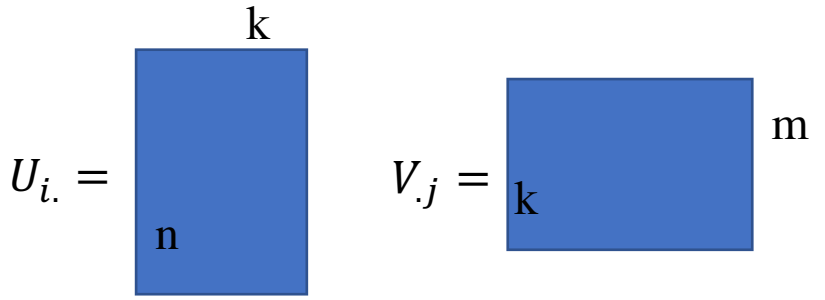
$$\sum_{i=1}^{N}O_{1i}=\mu$$

同时我们保证O_{1i}和为常数，
否则，所有的O_{1i}都会变为1

SVD分解后，如果是正常节点，那么$G_{i\cdot}\cdot H_{\cdot j}$结构应该和$A_{ij}$比较接近，
但是对于异常节点$G_{i\cdot}\cdot H_{\cdot j}$结构应该和$A_{ij}$比较接远，所以$O_{1i}$值变得比较大，才能使的loss最小

4.2 Learning from the Attributes

初始的U, V向量　　$C \approx UV$

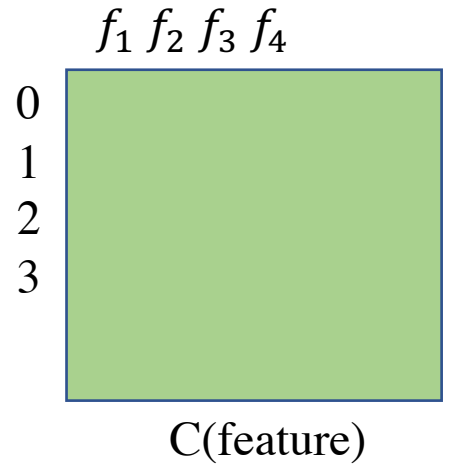


类似于结构优化，我们使用同样的方法去优化属性向量

$C_{i.} - v_i$的属性向量

$U \in \mathbb{R}^{N \times K}$ and $V \in \mathbb{R}^{K \times D}$

U表示网络嵌入　　V节点特征



O_{2i} – 节点属性异常分值

$$\mathcal{L}_{attr} = \sum_{i=1}^{N}\sum_{d=1}^{C} \log\left(\frac{1}{O_{2i}}\right)(C_{id} - U_{i\cdot} \cdot V_{\cdot d})^2 \qquad (2)$$

$$0 < O_{2i} \leq 1$$

$$\sum_{i=1}^{N} O_{2i} = \mu$$

4.3 Connecting Structure and Attributes 结合节点的属性和结构信息

上两步的做法，分别计算结构和属性特征。但是属性和结构特性应该是高度相关的(因为其都表示同一个节点)，正常节点(绝大部份节点)通过属性特征是可以通过映射到结构特性的(反之亦然)。
并且我们最终的目的是需要得到的是一个节点的embedding。

$$\sum_{i=1}^{N}\sum_{k=1}^{K}(G_{ik}-U_{ik})^2$$

G-结构 U-属性

一种最简单的方法是求之前得到的两个向量的距离，
但是这两个向量是单独训练的，他们之间是没有对齐的。

$$\mathcal{L}_{dis}=\sum_{i=1}^{N}\sum_{k=1}^{K}\log\left(\frac{1}{O_{3i}}\right)\left(G_{ik}-U_{i\cdot}\cdot(W^T)_{\cdot k}\right)^2 \quad (4)$$

G-结构 U-属性

通过W将节点属性特征映射为结构特征

O_{3i} – 结构和属性不一致异常分值

注意：W矩阵并不是一个可学习的参数，
而是一个根据矩阵分解得到的向量

Here the new matrices are defined as, $(\bar{G})_{i,k} = \sqrt{\log\left(\frac{1}{O_{3i}}\right)}G_{ik}$ and $\bar{U}_{ik} = \sqrt{\log\left(\frac{1}{O_{3i}}\right)}U_{ik}$. Say, $X\Sigma Y^T = \text{SVD}(\bar{G}^T\bar{U})$, then W can be obtained as:

$$\boxed{W = XY^T} \quad (11)$$

Procrustes problem

Procrustes这个名字来自于希腊神话中的一个强盗，他通过伸展受害者的四肢或将他们砍下使他们适合自己的床。

The **orthogonal Procrustes problem** [1] is a matrix approximation problem in linear algebra. In its classical form, one is given two matrices A and B and asked to find an orthogonal matrix Ω which most closely maps A to B. [2] Specifically,

单位正交矩阵

$$R = \arg\min_{\Omega} \|\Omega A - B\|_F \quad \text{subject to} \quad \Omega^T\Omega = I,$$

Solution [edit]

This problem was originally solved by Peter Schönemann in a 1964 thesis, and shortly after appeared in the journal Psychometrika.[3]

This problem is equivalent to finding the nearest orthogonal matrix to a given matrix $M = BA^T$, i.e. solving the *closest orthogonal approximation problem*

$$\min_{R} \|R - M\|_F \quad \text{subject to} \quad R^TR = I.$$

To find matrix R, one uses the singular value decomposition (for which the entries of Σ are non negative)

$$M = U\Sigma V^T$$

SVD

to write

$$\Omega R = UV^T.$$

人脸对其场景

$$\mathcal{L}_{dis} = \sum_{i=1}^{N}\sum_{k=1}^{K} \log\left(\frac{1}{O_{3i}}\right)\left(G_{ik} - U_{i\cdot}\cdot(W^T)_{\cdot k}\right)^2 \qquad (4)$$

将结构特征矩阵和属性特征矩阵对其

https://en.wikipedia.org/wiki/Orthogonal_Procrustes_problem

$$\mathcal{L}_{dis} = \sum_{i=1}^{N}\sum_{k=1}^{K} \log\left(\frac{1}{O_{3i}}\right)\left(G_{ik} - U_{i\cdot} \cdot (W^T)_{\cdot k}\right)^2$$

Here the new matrices are defined as, $(\bar{G})_{i,k} = \sqrt{\log\left(\frac{1}{O_{3i}}\right)} G_{ik}$ and $\bar{U}_{ik} = \sqrt{\log\left(\frac{1}{O_{3i}}\right)} U_{ik}$. Say, $X\Sigma Y^T = \text{SVD}(\bar{G}^T\bar{U})$, then W can be obtained as:

$$\boxed{W = XY^T} \qquad (11)$$

$$\mathcal{L} = argmin||\Omega A - B||_F$$
$$= argmin(\Omega A - B)^2$$
$$= argmin(B - \Omega A)^2$$
$$= argmin(B^T - A^T\Omega^T)^2$$

$$M = BA^T$$
$$SVD(M) = U\Sigma V^T$$
$$\Omega = UV^T$$

$$\mathcal{L}_{dis} = argmin(G - UW^T)^2$$
$$B^T = G$$
$$A^T = U$$
$$\Omega^T = W^T$$

$$M = G^T U$$
$$X\Sigma Y = SVD(M) = SVD(G^T U)$$
$$W = XY^T$$

4.4 Joint Loss Function

$$\mathcal{L} = \mathcal{L}_{str} + \alpha\mathcal{L}_{attr} + \beta\mathcal{L}_{dis} \tag{5}$$

$$0 < O_{1i}, O_{2i}, O_{3i} \leq 1 \ , \ \forall v_i \in V$$

$$\sum_{i=1}^{N} O_{1i} = \sum_{i=1}^{N} O_{2i} = \sum_{i=1}^{N} O_{3i} = \mu$$

$$W \in \mathcal{O}_K \iff W^T W = \mathcal{I}$$

如果节点i在结构中异常，则O_{1i}的值很大，则导致G_i优化不完全，同样会导致优化G_i和U_i相关性性能下降，同样导致O_{3i}变大。
所以我们最后会$O_{1i}O_{2i}O_{3i}$的值很大，则以为异常点。

4.6 Updating G, H, U, V

更新结构特征向量G, H

$$\frac{\partial \mathcal{L}}{\partial G_{ik}} = 0 \Rightarrow \sum_{j=1}^{N} \log\left(\frac{1}{O_{1i}}\right)(A_{ij} - G_{i\cdot} \cdot H_{\cdot j})(-H_{kj}) + \log\left(\frac{1}{O_{3i}}\right)(G_{ik} - U_{i\cdot} \cdot (W^T)_{\cdot k}) = 0$$

for G_{ik}

$$G_{ik} = \frac{G_{ik}^{num1} + \beta \log\left(\frac{1}{O_{3i}}\right)(W_{k\cdot} \cdot U_{i\cdot})}{\log\left(\frac{1}{O_{1i}}\right)(H_{k\cdot} \cdot H_{k\cdot}) + \beta \log\left(\frac{1}{O_{3i}}\right)}$$

$$G_{ik}^{num1} = \log\left(\frac{1}{O_{1i}}\right) \sum_{j=1}^{N} (A_{ij} - \sum_{k' \neq k} G_{ik'} H_{k'j}) H_{kj}$$

$$H_{kj} = \frac{\sum_{i=1}^{N} \log\left(\frac{1}{O_{1i}}\right)(A_{ij} - \sum_{k' \neq k} G_{ik'} H_{k'j}) G_{ik}}{\sum_{i=1}^{N} \log\left(\frac{1}{O_{1i}}\right) G_{ik}^2}$$

(7)

更新属性特征向量U, V $\mathcal{L} = \alpha\mathcal{L}_{str} + \beta\mathcal{L}_{attr} + \gamma\mathcal{L}_{dis}$

$$U_{ik} = \frac{U_{ik}^{num1} + U_{ik}^{num2}}{\beta \log\left(\frac{1}{O_{2i}}\right)(V_{k\cdot} \cdot V_{k\cdot}) + \gamma \log\left(\frac{1}{O_{3i}}\right) W_{\cdot k} \cdot W_{\cdot k}}$$

$$U_{ik}^{num1} = \beta \log\left(\frac{1}{O_{2i}}\right) \sum_{d=1}^{D} (C_{id} - \sum_{k' \neq k} U_{ik'} V_{k'd}) V_{kd}$$

$$U_{ik}^{num2} = \gamma \log\left(\frac{1}{O_{3i}}\right)\left(G_{i\cdot} - (U_{i\cdot} W) - (U_{ik} \cdot W_{\cdot k})\right) W_{\cdot k}$$

$$V_{kd} = \frac{\sum_{i=1}^{N} \log\left(\frac{1}{O_{2i}}\right)(C_{id} - \sum_{k' \neq k} U_{ik'} V_{k'd}) U_{ik}}{\sum_{i=1}^{N} \log\left(\frac{1}{O_{2i}}\right) U_{ik}^2}$$

(9)

4.7 Updating W

$$\begin{aligned}\mathcal{L}_{dis} &= \sum_{i=1}^{N}\sum_{k=1}^{K}\log\left(\frac{1}{O_{3i}}\right)\left(G_{ik} - U_{i\cdot}\cdot(W^T)_{\cdot k}\right)^2 \\ &= \sum_{i=1}^{N}\sum_{k=1}^{K}\left(\bar{G}_{ik} - \bar{U}_{i\cdot}\cdot(W^T)_{\cdot k}\right)^2 \end{aligned} \tag{10}$$

$$(\bar{G})_{i,k} = \sqrt{\log\left(\frac{1}{O_{3i}}\right)}G_{ik}$$

$$\bar{U}_{ik} = \sqrt{\log\left(\frac{1}{O_{3i}}\right)}U_{ik}$$

$$X\Sigma Y^T = \text{SVD}(\bar{G}^T\bar{U})$$

$$\boxed{W = XY^T} \tag{11}$$

4.8 Updating O

constraint $\sum_{i=1}^{N} O_{1i} = \mu$

$$\frac{\partial}{\partial O_{1i}} \sum_{i,j} \log\left(\frac{1}{O_{1i}}\right)(A_{ij} - G_{i\cdot} \cdot H_{\cdot j})^2 + \lambda(\sum_{i} O_{1i} - \mu)$$

$$\boxed{O_{1i} = \frac{\left(\sum_{j=1}^{N}(A_{ij} - G_{i\cdot} \cdot H_{\cdot j})^2\right) \cdot \mu}{\sum_{i=1}^{N}\sum_{j=1}^{N}(A_{ij} - G_{i\cdot} \cdot H_{\cdot j})^2}} \quad (12)$$

$$\boxed{O_{2i} = \frac{\left(\sum_{d=1}^{D}(C_{id} - U_{i\cdot} \cdot V_{\cdot d})^2\right) \cdot \mu}{\sum_{i=1}^{N}\sum_{j=1}^{D}(C_{id} - U_{i\cdot} \cdot V_{\cdot d})^2}} \quad (13)$$

$$\boxed{O_{3i} = \frac{\left(\sum_{k=1}^{K}(G_{ik} - W_{i\cdot} \cdot U_{\cdot k})^2\right) \cdot \mu}{\sum_{i=1}^{N}\sum_{k=1}^{K}(G_{ik} - W_{i\cdot} \cdot U_{\cdot k})^2}} \quad (14)$$

4.9 Algorithm: ONE

G, H, U and V can be initialized by any standard matrix factorization technique

$$A \approx GH$$
$$C \approx UV$$

一个节点的最终离群值得分：可作为三个离群值得分的加权平均值。

一个节点的最终embedding：结构嵌入和转换后的属性嵌入的平均值

$$\frac{G_{i\cdot}+U_{i\cdot}(W^T)}{2}, \forall v_i \in V.$$

Algorithm 1 ONE

Input: The network $\mathcal{G} = (V, E, C)$, K: Dimension of the embedding space where $K < min(n, d)$, ratio parameter θ

Output: The node embeddings of the network G, Outlier score of each node $vinV$

NMF

1: Initialize G and H by standard matrix factorization on A, and U and V by that on C. $A \approx GH$ $C \approx UV$
2: Initialize the outlier scores O_1, O_2 and O_3 uniformly.
3: **for** until stopping condition satisfied **do**
4: Update W by Eq. 11. Procrustes
5: Update G, H, U and V by Eq. from 6 to 9.
6: Update outlier scores by Eq. from 12 to 14.
7: **end for**
output embedding
8: Embedding for the node v_i is $\frac{G_{i\cdot} + U_{i\cdot}(W^T)}{2}, \forall v_i \in V$.
9: Final outlier score for the node v_i is a weighted average of O_{1i}, O_{2i} and O_{3i}, $\forall v_i \in V$. Outlier score

$$\mathcal{L}_{str} = \sum_{i=1}^{N}\sum_{j=1}^{N} \log\left(\frac{1}{O_{1i}}\right)(A_{ij} - G_{i\cdot} \cdot H_{\cdot j})^2 \quad (1)$$

$$\mathcal{L}_{attr} = \sum_{i=1}^{N}\sum_{d=1}^{C} \log\left(\frac{1}{O_{2i}}\right)(C_{id} - U_{i\cdot} \cdot V_{\cdot d})^2 \quad (2)$$

$$\mathcal{L}_{dis} = \sum_{i=1}^{N}\sum_{k=1}^{K} \log\left(\frac{1}{O_{3i}}\right)\left(G_{ik} - U_{i\cdot} \cdot (W^T)_{\cdot k}\right)^2 \quad (4)$$

$$\boxed{W = XY^T} \quad (11)$$

$$\boxed{O_{1i} = \frac{\left(\sum_{j=1}^{N}(A_{ij} - G_{i\cdot} \cdot H_{\cdot j})^2\right) \cdot \mu}{\sum_{i=1}^{N}\sum_{j=1}^{N}(A_{ij} - G_{i\cdot} \cdot H_{\cdot j})^2}} \quad (12)$$

$$\boxed{O_{2i} = \frac{\left(\sum_{d=1}^{D}(C_{id} - U_{i\cdot} \cdot V_{\cdot d})^2\right) \cdot \mu}{\sum_{i=1}^{N}\sum_{j=1}^{D}(C_{id} - U_{i\cdot} \cdot V_{\cdot d})^2}} \quad (13)$$

$$\boxed{O_{3i} = \frac{\left(\sum_{k=1}^{K}(G_{ik} - W_{i\cdot} \cdot U_{\cdot k})^2\right) \cdot \mu}{\sum_{i=1}^{N}\sum_{k=1}^{K}(G_{ik} - W_{i\cdot} \cdot U_{\cdot k})^2}} \quad (14)$$

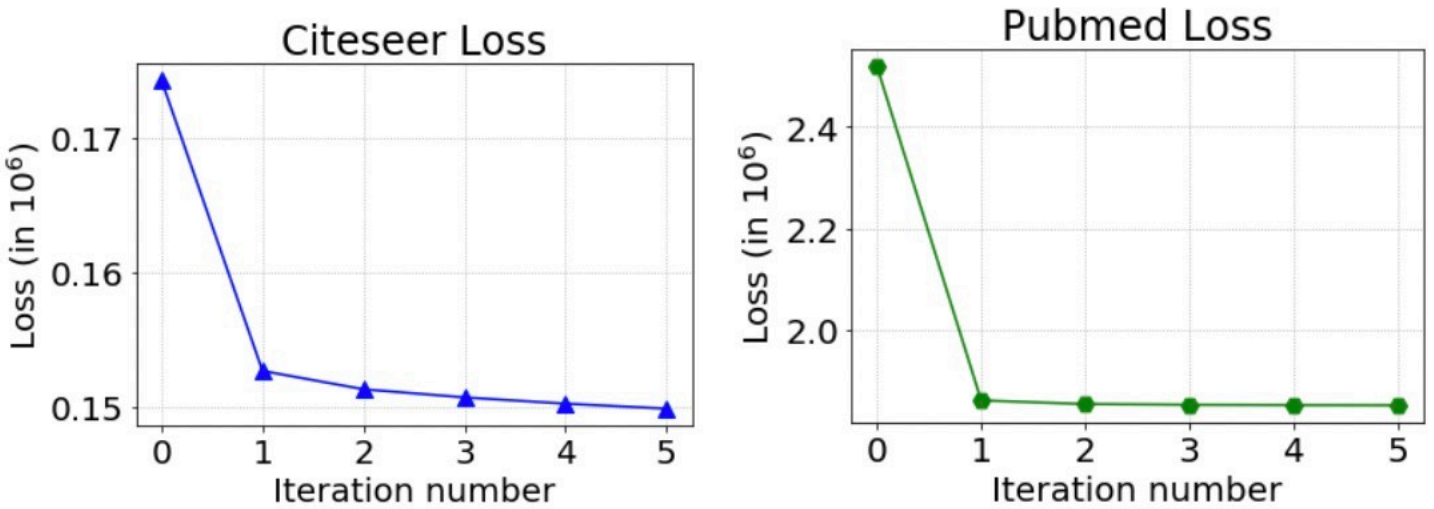


Figure 2: Values of Loss function over different iterations of ONE for Citeseer and Pubmed (seeded) datasets

迭代轮次和loss关系

算法收敛很快

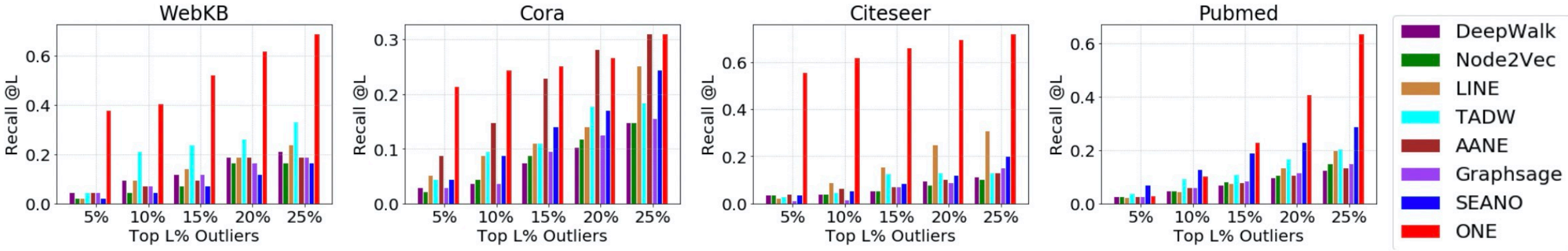


Figure 3: Outlier Recall at top L% from the ranked list of outliers for all the datasets. ONE, though it is an unsupervised algorithm, outperforms all the baseline algorithms in most of the cases. SEANO uses 20% labeled data for training.

离群点检测

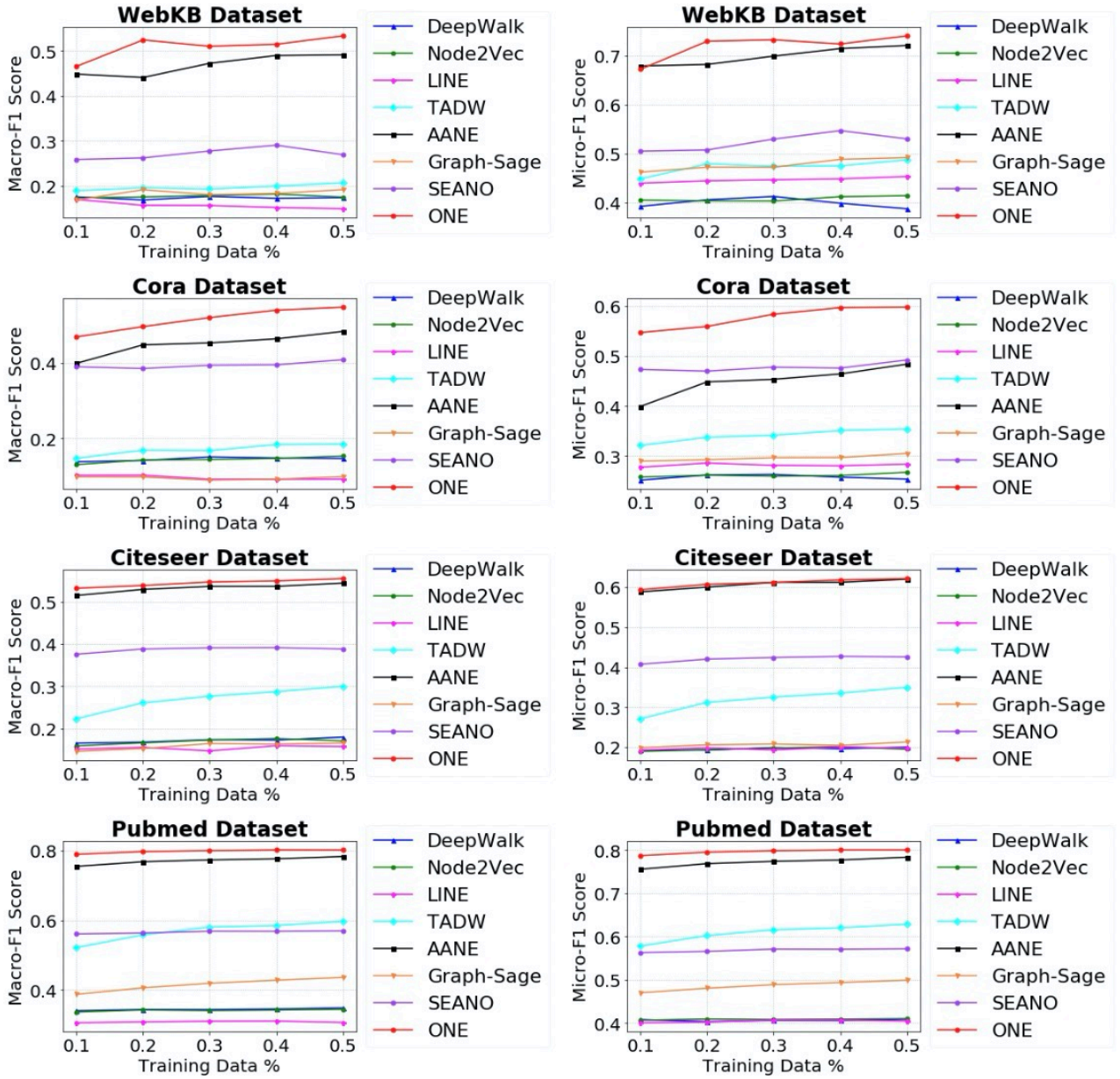
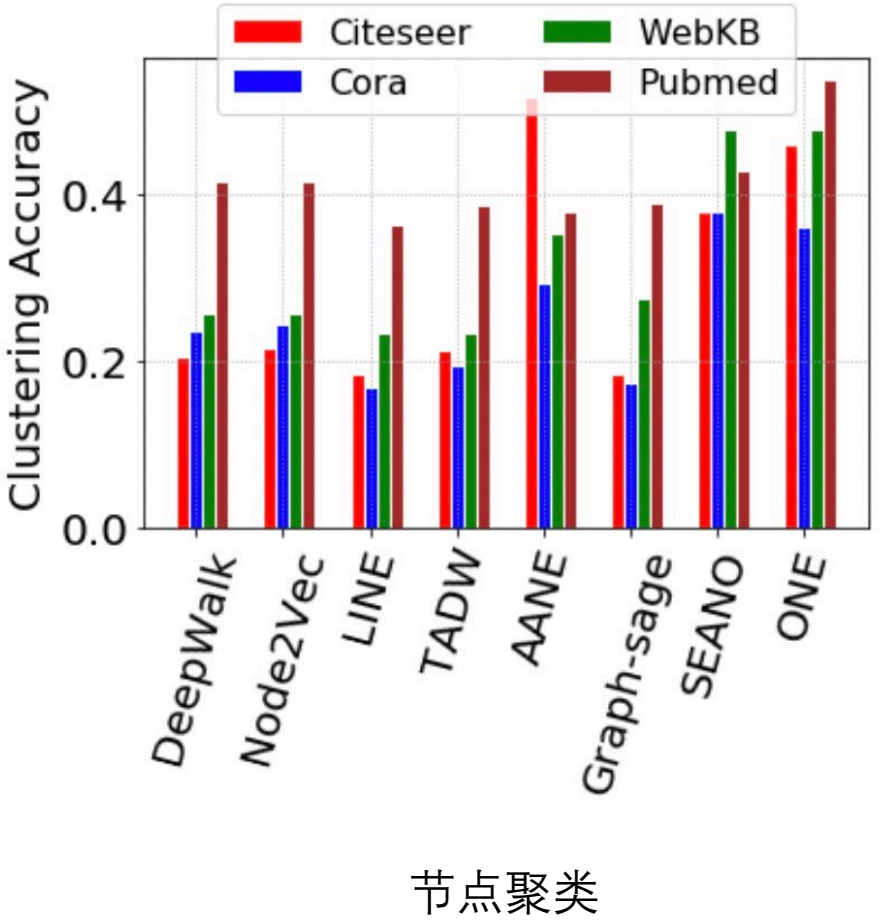


Figure 4: Performance of different embedding algorithms for Classification with Random Forest

节点分类



节点聚类

Code

NMF(Non-negative matrix factorization)，即对于任意给定的一个非负矩阵V，其能够寻找到一个非负矩阵W和一个非负矩阵H，满足条件V=W*H,从而将一个非负的矩阵分解为左右两个非负矩阵的乘积。

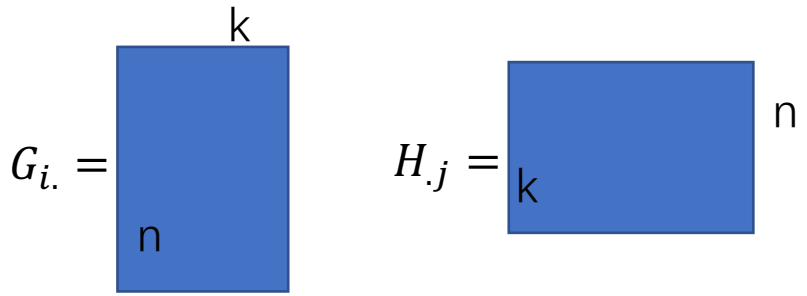
https://www.bilibili.com/video/BV1Wv411h7kN?p=59　　1:11:00会介绍NMF

```
model = NMF(n_components=K, init='random', random_state=0)  # 非负矩阵分解
G = model.fit_transform(A)  # 邻接矩阵
H = model.components_  # (12, 3477)
```

1: Initialize G and H by standard matrix factorization on A, and U and V by that on C.

$A \approx GH$

处理结构信息




```
model = NMF(n_components=K, init='random', random_state=0)
U = model.fit_transform(C)  # 特征(3477, 3703)分解; (3477, 12)
V = model.components_  # (12, 3703)
```

$C \approx UV$

处理特征信息

2: Initialize the outlier scores O_1, O_2 and O_3 uniformly.

```
outl1 = np.ones((A.shape[0]))
outl2 = np.ones((A.shape[0]))
outl3 = np.ones((A.shape[0]))
```

```
bet = nx.betweenness_centrality(Graph)
```

GNN第一课14:00

```
outl1 = outl1/sum(outl1)
outl2 = outl2/sum(outl2)
outl3 = outl3/sum(outl3)
```

$$\sum_{i=1}^{N} O_{1i} = \mu$$

$$\sum_{i=1}^{N} O_{2i} = \mu$$

$$\sum_{i=1}^{N} O_{3i} = \mu$$

```
temp1 = A - np.matmul(G,H)
temp1 = np.multiply(temp1,temp1)
temp1 = np.multiply( np.log(np.reciprocal(outl1)), np.sum(temp1, axis=1) )
temp1 = np.sum(temp1)
```

倒数　　行求和

$$\mathcal{L}_{str} = \sum_{i=1}^{N}\sum_{j=1}^{N} \log\left(\frac{1}{O_{1i}}\right)(A_{ij} - G_{i\cdot} \cdot H_{\cdot j})^2 \qquad (1)$$

```
temp2 = C - np.matmul(U,V)
temp2 = np.multiply(temp2,temp2)
temp2 = np.multiply( np.log(np.reciprocal(outl2)), np.sum(temp2, axis=1) )
temp2 = np.sum(temp2)
```

G:结构特征　　(12, 12)　(12, 3477)

```
temp3 = G.T - np.matmul(W, U.T)
temp3 = np.multiply(temp3,temp3)
temp3 = np.multiply( np.log(np.reciprocal(outl3)), np.sum(temp3, axis=0).T )
temp3 = np.sum(temp3)
```

U:属性特征

$$\mathcal{L}_{attr} = \sum_{i=1}^{N}\sum_{d=1}^{C} \log\left(\frac{1}{O_{2i}}\right)(C_{id} - U_{i\cdot} \cdot V_{\cdot d})^2 \qquad (2)$$

$$\mathcal{L}_{dis} = \sum_{i=1}^{N}\sum_{k=1}^{K} \log\left(\frac{1}{O_{3i}}\right)\left(G_{ik} - U_{i\cdot} \cdot (W^T)_{\cdot k}\right)^2 \qquad (4)$$

```
alpha = 1  alpha: 1
beta = temp1/temp2
gamma = min(2*beta, temp3)
```

$$\mathcal{L} = \mathcal{L}_{str} + \alpha\mathcal{L}_{attr} + \beta\mathcal{L}_{dis} \quad (5)$$

$$\mathcal{L} = \alpha\mathcal{L}_{str} + \beta\mathcal{L}_{attr} + \gamma\mathcal{L}_{dis}$$ 代码公式

更新G

邻接矩阵分解的G和H　　投影参数

```
calc_lossValues(A, C, G, H, U, V, W, outl1, outl2, outl3)
```

邻接矩阵　特征向量　特征向量分解的V和W

$$G_{ik}^{num1} = \log\left(\frac{1}{O_{1i}}\right)\sum_{j=1}^{N}(A_{ij} - \sum_{k' \neq k} G_{ik'}H_{k'j})H_{kj}$$

```
Gik_numer = alpha * np.log(np.reciprocal(outl1[i])) * np.dot(H[k,:], \
                (A[i,:] - (np.matmul(G[i], H) - np.multiply(G[i,k], H[k,:]))) ) + \
                gamma * np.log(np.reciprocal(outl3[i])) * np.dot(U[i], W[k,:])
```

对应分子

$$G_{ik} = \frac{G_{ik}^{num1} + \beta\log\left(\frac{1}{O_{3i}}\right)(W_{k\cdot} \cdot U_{i\cdot})}{\log\left(\frac{1}{O_{1i}}\right)(H_{k\cdot} \cdot H_{k\cdot}) + \beta\log\left(\frac{1}{O_{3i}}\right)}$$

```
Gik_denom = alpha * np.log(np.reciprocal(outl1[i])) * np.dot(H[k,:], H[k,:]) + \
            gamma * np.log(np.reciprocal(outl3[i]))
```

对应分母

```
G[i,k] = Gik_numer / Gik_denom
```

更新H

$$\mathcal{L} = \alpha\mathcal{L}_{str} + \beta\mathcal{L}_{attr} + \gamma\mathcal{L}_{dis}$$

```
Hkj_numer = alpha * np.dot( np.multiply(np.log(np.reciprocal(outl1)), G[:,k]), \
                    ( A[:,j] - (np.matmul(G,H[:,j]) - np.multiply(G[:,k],H[k,j]) ) ) )
Hkj_denom = alpha * ( np.dot(np.log(np.reciprocal(outl1)), np.multiply(G[:,k], G[:,k])) )
```

$$H_{kj} = \frac{\sum_{i=1}^{N} \log\left(\frac{1}{O_{1i}}\right)(A_{ij} - \sum_{k' \neq k} G_{ik'} H_{k'j}) G_{ik}}{\sum_{i=1}^{N} \log\left(\frac{1}{O_{1i}}\right) G_{ik}^2} \tag{7}$$

更新U $\mathcal{L} = \alpha\mathcal{L}_{str} + \beta\mathcal{L}_{attr} + \gamma\mathcal{L}_{dis}$

$$\mathcal{L} = \mathcal{L}_{str} + \alpha\mathcal{L}_{attr} + \beta\mathcal{L}_{dis} \quad (5)$$

这个地方公式中$\alpha\beta\gamma$，是没有用公式5中的

```
Uik_numer_1 = beta * np.log(np.reciprocal(outl2[i])) * ( np.dot( V[k,:], \
                                (C[i] - (np.matmul(U[i,:], V) - np.multiply(U[i,k], V[k,:])) ) ))
```

$$U_{ik}^{num1} = \beta \log\left(\frac{1}{O_{2i}}\right) \sum_{d=1}^{D} (C_{id} - \sum_{k' \neq k} U_{ik'} V_{k'd}) V_{kd}$$

```
Uik_numer_2 = gamma * np.log(np.reciprocal(outl3[i])) * np.dot( \
                            (G[i,:] - (np.matmul(U[i,:], W) - np.multiply(U[i,k], W[:,k]))), W[:,k] )
```

$$U_{ik}^{num2} = \gamma \log\left(\frac{1}{O_{3i}}\right)\left(G_{i\cdot} - (U_{i\cdot}W) - (U_{ik} \cdot W_{\cdot k})\right) W_{\cdot k}$$

```
Uik_denom = beta * np.log(np.reciprocal(outl2[i])) * np.dot(V[k,:], V[k,:] \
                            ) + gamma * np.log(np.reciprocal(outl3[i])) * np.dot(W[:,k], W[:,k])
```

$$U_{ik} = \frac{U_{ik}^{num1} + U_{ik}^{num2}}{\beta \log\left(\frac{1}{O_{2i}}\right)(V_{k\cdot} \cdot V_{k\cdot}) + \gamma \log\left(\frac{1}{O_{3i}}\right) W_{\cdot k} \cdot W_{\cdot k}}$$

```
U[i,k] = (Uik_numer_1 + Uik_numer_2) / Uik_denom
```

```
Vkd_numer = beta * np.dot( np.multiply(np.log(np.reciprocal(outl2)), U[:,k]), ( C[:,d] \
                                - (np.matmul(U,V[:,d]) - np.multiply(U[:,k],V[k,d]) ) ) )
Vkd_denom = beta * ( np.dot(np.log(np.reciprocal(outl2)), np.multiply(U[:,k], U[:,k])) )
```

分子分母都有beta

```
V[k][d] = Vkd_numer / Vkd_denom
```

$$V_{kd} = \frac{\sum_{i=1}^{N} \log\left(\frac{1}{O_{2i}}\right)(C_{id} - \sum_{k' \neq k} U_{ik'} V_{k'd}) U_{ik}}{\sum_{i=1}^{N} \log\left(\frac{1}{O_{2i}}\right) U_{ik}^2} \tag{9}$$

Update $\bar{W}$ by Eq. 11.

```
logoi = np.log(np.reciprocal(outl3))
sqrt_logoi = np.sqrt(logoi)
sqrt_logoi = np.tile(sqrt_logoi, (K,1))
assert(sqrt_logoi.shape == G.T.shape)
```

(12, 3477)

$$\sqrt{\log\left(\frac{1}{O_{3i}}\right)}$$

```
term1 = np.multiply(sqrt_logoi, G.T)
term2 = np.multiply(sqrt_logoi, U.T)

svd_matrix = np.matmul(term1, term2.T)

svd_u, svd_sigma, svd_vt = np.linalg.svd(svd_matrix)

W = np.matmul(svd_u, svd_vt)
```

G.T

U.T

都做了转置操作

X Σ Y

X Y

Here the new matrices are defined as, $(\bar{G})_{i,k} = \sqrt{\log\left(\frac{1}{O_{3i}}\right)} G_{ik}$ and $\bar{U}_{ik} = \sqrt{\log\left(\frac{1}{O_{3i}}\right)} U_{ik}$. Say, $X\Sigma Y^T = \text{SVD}(\bar{G}^T\bar{U})$, then W can be obtained as:

$$\boxed{W = XY^T} \tag{11}$$

Update outlier scores by Eq. from 12 to 14.

```
GH = np.matmul(G, H)
UV = np.matmul(U,V)
WUTrans = np.matmul(W, U.T)

outl1_numer = alpha * (np.multiply((A - GH),(A - GH))).sum(axis=1)

outl1_denom =  alpha * pow(np.linalg.norm((A - GH),'fro'),2)

outl1_numer = outl1_numer * mu
outl1 = outl1_numer / outl1_denom
```

$\mu = 1$

np.linalg.norm:求范数；fro:矩阵元素绝对值的平方和再开平方

$$\sum_{i=1}^{N} O_{1i} = \mu,$$

$$O_{1i} = \frac{\left(\sum_{j=1}^{N}(A_{ij} - G_{i\cdot} \cdot H_{\cdot j})^2\right) \cdot \mu}{\sum_{i=1}^{N}\sum_{j=1}^{N}(A_{ij} - G_{i\cdot} \cdot H_{\cdot j})^2} \tag{12}$$

```
outl2_numer = beta * (np.multiply((C - UV),(C - UV))).sum(axis=1)

outl2_denom =  beta * pow(np.linalg.norm((C - UV),'fro'),2)

outl2_numer = outl2_numer * mu
outl2 = outl2_numer / outl2_denom
```

$$O_{2i} = \frac{\left(\sum_{d=1}^{D}(C_{id} - U_{i\cdot} \cdot V_{\cdot d})^2\right) \cdot \mu}{\sum_{i=1}^{N}\sum_{j=1}^{D}(C_{id} - U_{i\cdot} \cdot V_{\cdot d})^2} \tag{13}$$

```
outl3_numer = gamma * (np.multiply((G.T - WUTrans),(G.T - WUTrans))).sum(axis=0).T

outl3_denom = gamma * pow(np.linalg.norm((G.T - WUTrans),'fro'),2)

outl3_numer = outl3_numer * mu
outl3 = outl3_numer / outl3_denom
```

$$O_{3i} = \frac{\left(\sum_{k=1}^{K}(G_{ik} - W_{i\cdot} \cdot U_{\cdot k})^2\right) \cdot \mu}{\sum_{i=1}^{N}\sum_{k=1}^{K}(G_{ik} - W_{i\cdot} \cdot U_{\cdot k})^2} \tag{14}$$

8: Embedding for the node v_i is $\frac{G_{i\cdot}+U_{i\cdot}(W^T)}{2}, \forall v_i \in V$.

```
In[74]: G.shape
Out[74]: (3477, 12)
In[75]: U.shape
Out[75]: (3477, 12)
In[76]: W.shape
Out[76]: (12, 12)
In[77]: (G+np.matmul(U,W)).shape
Out[77]: (3477, 12)
```

9: Final outlier score for the node v_i is a weighted average of O_{1i}, O_{2i} and O_{3i}, $\forall v_i \in V$.

每个节点对应到的异常性

```
In[79]: outl1.shape
Out[79]: (3477,)
```

DEEP AUTOENCODING GAUSSIAN MIXTURE MODEL FOR UNSUPERVISED ANOMALY DETECTION

利用AutoEncoder和高斯混合模型的无监督异常检测

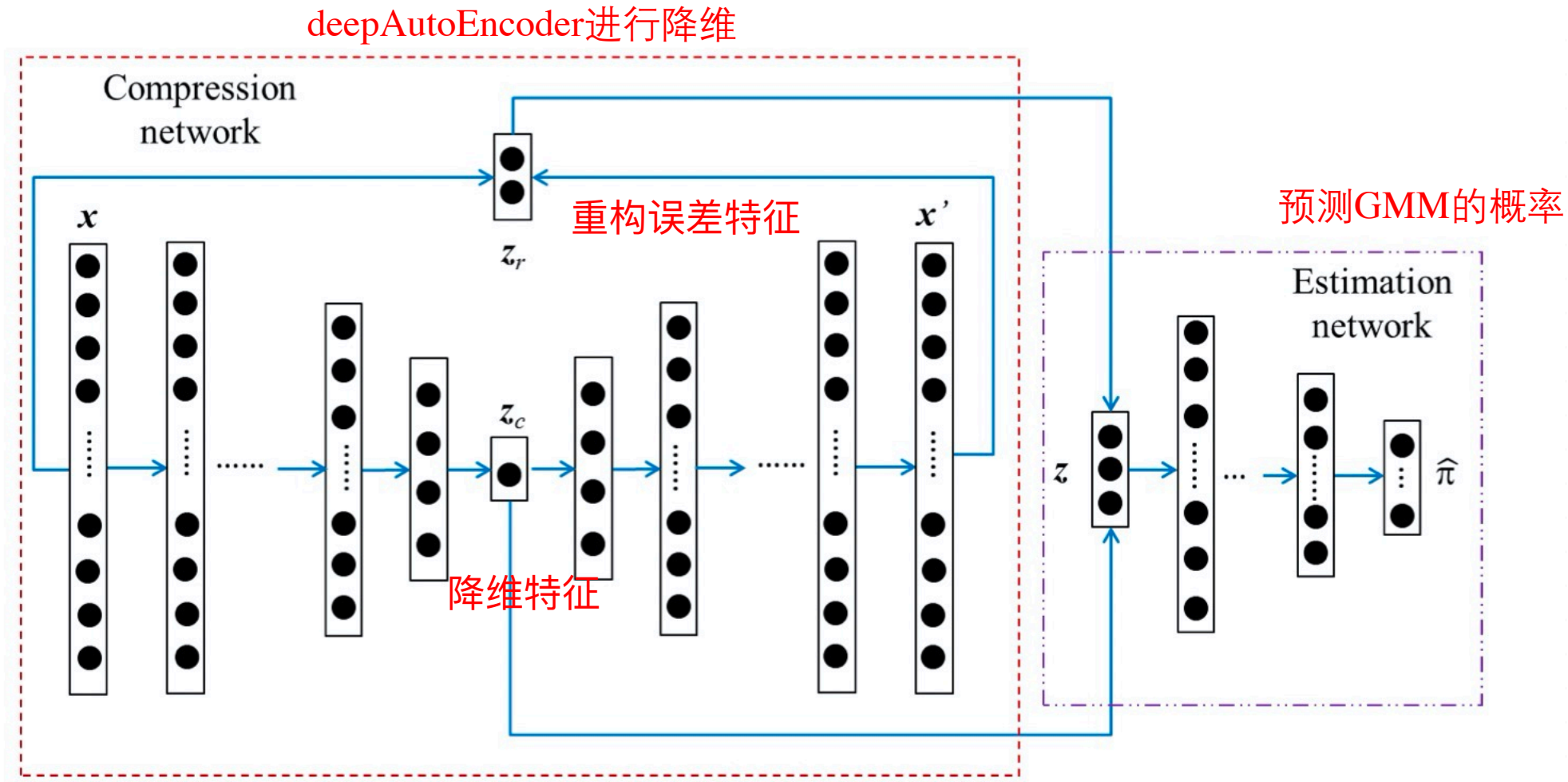
一种常用的异常检测方法是：
首先我们使用AutoEncoder的方法进行降维，然后使用降维后的特征进行聚类，或者密度评估方法，发现异常。

但是这种方法，两步走的方式一般是脱离的（解耦）。因为两个部分是分开训练的，降维时，完全没有考虑聚类的效果。所以在降维过程中，可能会丢失聚类分析的关键信息。

因此作者提出了一种新的异常检测方法，
DAGMM：采用AutoEncoder和GMM以一种端到端的方式进行训练，以解决分步训练带来的缺点。

时间序列降维　　聚类方法（密度评估方法）

3 DEEP AUTOENCODING GAUSSIAN MIXTURE MODEL

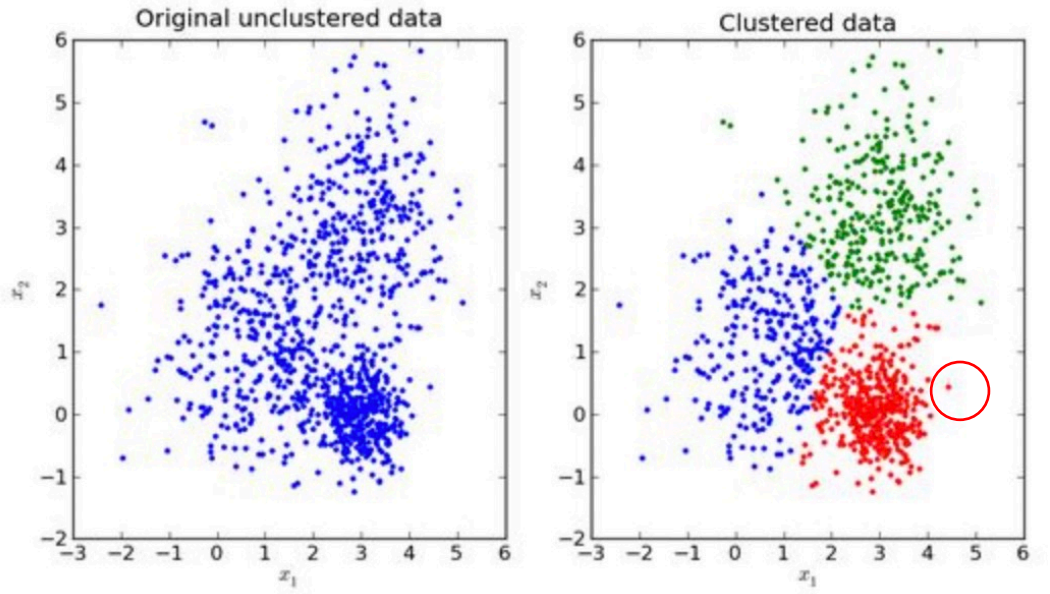


compression
estimation network

使用神经网络去学习到每个节点属于哪个高斯分布的概率
（参考EM算法学习到类别的概率）

EM: https://www.bilibili.com/video/BV16W411f7Y7?p=19

假设这些样本是由多个高斯分布组成的高斯混合模型(GMM)



我们目标是估计节点属于哪个高斯分布的概率

γ_{jk}: 节点j属于高斯分布k的概率

GMM最常用的一种参数评估方法就是EM算法

- 首先初始化参数　每个高斯分布的均值，方差和每个分布占比
- E-step：依据当前参数，计算每个数据 j 来自子模型 k 的可能性

$$\gamma_{jk} = \frac{\alpha_k \phi(x_j|\theta_k)}{\sum_{k=1}^{K} \alpha_k \phi(x_j|\theta_k)}, j = 1, 2, \ldots, N; k = 1, 2, \ldots, K$$

- M-step：计算新一轮迭代的模型参数

$$\mu_k = \frac{\sum_j^N (\gamma_{jk} x_j)}{\sum_j^N \gamma_{jk}}, k = 1, 2, \ldots, K$$

计算每个分布的mu，sigma

$$\Sigma_k = \frac{\sum_j^N \gamma_{jk}(x_j - \mu_k)(x_j - \mu_k)^T}{\sum_j^N \gamma_{jk}}, k = 1, 2, \ldots, K$$ （用这一轮更新后的 μ_k ）

$$\alpha_k = \frac{\sum_{j=1}^N \gamma_{jk}}{N}, k = 1, 2, \ldots, K$$

计算每个分布占比

3.2 COMPRESSION NETWORK 目的是将高维度特征，压缩到底维度

压缩网络提供的低维表示包含两个特征源：

1. deep autoencoder学习到的低维特征
2. 重构误差特征

$$\mathbf{z}_c = h(\mathbf{x}; \theta_e), \tag{1}$$

deep autoencoder学习到的特征

deep autoencoder还原的x

$$\mathbf{x}' = g(\mathbf{z}_c; \theta_d), \tag{1}$$

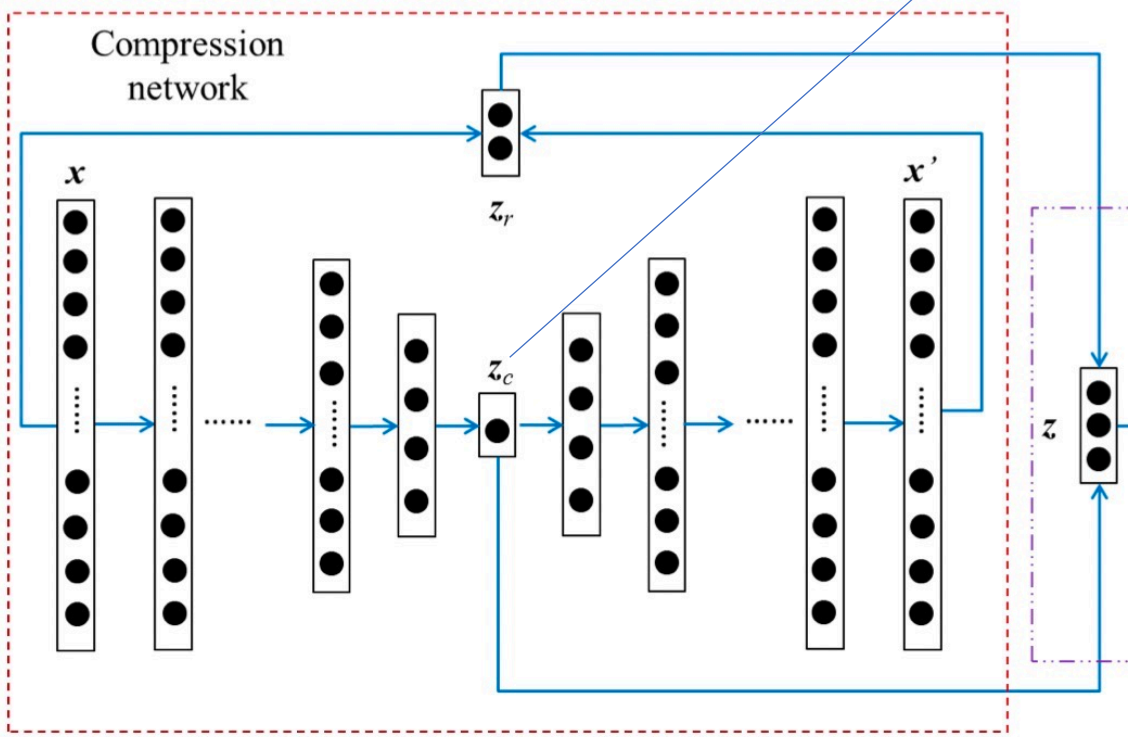
重构误差特征

$$\mathbf{z}_r = f(\mathbf{x}, \mathbf{x}'), \tag{2}$$

$$\mathbf{z} = [\mathbf{z}_c, \mathbf{z}_r], \tag{3}$$

压缩最终特征

计算重构误差的函数

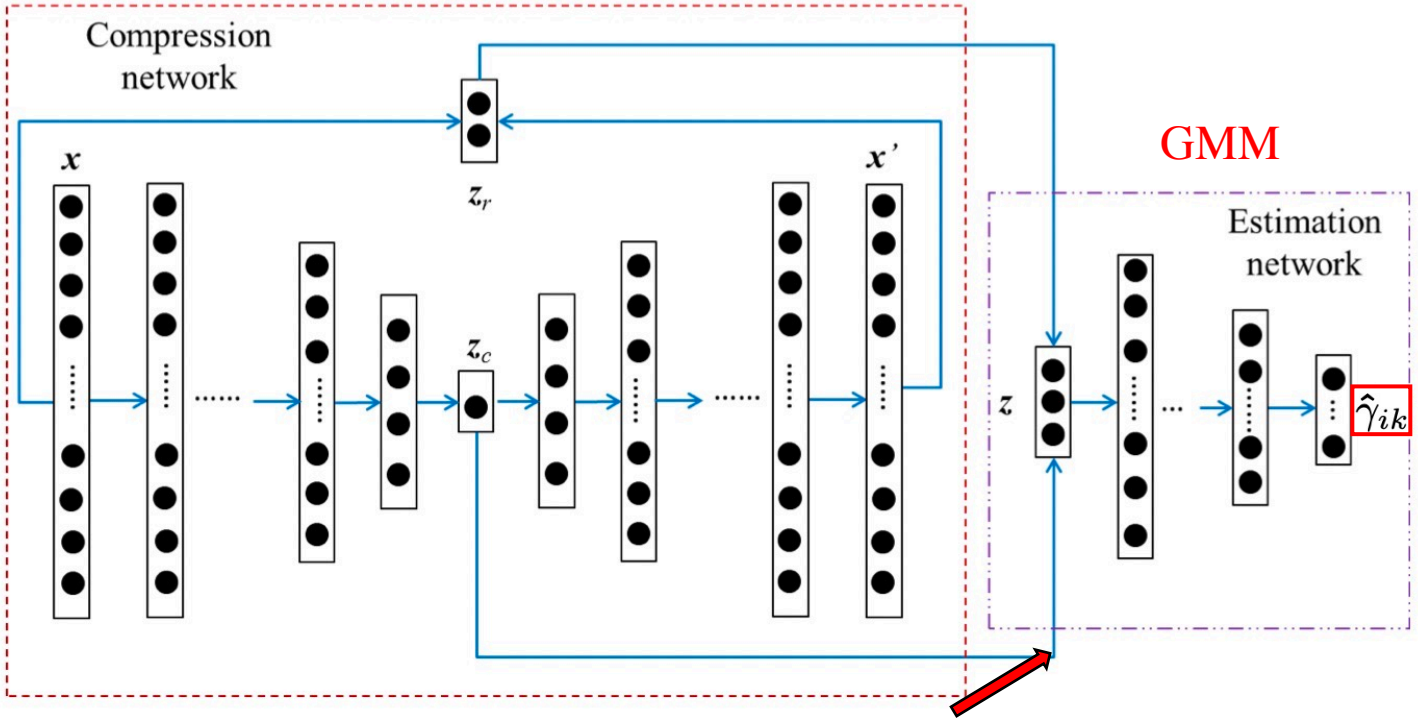


重构误差特征的选择

absolute Euclidean distance,
relative Euclidean distance and cosine similarity

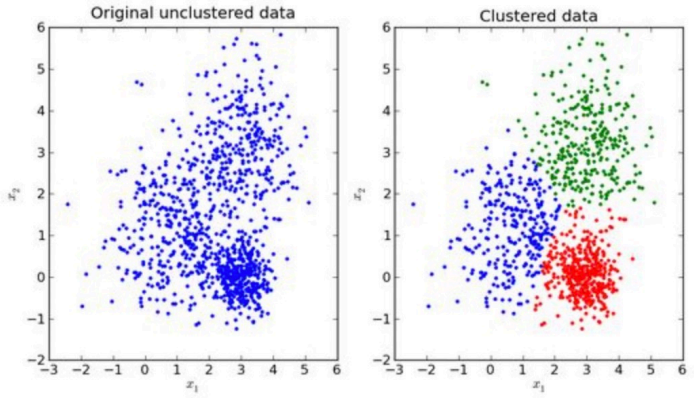
$$\mathbf{z}_r = \frac{\|\mathbf{x}-\mathbf{x}'\|_2}{\|\mathbf{x}\|_2} \quad \frac{\mathbf{x}\cdot\mathbf{x}'}{\|\mathbf{x}\|_2\|\mathbf{x}'\|_2}$$

Estimation Network



我们假设Compression Network输出的特征点，符合高斯混合分布，传统做法是使用EM算法。但是使用EM算法，很难去使用端到端的训练方式，于是，作者采用了全连接来学习GMM。

Estimation network



γ_{ik}: 节点i属于高斯分布k的概率

- 首先初始化参数
- E-step：依据当前参数，计算每个数据 j 来自子模型 k 的可能性

$$\gamma_{jk} = \frac{\alpha_k \phi(x_j|\theta_k)}{\sum_{k=1}^{K} \alpha_k \phi(x_j|\theta_k)}, j = 1, 2, \ldots, N; k = 1, 2, \ldots, K$$

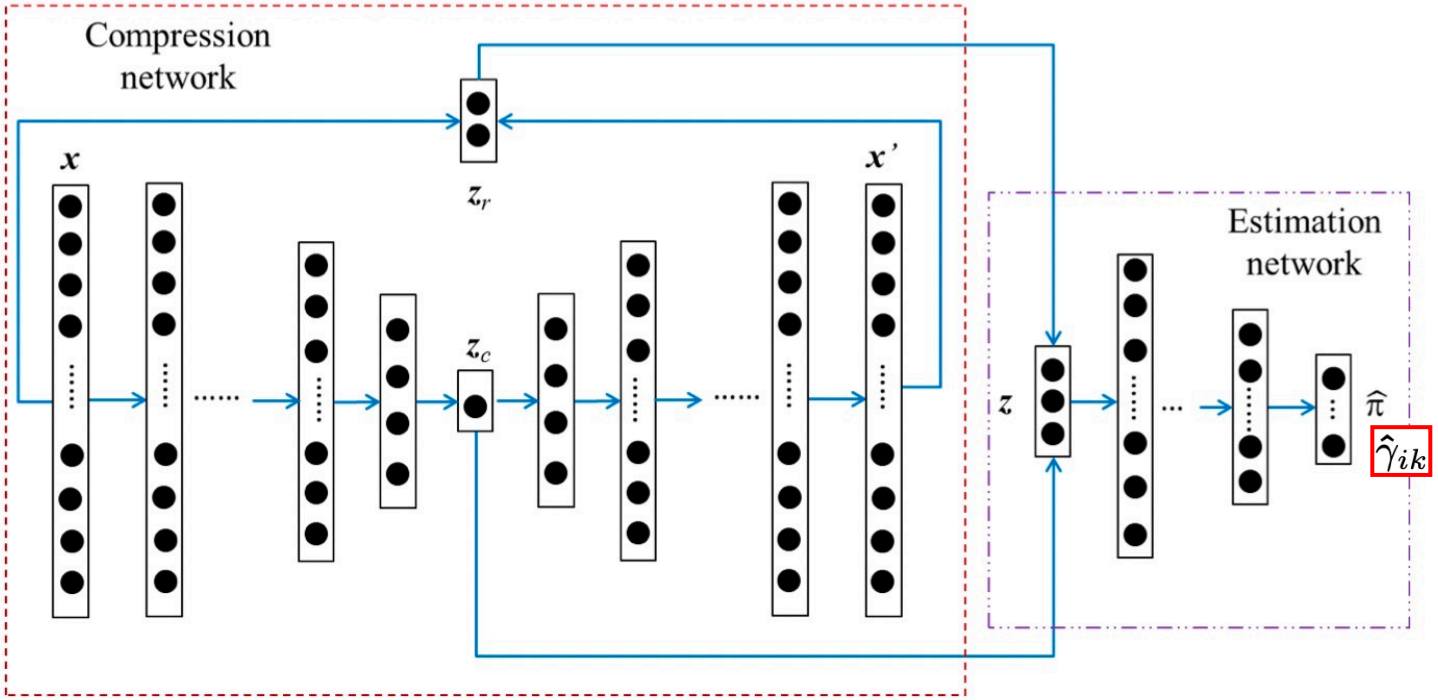
- M-step：计算新一轮迭代的模型参数

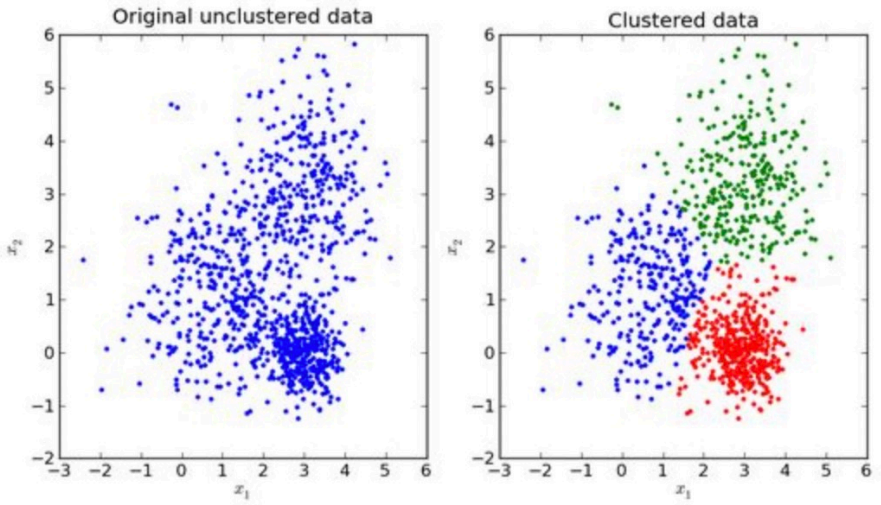
$$\mu_k = \frac{\sum_j^N (\gamma_{jk} x_j)}{\sum_j^N \gamma_{jk}}, k = 1, 2, \ldots, K$$

$$\Sigma_k = \frac{\sum_j^N \gamma_{jk}(x_j - \mu_k)(x_j - \mu_k)^T}{\sum_j^N \gamma_{jk}}, k = 1, 2, \ldots, K \quad \text{（用这一轮更新后的 } \mu_k \text{）}$$

现在，我们采用一个MLP学习到隐层向量，直接学习到z属于K个高斯分布的概率。即 $\hat{\gamma}_{ik}$，而EM算法是通过迭代学习到的 $\hat{\gamma}_{ik}$。

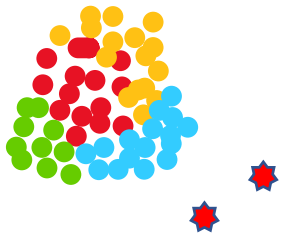
我们使用MLP去模拟了EM算法的学习结果。
同样的，知道了 $\hat{\gamma}_{ik}$，我就可以计算mu和sigma了。





但是高斯混合分布，其中一个超参数K(多少个高斯混合分布)是需要我们定义的。
但是我们事先是不知道隐变量z是符合多少个高斯分布的。（我们在代码中定义K=2）

虽然超参数K对高斯混合分布很重要，但是在这个实例中，K却显得没那么重要。
我们先假设正常的数据其实都应该聚集在一起，不论我K选择多少，正常点都应该是聚集的，而异常点就要偏离中心。

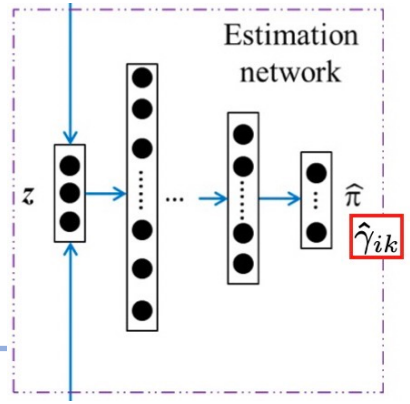


还有一点是，隐层向量z是端到端训练的，所以他会将数据训练成我们需要的K个高斯分布。

EM: https://www.bilibili.com/video/BV16W411f7Y7?p=19

3.3 Estimation Network

给定低维的输入，计算在GMM框架中的密度



$$\mathbf{p} = MLN(\mathbf{z}; \theta_m), \qquad \hat{\gamma} = \text{softmax}(\mathbf{p}), \tag{4}$$

多层神经网络

属于每个高斯分布的概率

给定N个样本，对GMM参数进行评估。 $\hat{\gamma}_i$ 是输入特征$\mathbf{z}_i$的预测

$$\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N}, \qquad \hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik}\mathbf{z}_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}, \qquad \hat{\mathbf{\Sigma}}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik}(\mathbf{z}_i - \hat{\mu}_k)(\mathbf{z}_i - \hat{\mu}_k)^T}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}. \tag{5}$$

mixture probability

每个高斯分布的占比

mean

每个高斯分布的mu

covariance

每个高斯分布的方差

根据估计的参数，可以进一步推断出样品能量值(异常值)

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

高斯分布的概率密度函数

该点的似然函数

$$E(\mathbf{z}) = -\log\left(\sum_{k=1}^{K} \hat{\phi}_k \frac{\exp\left(-\frac{1}{2}(\mathbf{z}-\hat{\mu}_k)^T\hat{\mathbf{\Sigma}}_k^{-1}(\mathbf{z}-\hat{\mu}_k)\right)}{\sqrt{|2\pi\hat{\mathbf{\Sigma}}_k|}}\right). \tag{6}$$

取相反数，越小越好，相当于loss函数

在使用学习的 GMM 参数的测试阶段，可以直接估计样本能量，并通过预先选择的阈值将高能量样本预测为异常

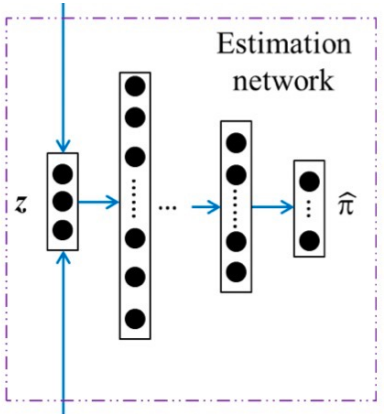
3.3 Estimation Network

给定低维的输入，计算在GMM框架中的密度

属于每个GMM的概率

$$\mathbf{p} = MLN(\mathbf{z}; \theta_m), \qquad \hat{\gamma} = \text{softmax}(\mathbf{p}), \qquad (4)$$

多层神经网络



batch size

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} \mathbf{z}_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$$

每个高斯分布的mu

$$\hat{\mathbf{\Sigma}}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} (\mathbf{z}_i - \hat{\mu}_k)(\mathbf{z}_i - \hat{\mu}_k)^T}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$$

每个高斯分布的方差

$$\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N},$$

每个高斯分布的占比

- 首先初始化参数
- E-step：依据当前参数，计算每个数据 j 来自子模型 k 的可能性

$$\gamma_{jk} = \frac{\alpha_k \phi(x_j|\theta_k)}{\sum_{k=1}^{K} \alpha_k \phi(x_j|\theta_k)}, j = 1, 2, \ldots, N; k = 1, 2, \ldots, K$$

- M-step：计算新一轮迭代的模型参数

$$\mu_k = \frac{\sum_{j}^{N} (\gamma_{jk} x_j)}{\sum_{j}^{N} \gamma_{jk}}, k = 1, 2, \ldots, K$$

$$\Sigma_k = \frac{\sum_{j}^{N} \gamma_{jk} (x_j - \mu_k)(x_j - \mu_k)^T}{\sum_{j}^{N} \gamma_{jk}}, k = 1, 2, \ldots, K$$ （用这一轮更新后的 μ_k ）

$$\alpha_k = \frac{\sum_{j=1}^{N} \gamma_{jk}}{N}, k = 1, 2, \ldots, K$$

3.4 OBJECTIVE FUNCTION

$\lambda_1 = 0.1$ and $\lambda_2 = 0.005$

$$J(\theta_e, \theta_d, \theta_m) = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{x}_i, \mathbf{x}'_i) + \frac{\lambda_1}{N}\sum_{i=1}^{N} E(\mathbf{z}_i) + \lambda_2 P(\hat{\mathbf{\Sigma}}). \quad (7)$$

DAGMM的loss函数

惩罚项

模拟了我们能观察到输入样本的概率，
我们寻找压缩和估计网络的最佳组合，使观测输入样本的可能性最大化。

deep autoencoder重构误差

$$L(\mathbf{x}_i, \mathbf{x}'_i) = \|\mathbf{x}_i - \mathbf{x}'_i\|_2^2$$

$$P(\hat{\mathbf{\Sigma}}) = \sum_{k=1}^{K}\sum_{j=1}^{d}\frac{1}{\hat{\mathbf{\Sigma}}_{kjj}}$$

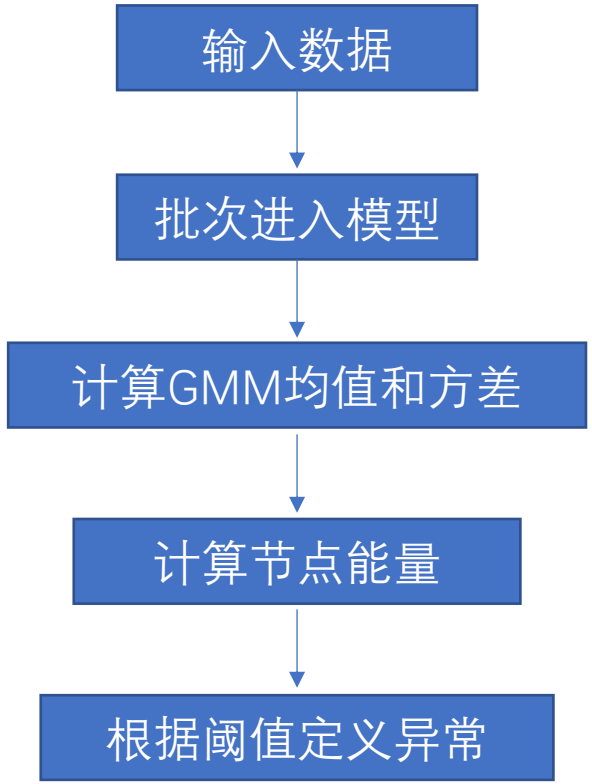
惩罚项，避免GMM模型中，协方差矩阵对角线为接近为0

同一个维度之间，方差不能太小

每个高斯分布，有三个维度，
所以形成3*3协方差矩阵

```
In[28]: cov_k
Out[28]:
tensor([[ 9.4451e-05, -4.5261e-05,  2.3089e-04],
        [-4.5261e-05,  3.9020e-02,  1.5424e-03],
        [ 2.3089e-04,  1.5424e-03,  4.6990e-03]], grad_fn=<SelectBackward>)
```

测试阶段



$$E(\mathbf{z}) = -\log\left(\sum_{k=1}^{K}\hat{\phi}_k \frac{\exp\left(-\frac{1}{2}(\mathbf{z}-\hat{\mu}_k)^T\hat{\mathbf{\Sigma}}_k^{-1}(\mathbf{z}-\hat{\mu}_k)\right)}{\sqrt{|2\pi\hat{\mathbf{\Sigma}}_k|}}\right). \tag{6}$$

能量越大，说明异常越大

DAGMM的几个变体

移除重构误差

从目标函数中移除能量函数
变化为AutoEncoder方式

类似于PAE，end2end方式训练

两步走策略
GMM使用EM算法

两步走策略
分别训练模型

先训练ae，在end2end训练

变分推断方法

- GMM-EN. In this variant, we remove the reconstruction error component from the objective function of DAGMM. In other words, the estimation network in DAGMM performs membership estimation without the constraints from the compression network. With the learned membership estimation, we infer sample energy by Equation (5) and (6) under the GMM framework. Sample energy is used as the criterion for anomaly detection.
- PAE. We obtain this variant by removing the energy function from the objective function of DAGMM, and this DAGMM variant is equivalent to a deep autoenoder. To ensure the compression network is well trained, we adopt the pre-training strategy (Vincent et al. (2010)). Sample reconstruction error is the criterion for anomaly detection.
- E2E-AE. This variant shares the same setting with PAE, but the deep autoencoder is learned by end-to-end training. Sample reconstruction error is the criterion for anomaly detection
- PAE-GMM-EM. This variant adopts a two-step approach. At step one, we learn the compression network by pre-training deep autoencoder. At step two, we use the output from the compression network to train the GMM by a traditional EM algorithm. The training procedures in the two steps are separated. Sample energy is used as the criterion for anomaly detection.
- PAE-GMM. This variant also adopts a two-step approach. At step one, we learn the compression network by pre-training deep autoencoder. At step two, we use the output from the compression network to train the estimation network. The training procedures in the two steps are separated. Sample energy is used as the criterion for anomaly detection.
- DAGMM-p. This variant is a compromise between DAGMM and PAE-GMM: we first train the compression network by pre-training, and then fine-tune DAGMM by end-to-end training. Sample energy is the criterion for anomaly detection.
- DAGMM-NVI. The only difference between this variant and DAGMM is that this variant adopts the framework of neural variational inference (Mnih & Gregor (2014)) and replaces Equation (6) with the upper bound in Equation (10) as a part of the objective function.

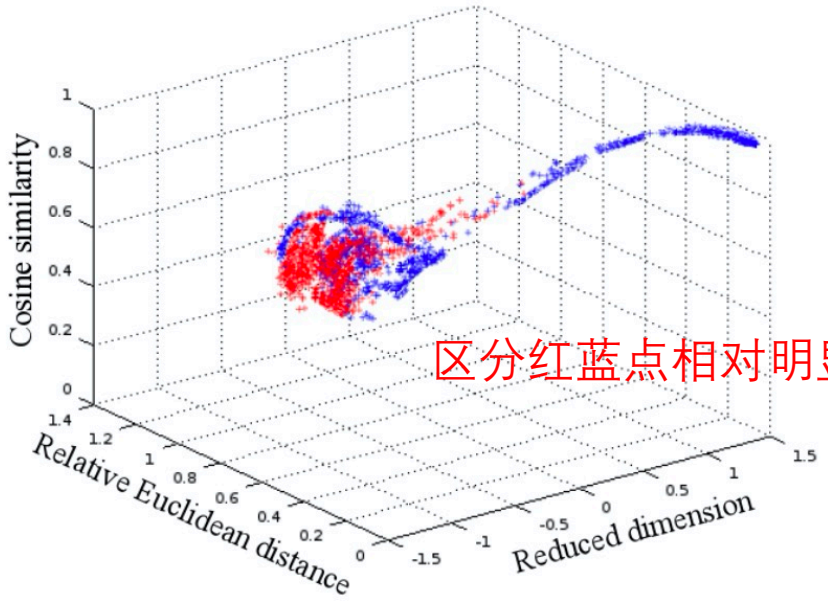
Method	KDDCUP			Thyroid		
	Precision	Recall	F_1	Precision	Recall	F_1
OC-SVM	0.7457	0.8523	0.7954	0.3639	0.4239	0.3887
DSEBM-r	0.1972	0.2001	0.1987	0.0404	0.0403	0.0403
DSEBM-e	0.7369	0.7477	0.7423	0.1319	0.1319	0.1319
DCN	0.7696	0.7829	0.7762	0.3319	0.3196	0.3251
GMM-EN	0.1932	0.1967	0.1949	0.0213	0.0227	0.0220
PAE	0.7276	0.7397	0.7336	0.1894	0.2062	0.1971
E2E-AE	0.0024	0.0025	0.0024	0.1064	0.1316	0.1176
PAE-GMM-EM	0.7183	0.7311	0.7246	0.4745	0.4538	0.4635
PAE-GMM	0.7251	0.7384	0.7317	0.4532	**0.4881**	0.4688
DAGMM-p	0.7579	0.7710	0.7644	0.4723	0.4725	0.4713
DAGMM-NVI	0.9290	**0.9447**	0.9368	0.4383	0.4587	0.4470
DAGMM	**0.9297**	0.9442	**0.9369**	**0.4766**	0.4834	**0.4782**

Method	Arrhythmia			KDDCUP-Rev		
	Precision	Recall	F_1	Precision	Recall	F_1
OC-SVM	**0.5397**	0.4082	0.4581	0.7148	**0.9940**	0.8316
DSEBM-r	0.1515	0.1513	0.1510	0.2036	0.2036	0.2036
DSEBM-e	0.4667	0.4565	0.4601	0.2212	0.2213	0.2213
DCN	0.3758	0.3907	0.3815	0.2875	0.2895	0.2885
GMM-EN	0.3000	0.2792	0.2886	0.1846	0.1746	0.1795
PAE	0.4393	0.4437	0.4403	0.7835	0.7817	0.7826
E2E-AE	0.4667	0.4538	0.4591	0.7434	0.7463	0.7448
PAE-GMM-EM	0.3970	0.4168	0.4056	0.2822	0.2847	0.2835
PAE-GMM	0.4575	0.4823	0.4684	0.6307	0.6278	0.6292
DAGMM-p	0.4909	0.4679	0.4787	0.2750	0.2810	0.2780
DAGMM-NVI	0.5091	0.4892	0.4981	0.9211	0.9211	0.9211
DAGMM	0.4909	**0.5078**	**0.4983**	**0.9370**	0.9390	**0.9380**

Ratio c	DAGMM			DCN		
	Precision	Recall	F_1	Precision	Recall	F_1
1%	0.9201	0.9337	0.9268	0.7585	0.7611	0.7598
2%	0.9186	0.9340	0.9262	0.7380	0.7424	0.7402
3%	0.9132	0.9272	0.9201	0.7163	0.7293	0.7228
4%	0.8837	0.8989	0.8912	0.6971	0.7106	0.7037
5%	0.8504	0.8643	0.8573	0.6763	0.6893	0.6827
Ratio c	DSEBM-e			OC-SVM		
	Precision	Recall	F_1	Precision	Recall	F_1
1%	0.6995	0.7135	0.7065	0.7129	0.6785	0.6953
2%	0.6780	0.6876	0.6827	0.6668	0.5207	0.5847
3%	0.6213	0.6367	0.6289	0.6393	0.4470	0.5261
4%	0.5704	0.5813	0.5758	0.5991	0.3719	0.4589
5%	0.5345	0.5375	0.5360	0.1155	0.3369	0.1720

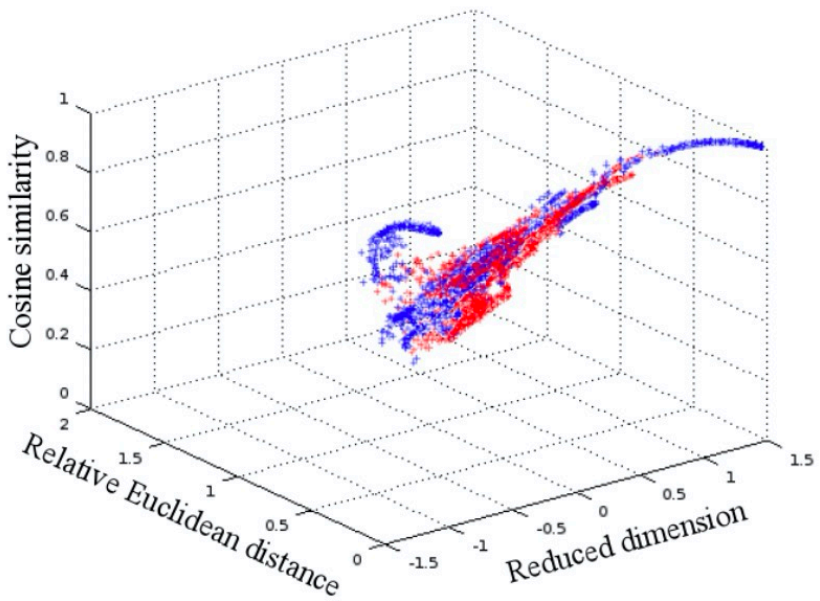
增加一定量的异常数据到训练集后的测试集评估效果

红色异常
蓝色正常

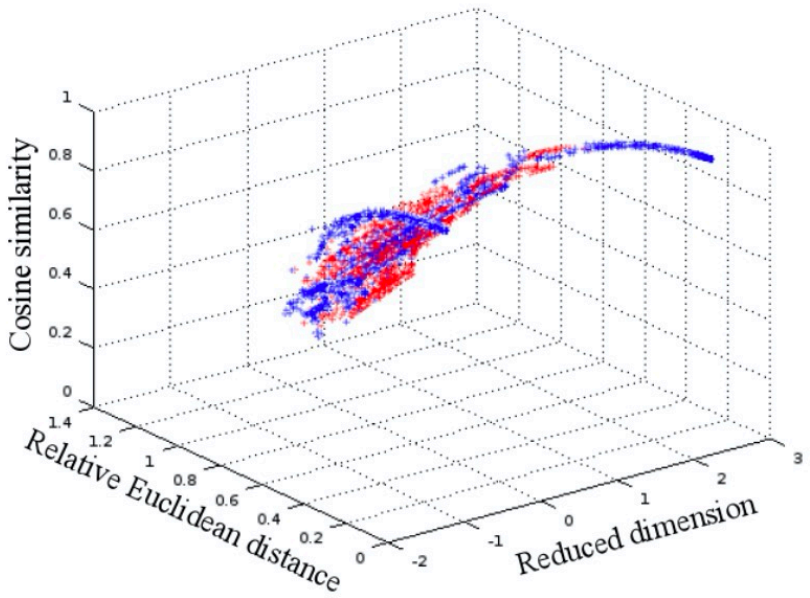


区分红蓝点相对明显

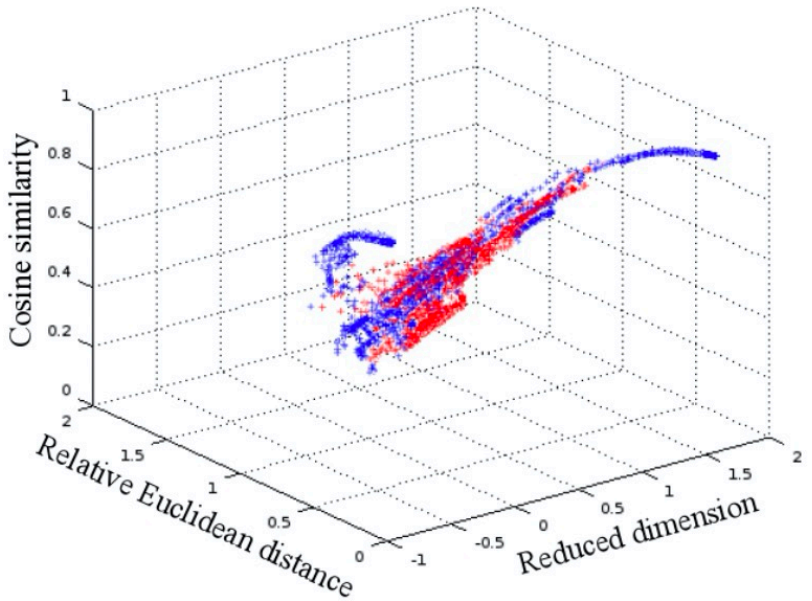
(a) KDDCUP@DAGMM



(b) KDDCUP@PAE



(c) KDDCUP@DAGMM-p



(d) KDDCUP@DCN

Code

这是与KDD-99第五届知识发现和数据挖掘国际会议联合举办的第三届国际知识发现和数据挖掘工具竞赛所使用的数据集。竞赛任务是建立一个网络入侵检测器，一个能够区分坏的“连接”(被称为入侵或攻击)和好的“正常连接”的预测模型。该数据库包含一组要审计的标准数据，其中包括在军事网络环境中模拟的各种各样的入侵。

```
tensor([0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        1.0000e+00, 0.0000e+00, 0.0000e+00, 3.3892e-07, 2.5934e-04, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 1.5656e-02, 1.5656e-02, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00,
        1.1024e-01, 1.1024e-01, 1.0000e+00, 0.0000e+00, 3.0000e-02, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00])
```

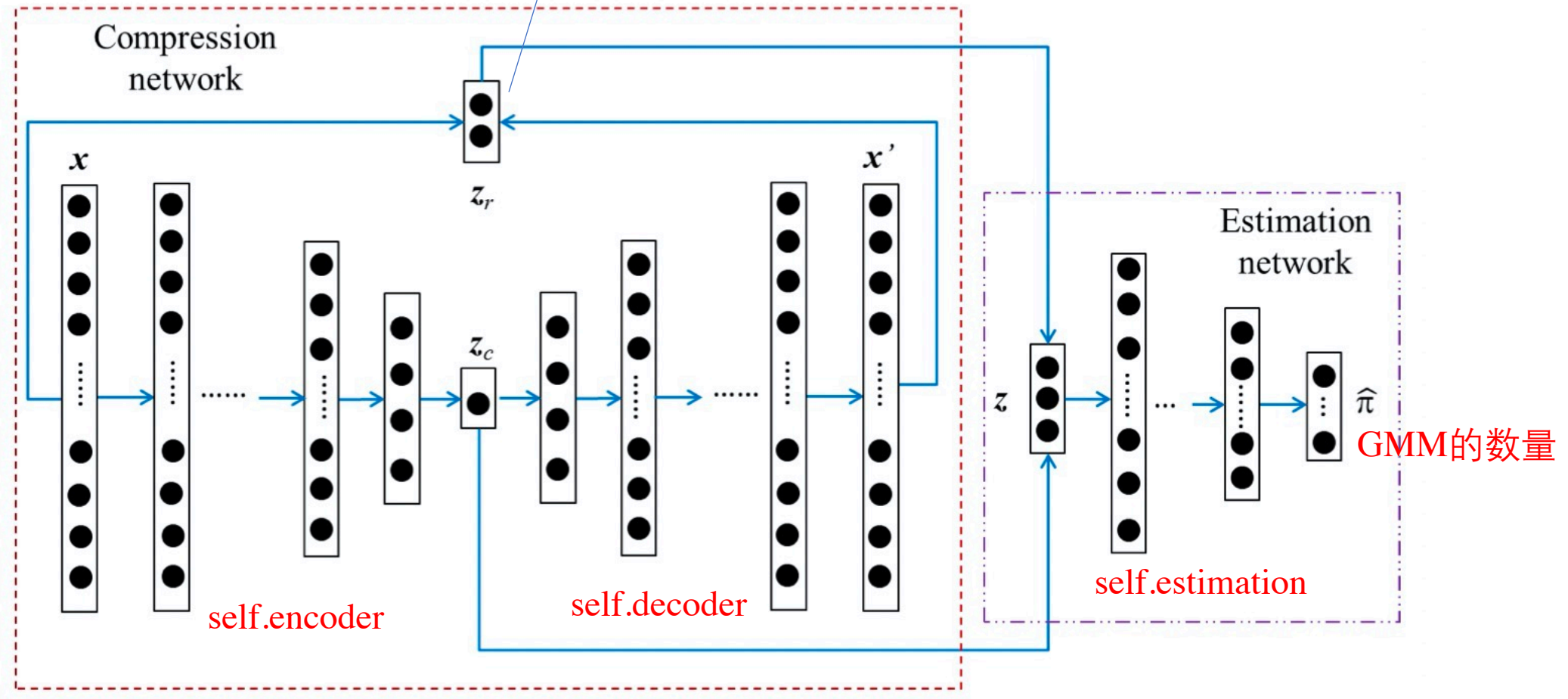
- originally contains samples of 41 dimensions, where 34 of them are continuous and 7 are categorical.

定义DAGMM

cosine similarity

```
rec_cosine = F.cosine_similarity(x, dec, dim=1)  # cos误差
rec_euclidean = F.pairwise_distance(x, dec,p=2)  # 欧式距离误差
```

relative Euclidean distance



```
torch.dist(x[i], x_hat[i])  # 均方误差
```

$$J(\theta_e, \theta_d, \theta_m) = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{x}_i, \mathbf{x}'_i) + \frac{\lambda_1}{N} \sum_{i=1}^{N} E(\mathbf{z}_i) + \lambda_2 P(\hat{\mathbf{\Sigma}}). \tag{7}$$

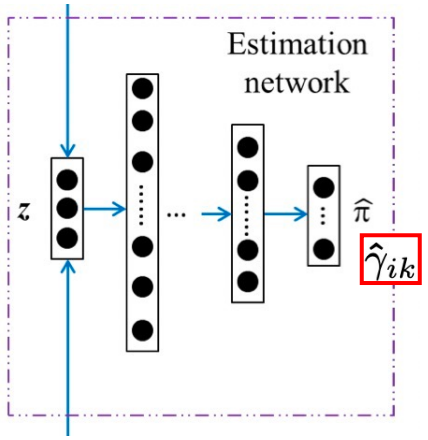
样本数

更新高斯混合模型参数

$$\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N}, \qquad \hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} \mathbf{z}_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}, \qquad \hat{\mathbf{\Sigma}}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} (\mathbf{z}_i - \hat{\mu}_k)(\mathbf{z}_i - \hat{\mu}_k)^T}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}. \quad (5)$$

```
#2 更新GMM参数; phi, mu, sigma
ceta, mean, cov = self.get_gmm_param(gamma, z)  # 更新gmm参数
```

$\hat{\gamma}_{ik}$ $\mathbf{z}_i$




```
ceta = torch.sum(gamma, dim=0) / N
```

```
mean = torch.sum(gamma.unsqueeze(-1) * z.unsqueeze(1), dim=0)
mean = mean / torch.sum(gamma, dim=0).unsqueeze(-1)  #shape: [n_gmm, z_dim]
```

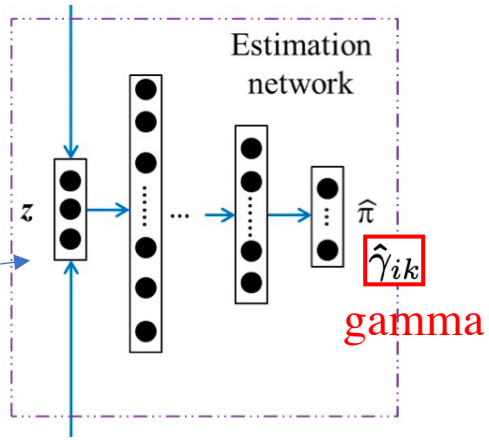
```
z_mean = (z.unsqueeze(1)- mean.unsqueeze(0))
cov = torch.sum(gamma.unsqueeze(-1).unsqueeze(-1) * z_mean.unsqueeze(-1) * z_mean.unsqueeze(-2), dim = 0) / torch.sum(gamma, dim=0)
```

GMM的数量

cov

$$E(\mathbf{z}) = -\log\left(\sum_{k=1}^{K}\hat{\phi}_k \frac{\exp\left(-\frac{1}{2}(\mathbf{z}-\hat{\mu}_k)^T\hat{\mathbf{\Sigma}}_k^{-1}(\mathbf{z}-\hat{\mu}_k)\right)}{\sqrt{|2\pi\hat{\mathbf{\Sigma}}_k|}}\right). \quad (6)$$

ceta



gamma

```
#3 E(x) energy损失
e = torch.tensor(0.0)  e: tensor(0.)
for i in range(z.shape[0]):  i: 0
    zi = z[i].unsqueeze(1)  zi: tensor([[ 0.0810],\n
    ei = self.sample_energy(ceta, mean, cov, zi,n_gmm,bs)
    e += ei
```

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N}\hat{\gamma}_{ik}\mathbf{z}_i}{\sum_{i=1}^{N}\hat{\gamma}_{ik}}$$

n_gmm: GMM的数量(2)

```
for k in range(n_gmm):
    miu_k = mean[k].unsqueeze(1)
    d_k = zi - miu_k

    inv_cov = torch.inverse(cov[k] + cov_eps)
    e_k = torch.exp(-0.5 * torch.chain_matmul(torch.t(d_k), inv_cov, d_k))
    e_k = e_k / torch.sqrt(torch.abs(torch.det(2 * math.pi * cov[k])))
    e_k = e_k * ceta[k]  # phi
    e += e_k.squeeze()
return -torch.log(e)
```

求行列式

```
p = torch.tensor(0.0)  p: tensor(0.)
for k in range(n_gmm):
    cov_k = cov[k]
    p_k = torch.sum(1 / torch.diagonal(cov_k, 0))  # 取对角线值
    p += p_k
```

$$P(\hat{\mathbf{\Sigma}}) = \sum_{k=1}^{K} \sum_{j=1}^{d} \frac{1}{\hat{\mathbf{\Sigma}}_{kjj}}$$

test

```
enc,dec,z,gamma = model(x)
m_prob, m_mean, m_cov = model.get_gmm_param(gamma, z)


sample_energy = model.sample_energy(m_prob, m_mean, m_cov, zi,gamma.shape[1],gamma.shape[0])
```

计算经过模型后的值

GMM的数量

cov

ceta

$$E(\mathbf{z}) = -\log\left(\sum_{k=1}^{K}\hat{\phi}_k \frac{\exp\left(-\frac{1}{2}(\mathbf{z}-\hat{\mu}_k)^T\hat{\mathbf{\Sigma}}_k^{-1}(\mathbf{z}-\hat{\mu}_k)\right)}{\sqrt{|2\pi\hat{\mathbf{\Sigma}}_k|}}\right). \tag{6}$$

比较常见的时序异常检测方法：

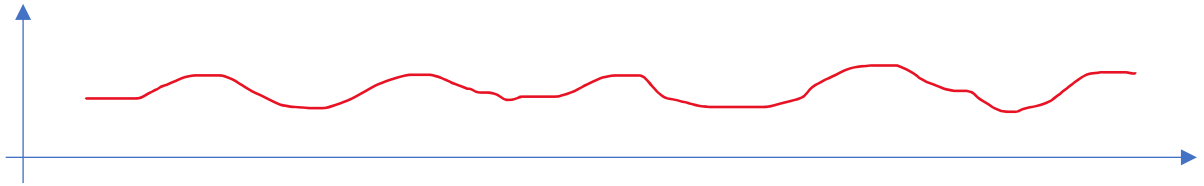
1. 基于统计（某时刻值偏离某段时间的均值和方差值超过定义的阈值）
2. Autoregressive Moving Average (ARMA)，异常未来时间的值，如果偏离正常值一定阈值，认定异常
3. One-Class SVM（根据正常点判断新的时间是否为异常）
4. KNN（近邻的样本数量，或近邻的样本距离计算其异常）
5. AutoEncoder或者PCA降维（根据重构误差来计算是否异常，假设异常在重构时会丢失信息）

A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data

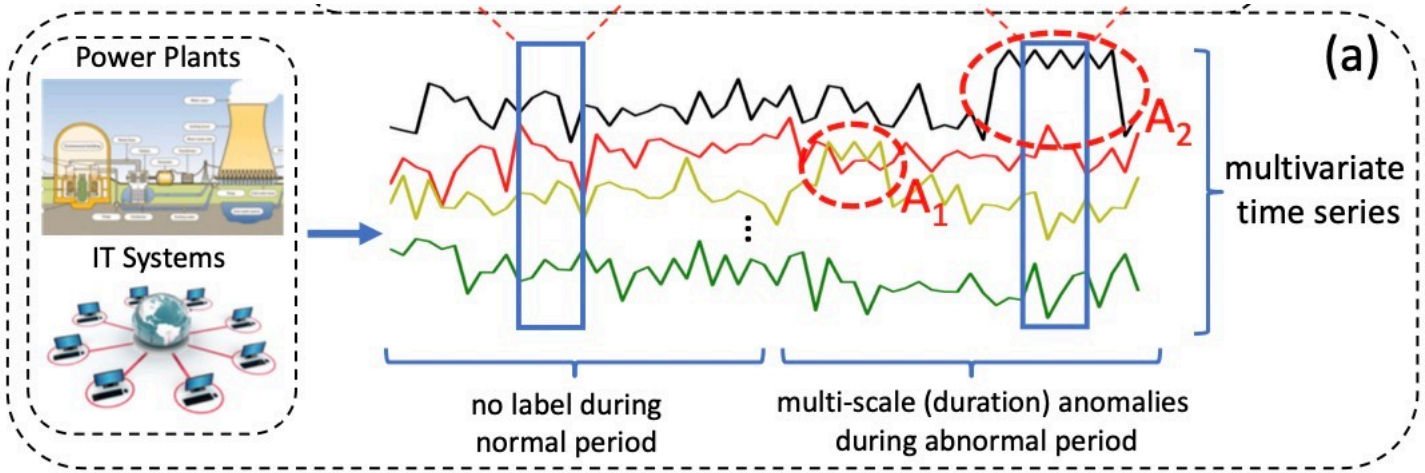
诊断在多变量中的时间序列

问题场景：发电厂

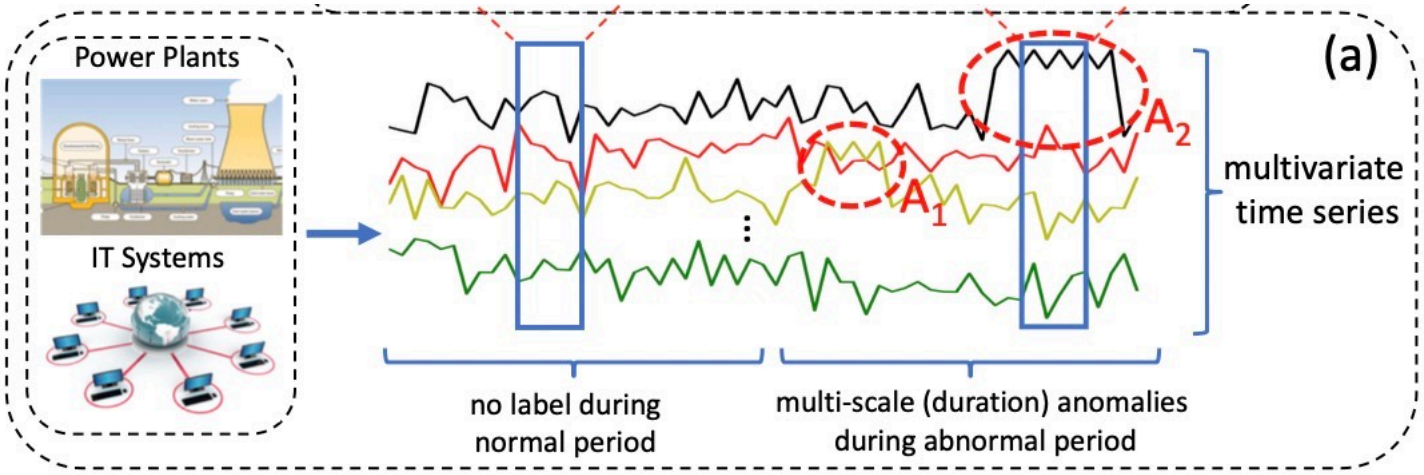
那么发电厂的每分钟的发电量，就是一条随时间变化的时间序列曲线（单变量）



多变量数据：多个传感器，测量发电厂信息（每个传感器就会形成一个时间序列，多个传感器就形成了多变量的时间序列）



数据格式



输入数据：
多个sensor测量发电厂多个信号的时间序列

t_0 t_1 t_{14}

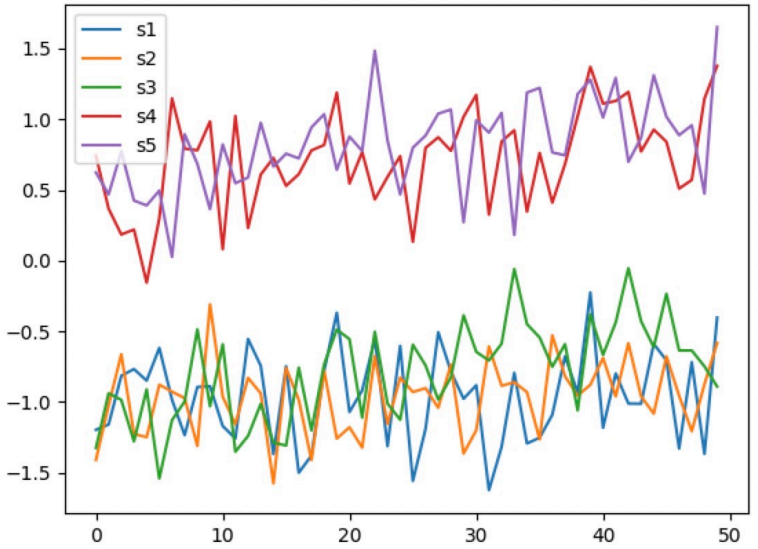
Sensor1: [-1.2, -1.16, -0.81, -0.77, -0.85, -0.62, -0.98, -1.23, -0.89, -0.89, -1.17, -1.26, -0.55, -0.74, -1.37]

Sensor2: [-1.41, -1.02, -0.66, -1.23, -1.25, -0.88, -0.93, -0.97, -1.31, -0.31, -0.96, -1.16, -0.83, -0.94, -1.58]

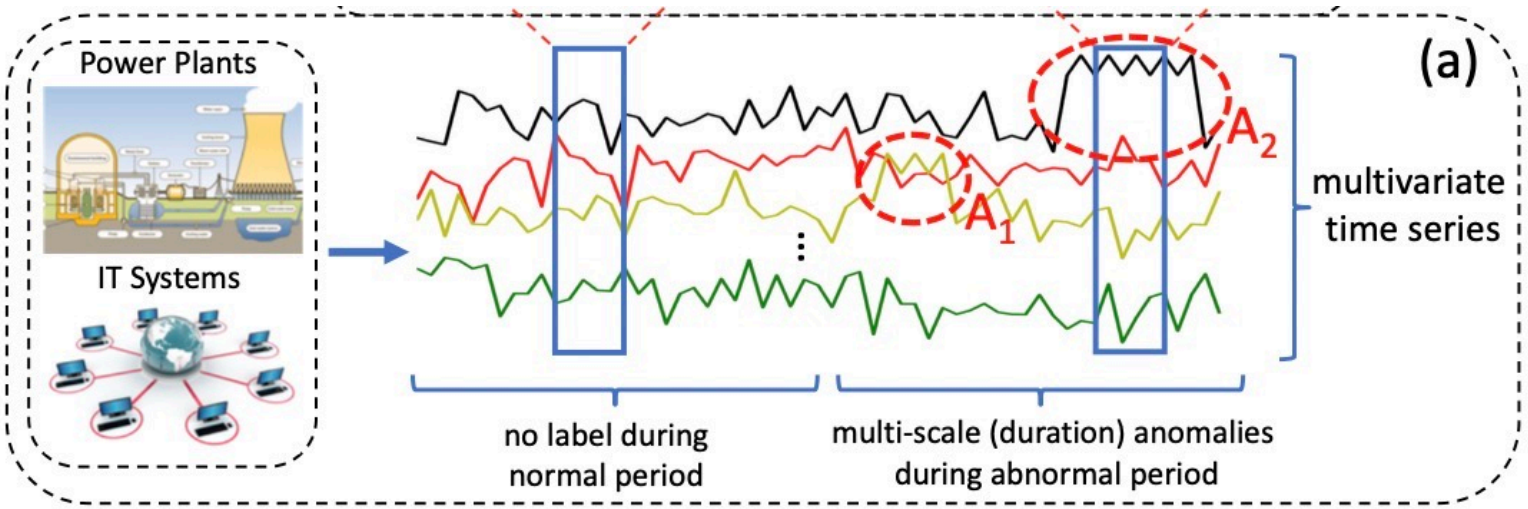
Sensor3: [-1.32, -0.94, -0.98, -1.28, -0.91, -1.54, -1.13, -1.0, -0.49, -1.03, -0.59, -1.35, -1.24, -1.01, -1.29]

Sensor4: [0.74, 0.37, 0.19, 0.22, -0.16, 0.3, 1.15, 0.79, 0.78, 0.98, 0.08, 1.02, 0.23, 0.61, 0.73]

Sensor5: [0.62, 0.47, 0.77, 0.43, 0.39, 0.5, 0.03, 0.89, 0.68, 0.36, 0.82, 0.55, 0.59, 0.98, 0.67]



我们要解决的问题？



正常情况下，某些传感器之间互相影响，呈现某种特定形式，但在某一个时刻，如果发生了异常，
必然会导致某些传感器发生突变（有些可能不变），那么这种异常一定会表现在不同传感器之间的关系上，
我们的目的是及时发现这类异常。

作者的方法不同于单变量的时序异常检测（单变量是检测偏离正常分布情况）。该方法是作用在多变量上（传感器），
考虑不同传感器之间的相关性，通过相关性来进行异常检测。

那么，我们要计算某一个时刻，多个传感器之间的相似度，如果仅仅考虑这一个时刻，那么就只有一个点的数据。一个比较直观的想法，就是考虑该时刻之前的某段时间，计算这段时间的相似程度。

Sensor1: [-1.2, -1.16, -0.81, -0.77, -0.85, -0.62, -0.98, -1.23, -0.89, -0.89, -1.17, -1.26, -0.55, -0.74, -1.37]

Sensor2: [-1.41, -1.02, -0.66, -1.23, -1.25, -0.88, -0.93, -0.97, -1.31, -0.31, -0.96, -1.16, -0.83, -0.94, -1.58]

t

ω – 时间步长

$$\mathbf{x}_i^w = (x_i^{t-w}, x_i^{t-w+1}, \cdots, x_i^t) \qquad X_1^{10} = [-0.62, -0.98, -1.23, -0.89, -0.89, -1.17, -1.26, -0.55, -0.74, -1.37]$$

i-sensor

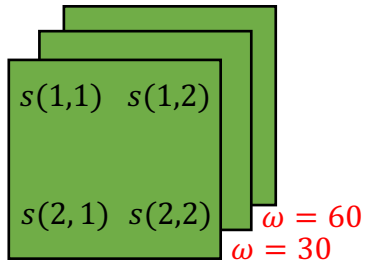
$$\mathbf{x}_j^w = (x_j^{t-w}, x_j^{t-w+1}, \cdots, x_j^t) \qquad X_2^{10} = [-0.88, -0.93, -0.97, -1.31, -0.31, -0.96, -1.16, -0.83, -0.94, -1.58]$$

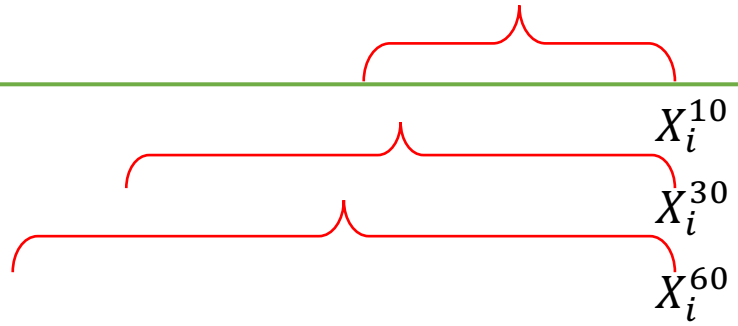
ω时间步

$$m_{ij}^t = \frac{\sum_{\delta=0}^{w} x_i^{t-\delta} x_j^{t-\delta}}{\kappa} \qquad (1)$$

内积

where κ is a rescale factor ($\kappa = w$).





如果我们有1355个时间序列，30个传感器，$\omega = \{10,30,60\}$
则每个时间点有[30, 30, 3]维数据。输入数据维度为[1355, 30, 30, 3]

算法思路：

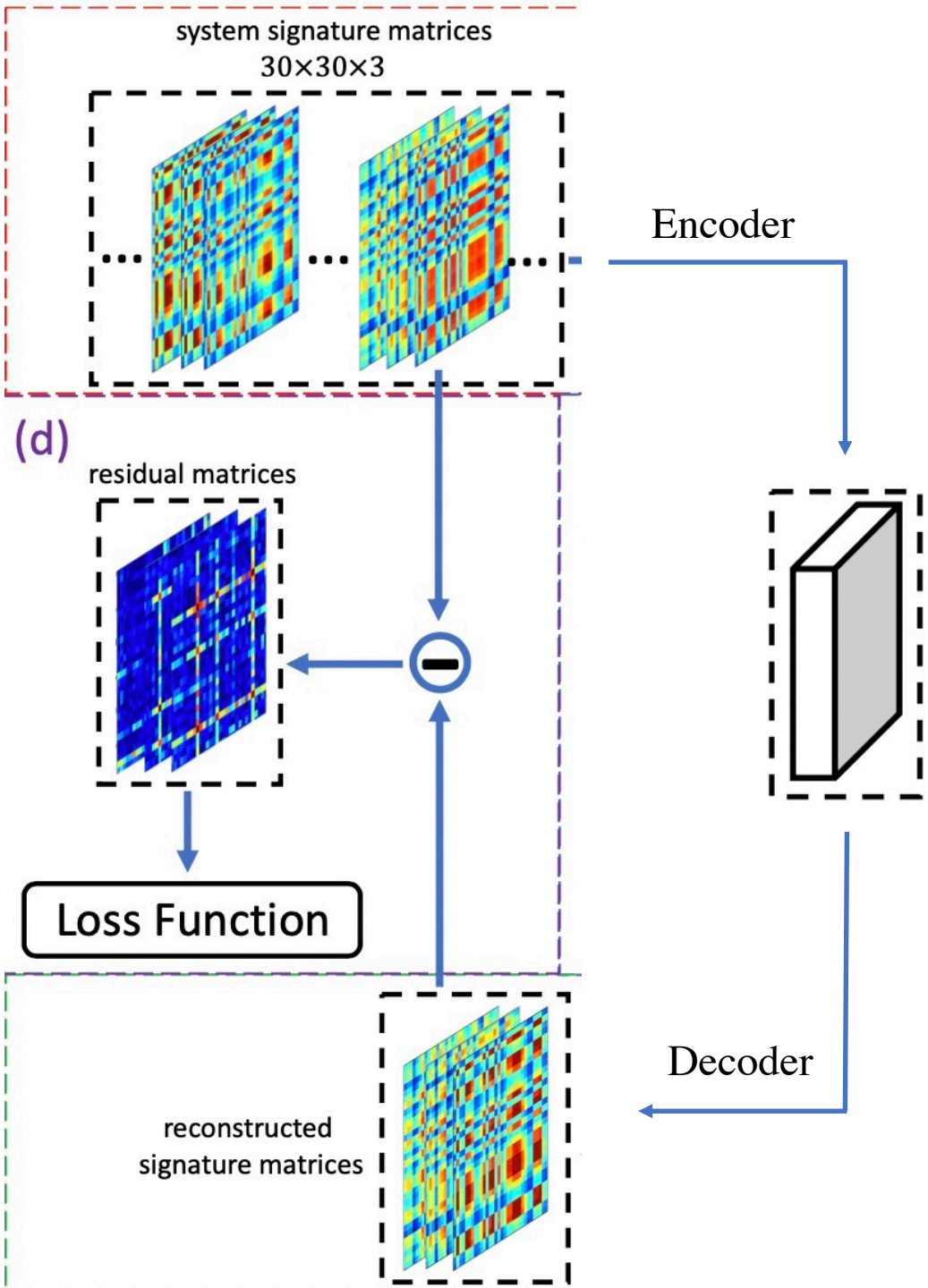
1355个时间点，一共30个传感器，$\omega = \{10,30,60\}$

通过残差来计算loss，
并根据残差来计算异常

Convolutional Encoder

Attention based ConvLSTM

Convolutional Decoder



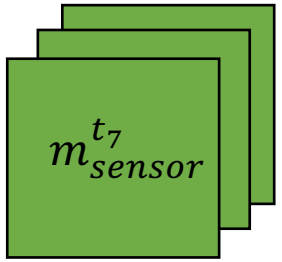
Convolutional Encoder

第0层 n: $sensor$数量

$$\mathcal{X}^{t,0} \in \mathbb{R}^{n \times n \times s}$$

s: 时间步数量

特征(相似矩阵)

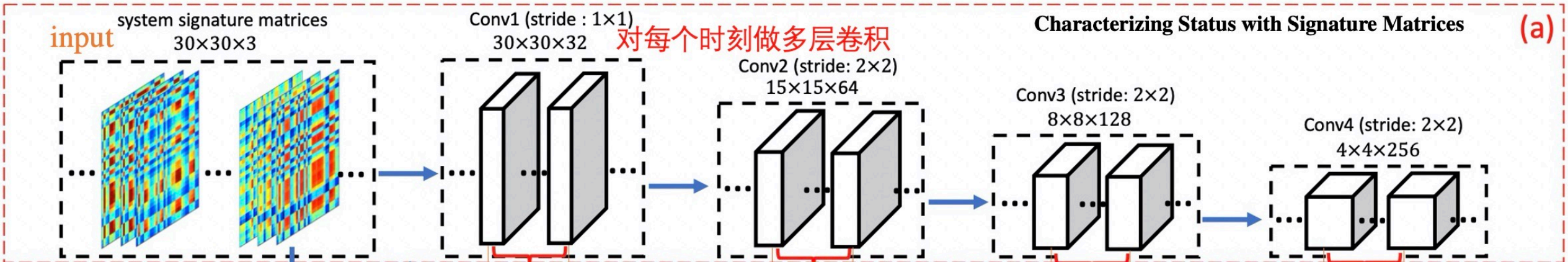


s=3

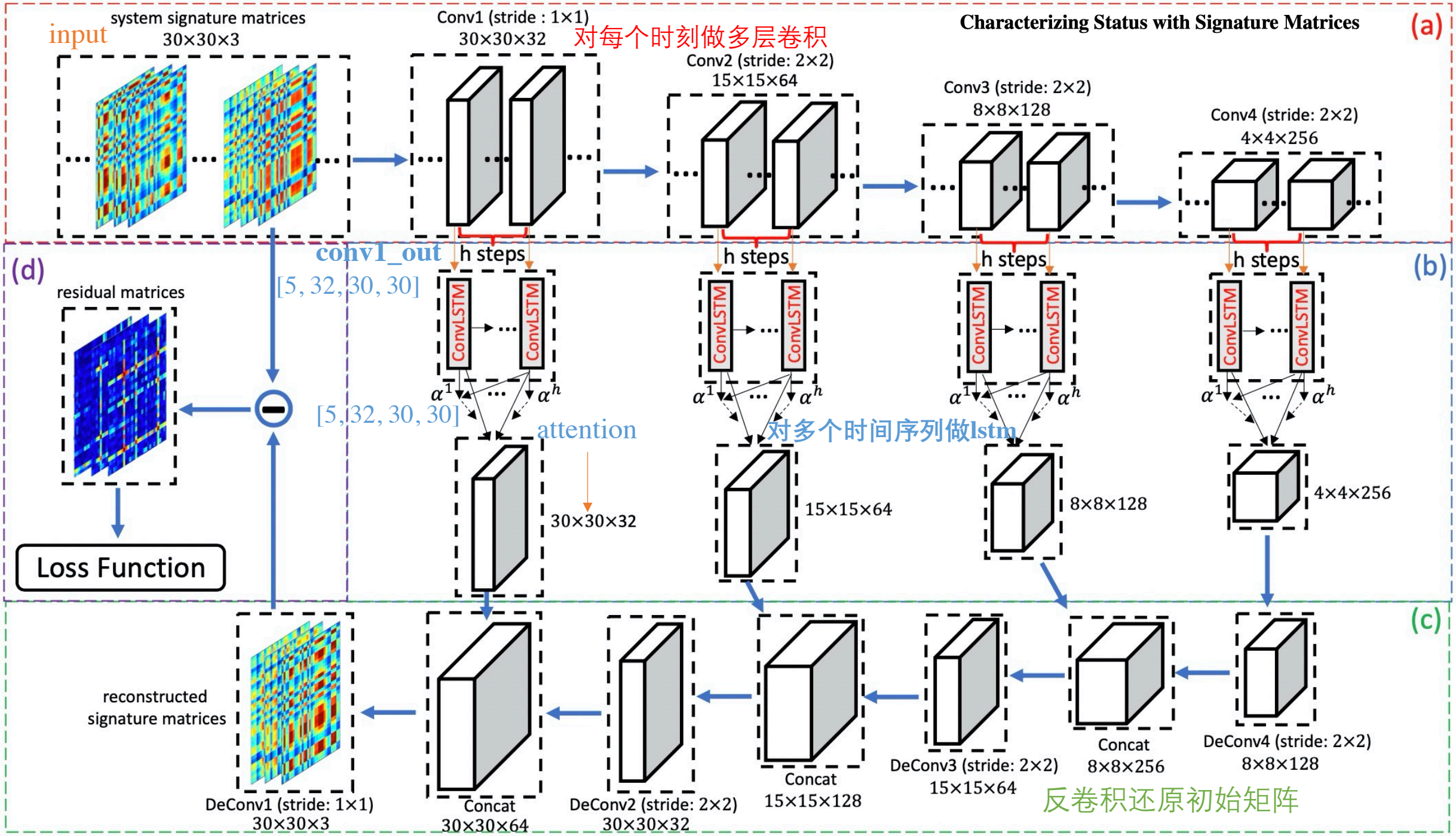
[30, 30, 3]

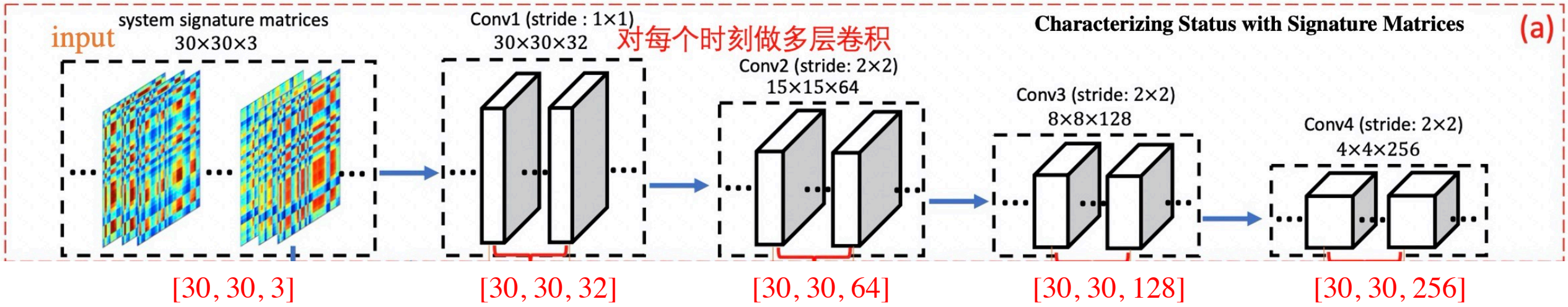
$$\mathcal{X}^{t,l} = f(W^l * \mathcal{X}^{t,l-1} + b^l) \qquad (2)$$

卷积操作　　每层间的更新迭代

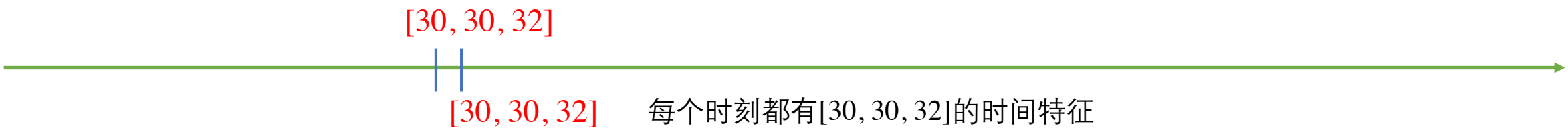


[1355, 30, 30, 3]


input
system signature matrices
30×30×3
Conv1 (stride : 1×1)
30×30×32
对每个时刻做多层卷积
Characterizing Status with Signature Matrices
(a)
Conv2 (stride: 2×2)
15×15×64
Conv3 (stride: 2×2)
8×8×128
Conv4 (stride: 2×2)
4×4×256
conv1_out
[5, 32, 30, 30]
h steps
(b)
(d)
residual matrices
ConvLSTM
attention
对多个时间序列做lstm
[5, 32, 30, 30]
30×30×32
15×15×64
8×8×128
4×4×256
Loss Function
(c)
reconstructed
signature matrices
DeConv1 (stride: 1×1)
30×30×3
Concat
30×30×64
DeConv2 (stride: 2×2)
30×30×32
Concat
15×15×128
DeConv3 (stride: 2×2)
15×15×64
Concat
8×8×256
DeConv4 (stride: 2×2)
8×8×128
反卷积还原初始矩阵



时间序列数据，为了捕获历史时间的影响，一个最基本的方法就是，使用LSTM的方式去捕捉历史时间数据



但是LSTM是捕获一维特征随时间变化的情况，但是我们是三维特征随时间变量的情况？
于是作者采用ConvLSTM的方式来进行计算。

Attention based ConvLSTM

$$\mathcal{H}^{t,l} = \text{ConvLSTM}(\mathcal{X}^{t,l}, \mathcal{H}^{t-1,l})$$

卷积操作

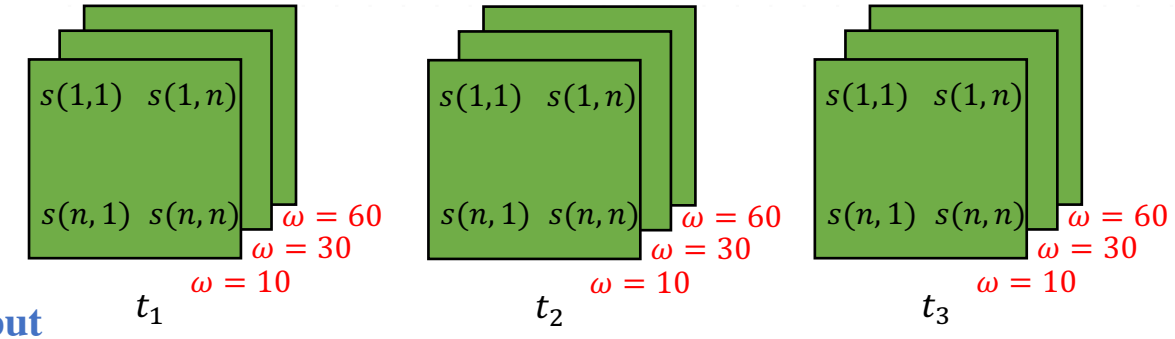
inputGate $$\mathbf{z}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XZ}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HZ}} * \mathcal{H}^{t-1,l} + \tilde{W}^{k}_{\mathcal{CZ}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^{l}_{\mathcal{Z}})$$

forgetGate $$\mathbf{r}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XR}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HR}} * \mathcal{H}^{t-1,l} + \tilde{W}^{l}_{\mathcal{CR}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^{l}_{\mathcal{R}})$$

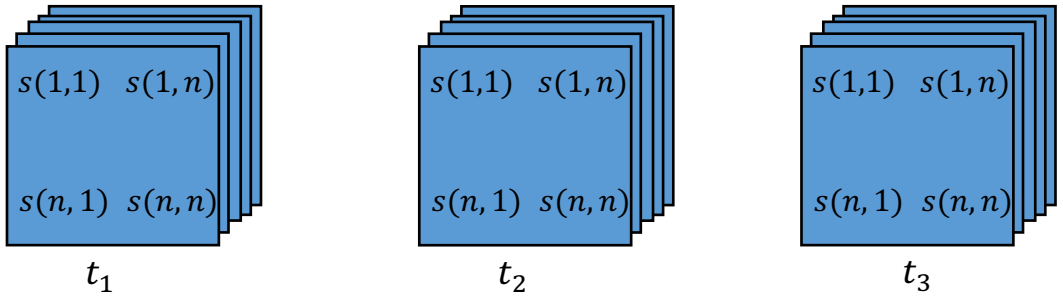
$$\mathcal{C}^{t,l} = \mathbf{z}^{t,l} \circ \tanh(\tilde{W}^{l}_{\mathcal{XC}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HC}} * \mathcal{H}^{t-1,l} + \tilde{b}^{l}_{\mathcal{C}}) + \mathbf{r}^{t,l} \circ \mathcal{C}^{t-1,l}$$

$$\mathbf{o}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XO}} * \mathcal{X}^{t,l} + \tilde{W}_{\mathcal{HO}} * \mathcal{H}^{t-1,l} + \tilde{W}_{\mathcal{CO}} \circ \mathcal{C}^{t,l} + \tilde{b}^{l}_{\mathcal{O}})$$

$$\mathcal{H}^{t,l} = \mathbf{o}^{t,l} \circ \tanh(\mathcal{C}^{t,l}) \qquad (3)$$



output



LSTM

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

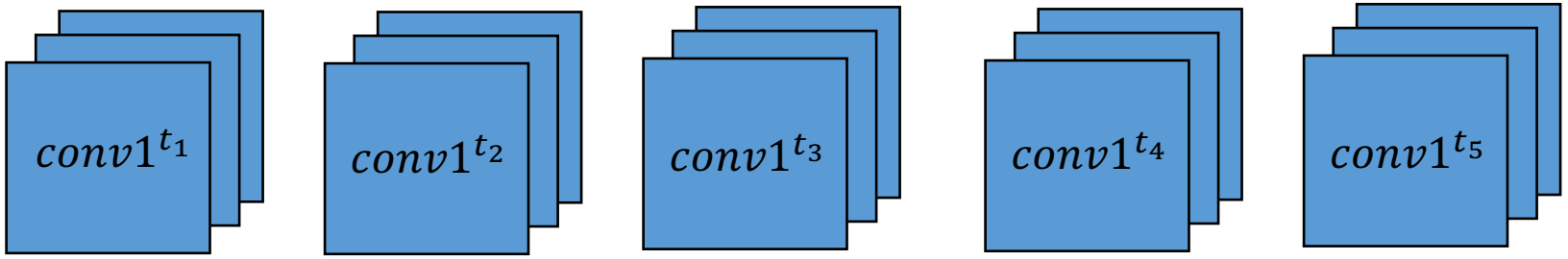
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$



$\mathcal{H}^{t_1,l} = [32, 30, 30]$ $\mathcal{H}^{t_4,l} = [32, 30, 30]$ $\mathcal{H}^{t_5,l} = [32, 30, 30]$

$$\hat{\mathcal{H}}^{t,l} = \sum_{i\in(t-h,t)} \alpha^i \mathcal{H}^{i,l}, \alpha^i = \frac{\exp\{\frac{\mathrm{Vec}(\mathcal{H}^{t,l})^{\mathrm{T}}\mathrm{Vec}(\mathcal{H}^{i,l})}{\chi}\}}{\sum_{i\in(t-h,t)}\exp\{\frac{\mathrm{Vec}(\mathcal{H}^{t,l})^{\mathrm{T}}\mathrm{Vec}(\mathcal{H}^{i,l})}{\chi}\}} \quad (4)$$

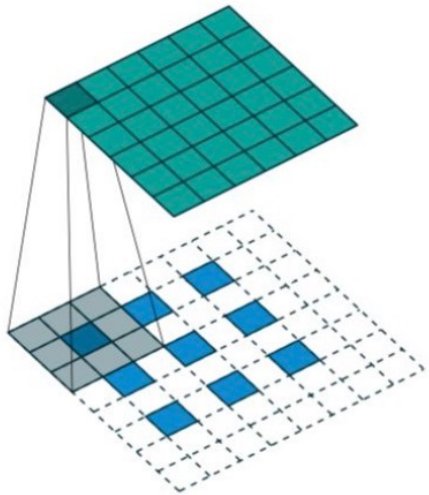
默认$h = 5$

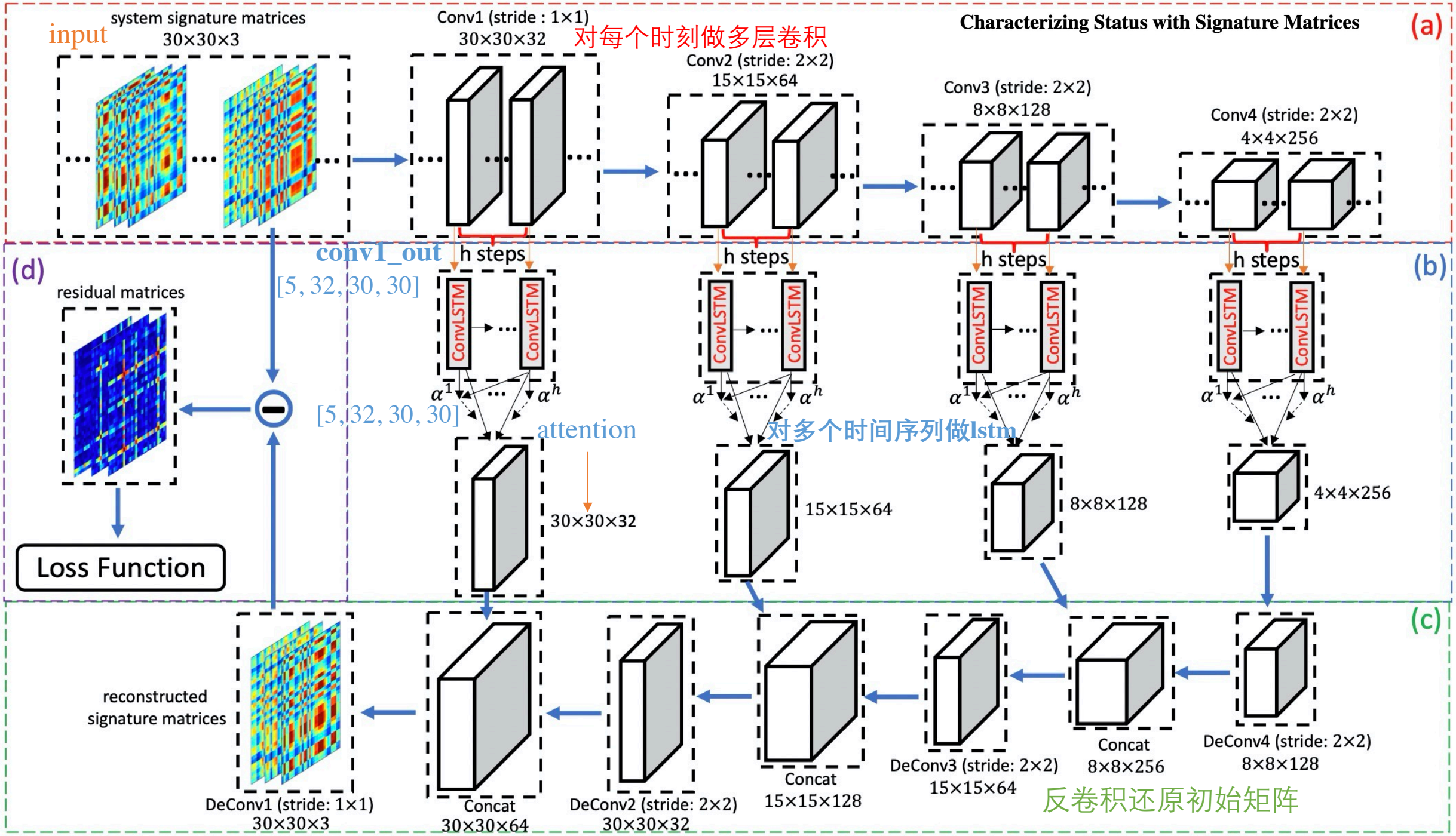
对每个时刻H^i和H^t(最后一个时刻)计算$attention$系数

Convolutional Decoder

反卷积

$$\hat{\mathcal{X}}^{t,l-1} = \begin{cases} f(\hat{W}^{t,l} \circledast \hat{\mathcal{H}}^{t,l} + \hat{b}^{t,l}), & l = 4 \\ f(\hat{W}^{t,l} \circledast [\hat{\mathcal{H}}^{t,l} \oplus \hat{\mathcal{X}}^{t,l}] + \hat{b}^{t,l}), & l = 3, 2, 1 \end{cases} \quad (5)$$




input
system signature matrices
30×30×3
Conv1 (stride : 1×1)
30×30×32
对每个时刻做多层卷积
Characterizing Status with Signature Matrices
(a)
Conv2 (stride: 2×2)
15×15×64
Conv3 (stride: 2×2)
8×8×128
Conv4 (stride: 2×2)
4×4×256
conv1_out
[5, 32, 30, 30]
h steps
(b)
(d)
residual matrices
ConvLSTM
attention
对多个时间序列做lstm
[5, 32, 30, 30]
30×30×32
15×15×64
8×8×128
4×4×256
Loss Function
(c)
reconstructed
signature matrices
DeConv1 (stride: 1×1)
30×30×3
Concat
30×30×64
DeConv2 (stride: 2×2)
30×30×32
Concat
15×15×128
DeConv3 (stride: 2×2)
15×15×64
Concat
8×8×256
DeConv4 (stride: 2×2)
8×8×128
反卷积还原初始矩阵

Loss Function

$$\mathcal{L}_{MSCRED} = \sum_{t}\sum_{c=1}^{s}\left\|\mathcal{X}_{:,:,c}^{t,0} - \hat{\mathcal{X}}_{:,:,c}^{t,0}\right\|_F^2 \qquad (6)$$

重构误差

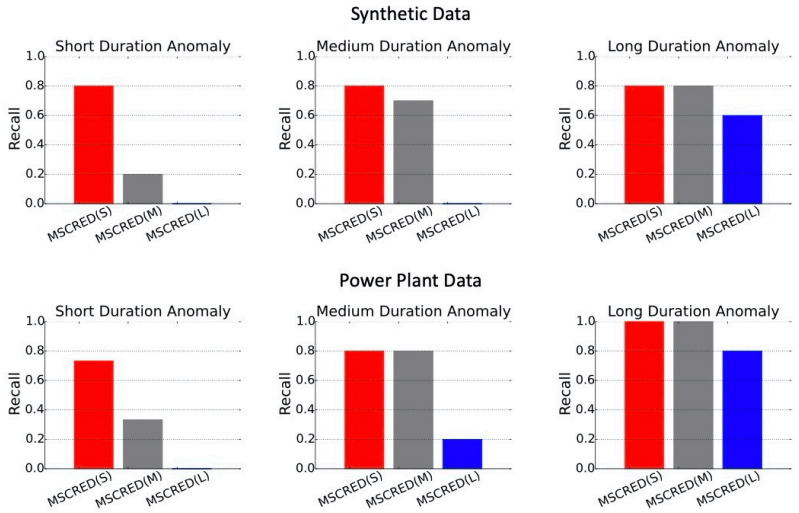


Figure 6: Performance of three channels of MSCRED over different types of anomalies.

异常定义

异常评分被定义为重建不良的两两相关的数量。

换句话说，在不同的数据集上，通过经验确定残差特征矩阵中大于给定阈值θ的元素个数和θ。

```
#compute number of broken element in residual matrix
select_matrix_error = np.square(np.subtract(select_gt_matrix, select_reconstructed_matrix))
num_broken = len(select_matrix_error[select_matrix_error > thred_b])
```

Θ=0.005

训练好的模型，可以进行在线预测，只需要训练集后，就可以预测测试集的效果（当然模型应该经常训练更新为好）

异常识别效果

Table 2: Anomaly detection results on two datasets.

Method	Synthetic Data			Power Plant Data		
	Pre	Rec	F_1	Pre	Rec	F_1
OC-SVM	0.14	0.44	0.22	0.11	0.28	0.16
DAGMM	0.33	0.20	0.25	0.26	0.20	0.23
HA	0.71	0.52	0.60	0.48	0.52	0.50
ARMA	0.91	0.52	0.66	0.58	0.60	0.59
LSTM-ED	1.00	0.56	0.72	0.75	0.68	0.71
$\text{CNN}^{ED(4)}_{ConvLSTM}$	0.37	0.24	0.29	0.67	0.56	0.61
$\text{CNN}^{ED(3,4)}_{ConvLSTM}$	0.63	0.56	0.59	0.80	0.72	0.76
$\text{CNN}^{ED}_{ConvLSTM}$	0.80	0.76	0.78	0.85	0.72	0.78
MSCRED	**1.00**	**0.80**	**0.89**	**0.85**	**0.80**	**0.82**
Gain (%)	–	30.0	23.8	13.3	19.4	15.5

效果如何

每个成分对效果的影响

$\text{CNN}^{ED(4)}_{ConvLSTM}$ 只保留ConvLSTM第4个Conv

$\text{CNN}^{ED(3,4)}_{ConvLSTM}$ 只保留ConvLSTM第3，4个Conv

$\text{CNN}^{ED}_{ConvLSTM}$ 移除整个attention模块

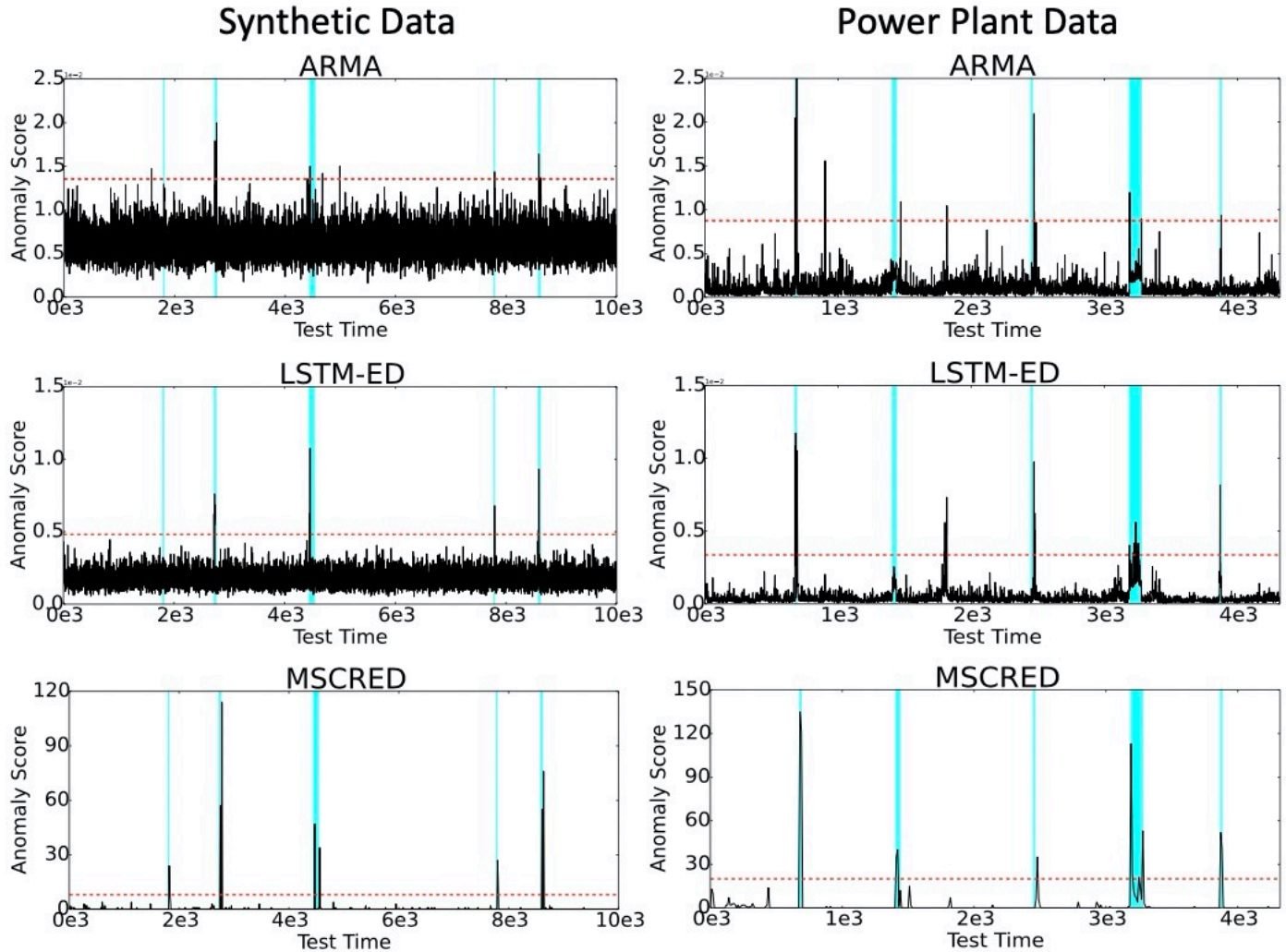


Figure 3: Case study of anomaly detection. The shaded regions represent anomaly periods. The red dash line is the cutting threshold of anomaly.

其他的方法效果不稳定，包含没有识别或误识别情况
但MSCRED方法识别率和准确率很高

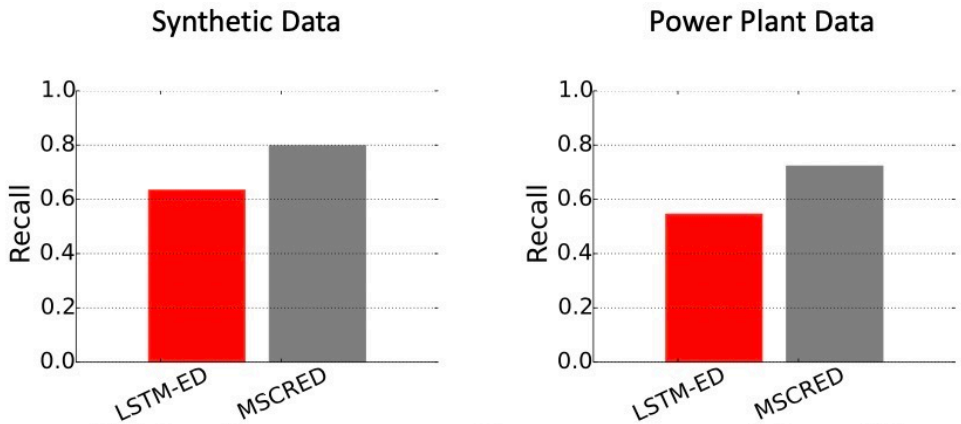


Figure 5: Performance of root cause identification.

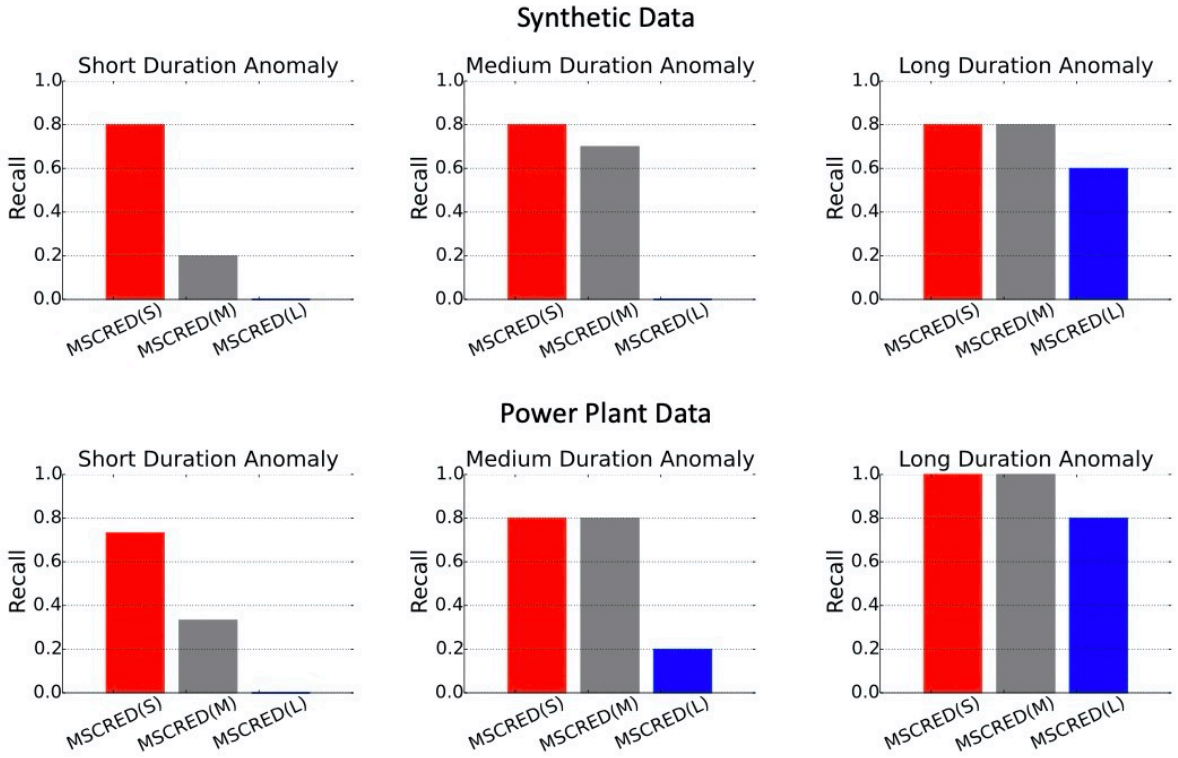
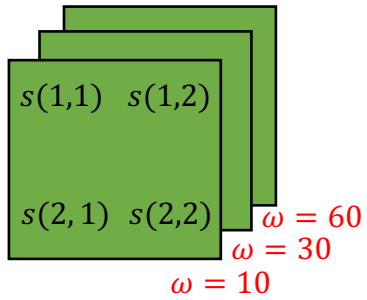


Figure 6: Performance of three channels of MSCRED over different types of anomalies.



因为我们在计算不同sensor在某时刻相似度时，分别计算了不同时间步长的情况$\omega = \{10,30,60\}$。所以，分别表示了异常持续的时间。

L表示持续时间比较长的异常 $\omega = \{60\}$
S表示持续时间比较短的异常 $\omega = \{10\}$

MSCRED(S)和MSCRED(M)可以侦测大部分异常，但是MSCRED(L)有时并不能侦测异常

因此，我们可以综合考虑三个异常评分表达了异常的严重程度。如果能在所有三个通道中检测到异常，则异常更有可能持续时间较长。

最终，我们如果考虑异常的召回率，尽量选择$\omega = \{10\}$这种方案，以增加召回率

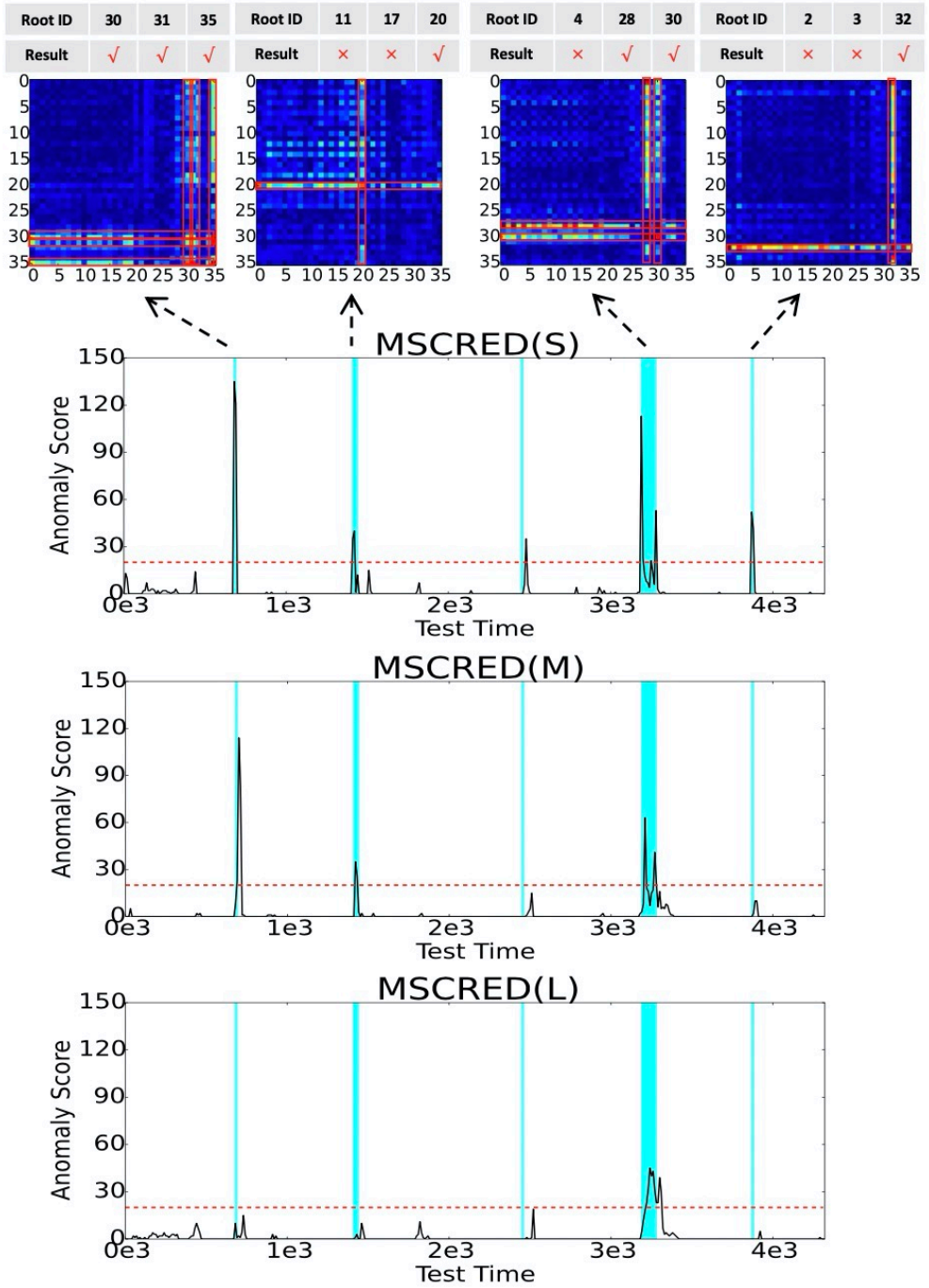


Figure 7: Case study of anomaly diagnosis.

可视化sensor之间的相似性矩阵

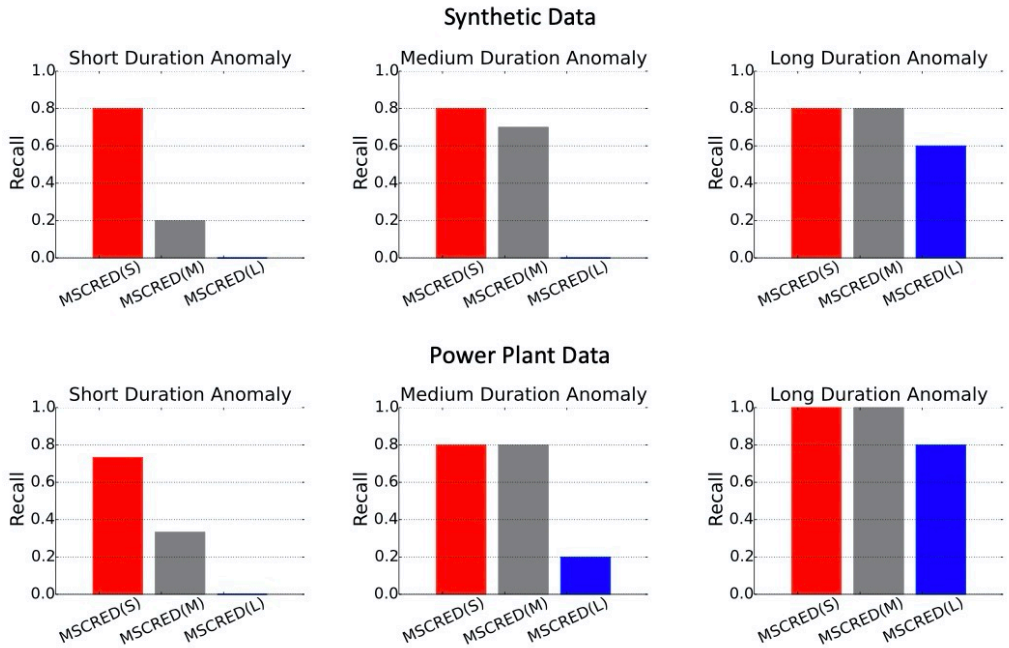


Figure 6: Performance of three channels of MSCRED over different types of anomalies.

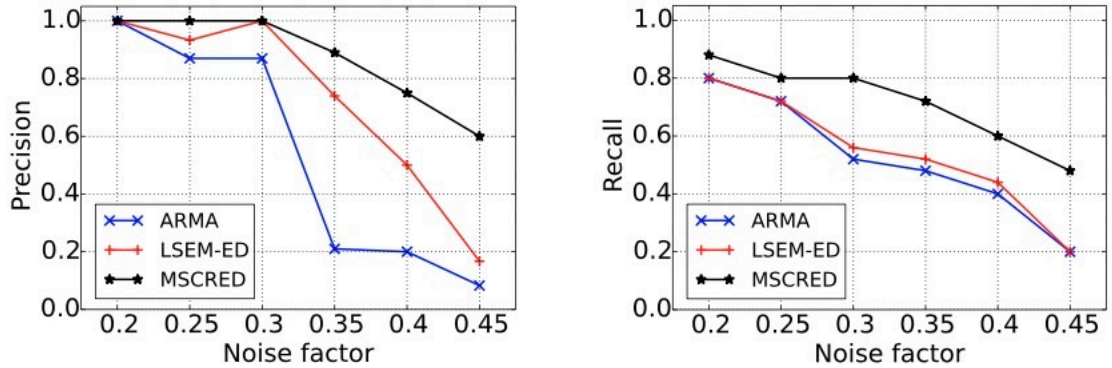
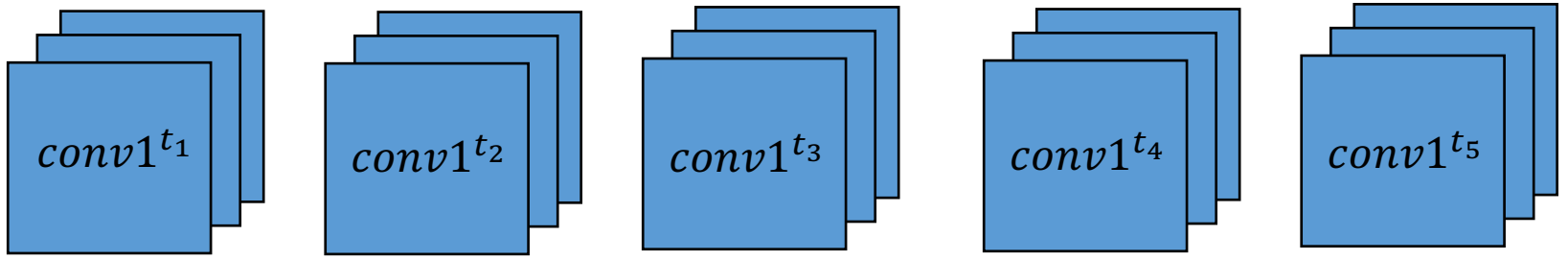


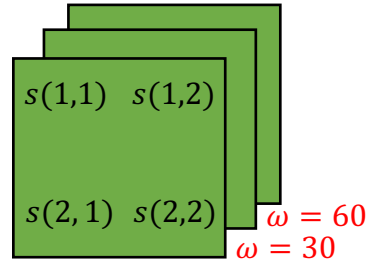
Figure 8: Impact of data noise on anomaly detection.

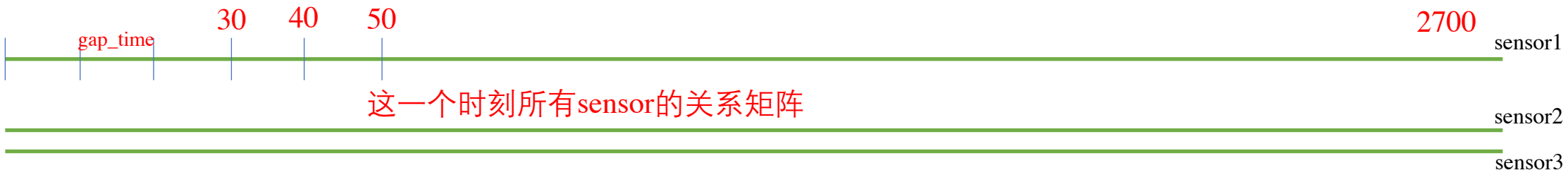
对于噪声鲁棒性

Code




```
ts_type = args.ts_type
step_max = args.step_max  # LSTM和attentino时，向前回溯的步长
min_time = args.min_time
max_time = args.max_time
gap_time = args.gap_time  # 多少时间间隔检测一次; gap_time=10
win_size = args.win_size  # windowSize:[10,30,60]
```

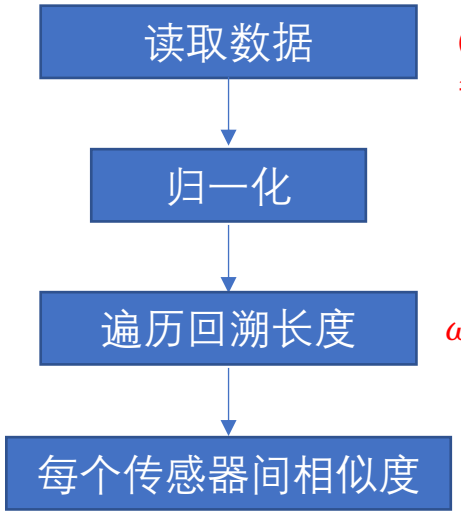




形成(270, 30, 30, 3)的特征矩阵

Code

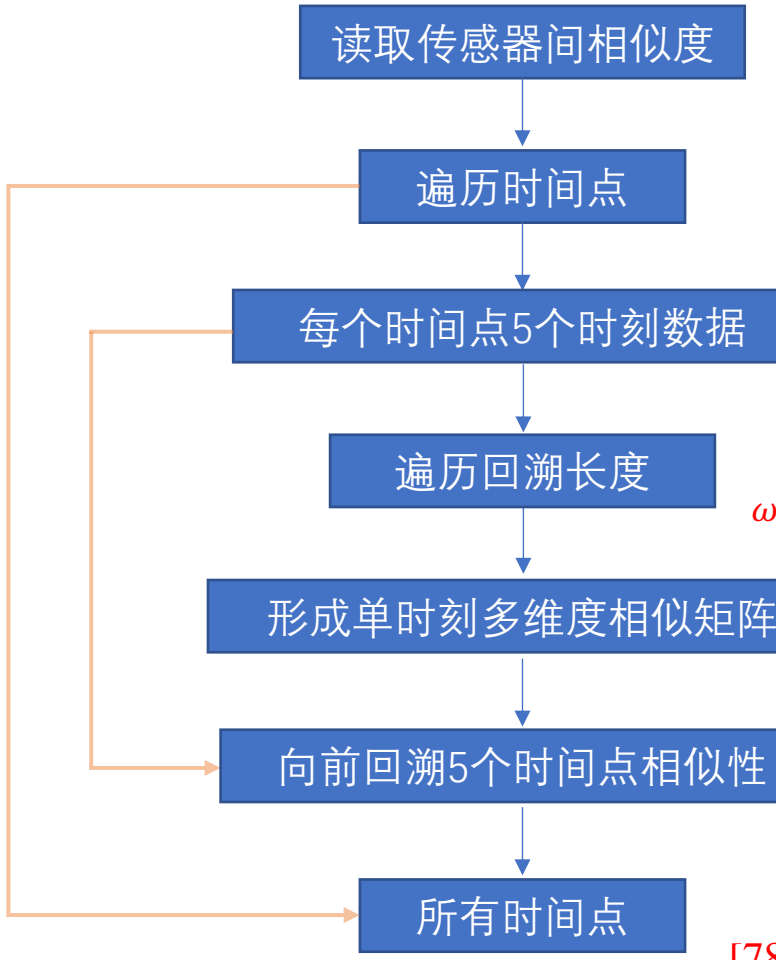
首先进行数据生成工作 (matrix_generator.py)



(30, 20000): 30个传感器，
每个20000个时间序列

$\omega = (10, 30, 60)$

```
ts_type = args.ts_type  ts_type: 'node'
step_max = args.step_max  step_max: 5
min_time = args.min_time  min_time: 0
max_time = args.max_time  max_time: 20000
gap_time = args.gap_time  # 多少时间间隔检测一次  ga
win_size = args.win_size  # windowSize:[10,30,60]
```

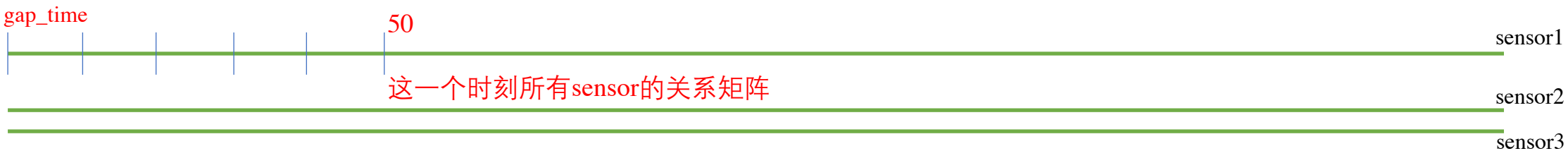


$\omega = (10, 30, 60)$

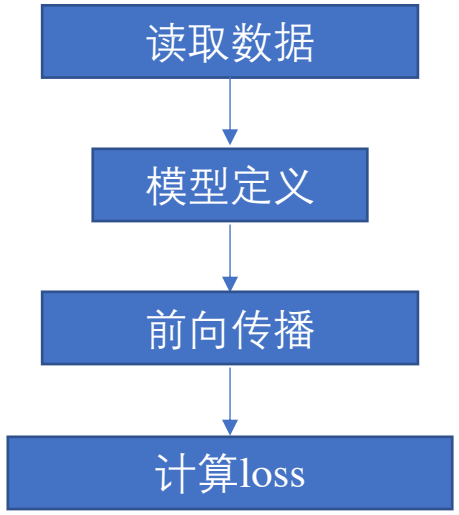
[3, 30, 30]

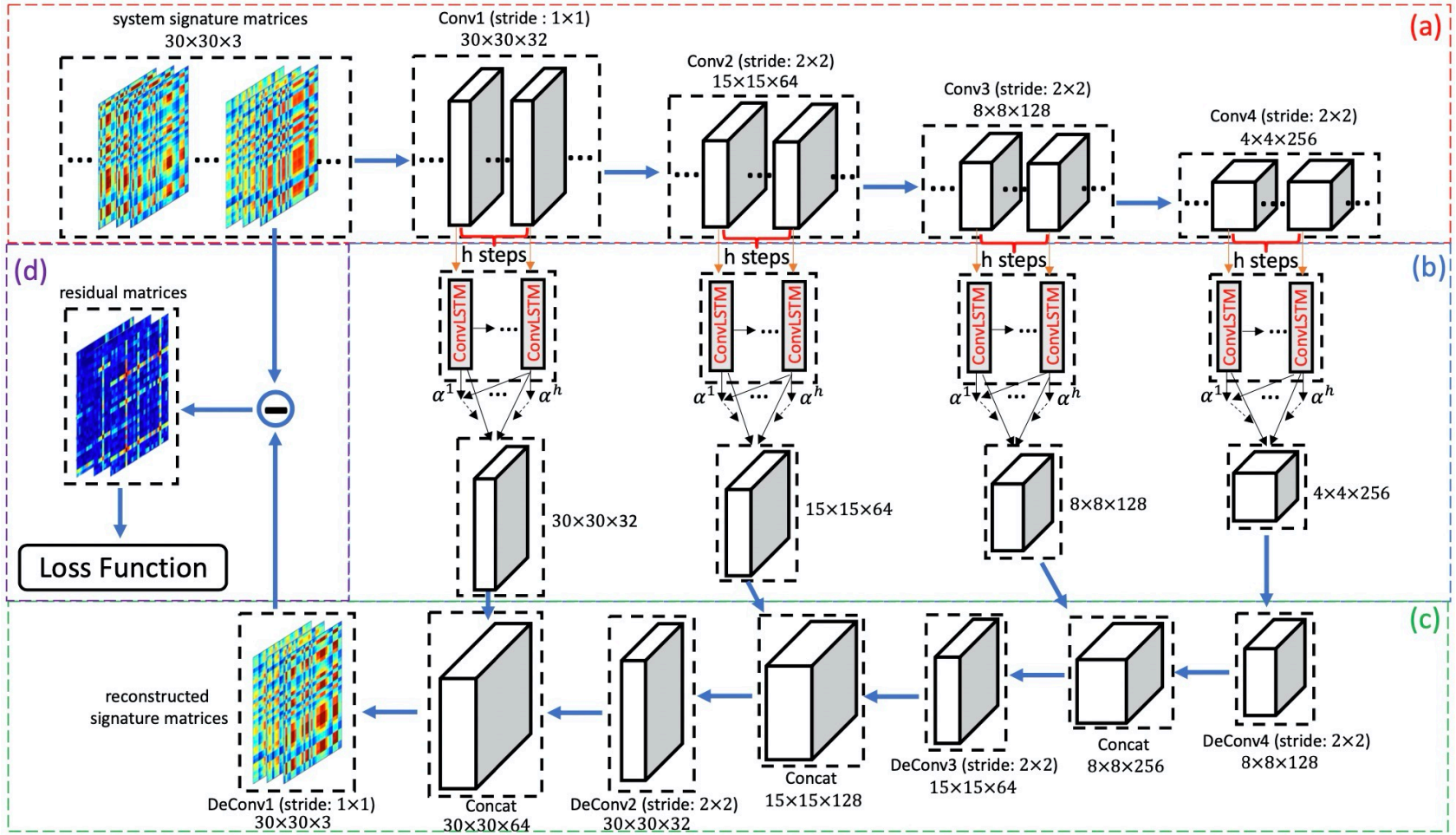
[5, 3, 30, 30]

[789, 5, 3, 30, 30]

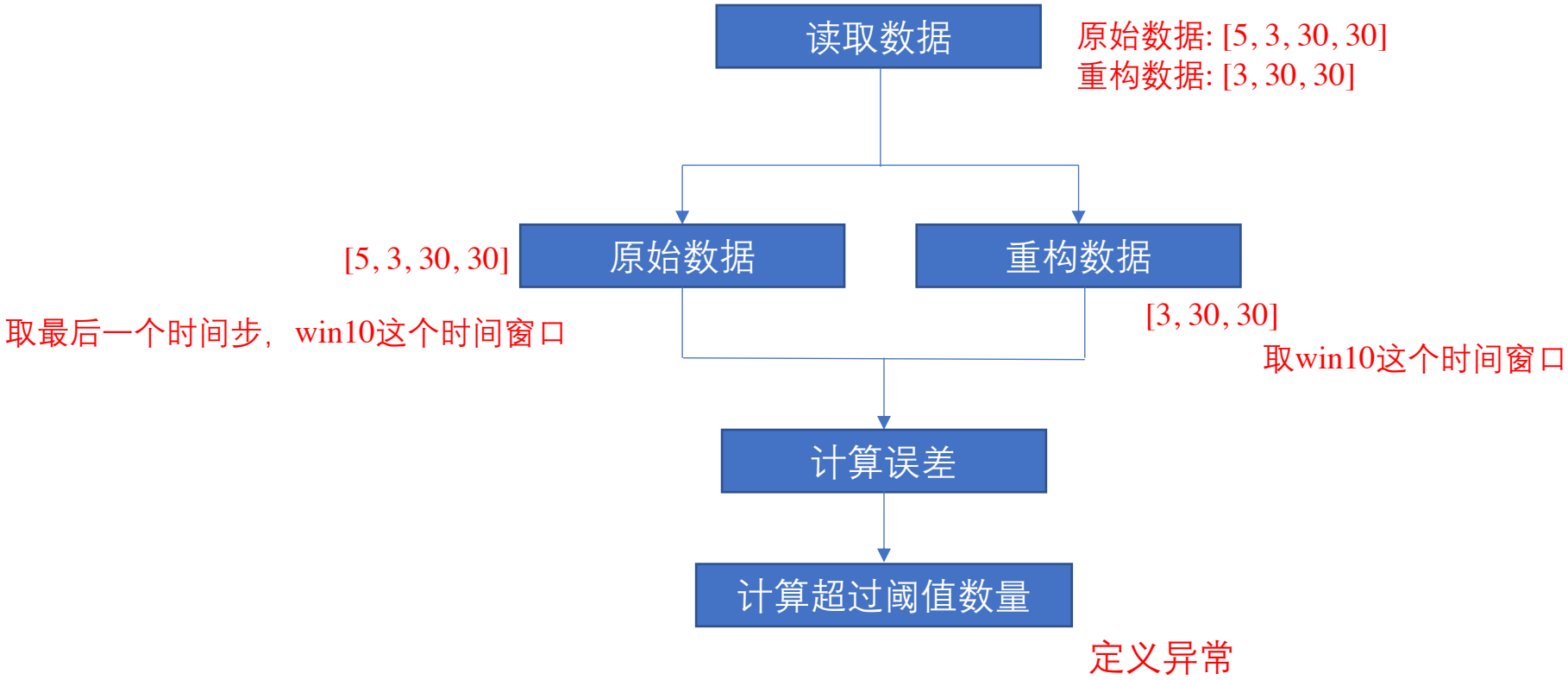


模型训练 (main.py)


读取数据
模型定义
前向传播
计算loss


(a)
system signature matrices
30×30×3
Conv1 (stride : 1×1)
30×30×32
Conv2 (stride: 2×2)
15×15×64
Conv3 (stride: 2×2)
8×8×128
Conv4 (stride: 2×2)
4×4×256
(b)
h steps
ConvLSTM
α^1
α^h
30×30×32
15×15×64
8×8×128
4×4×256
(c)
DeConv4 (stride: 2×2)
8×8×128
Concat
8×8×256
DeConv3 (stride: 2×2)
15×15×64
Concat
15×15×128
DeConv2 (stride: 2×2)
30×30×32
Concat
30×30×64
DeConv1 (stride: 1×1)
30×30×3
reconstructed
signature matrices
(d)
residual matrices
Loss Function

异常检测 (evaluate.py)

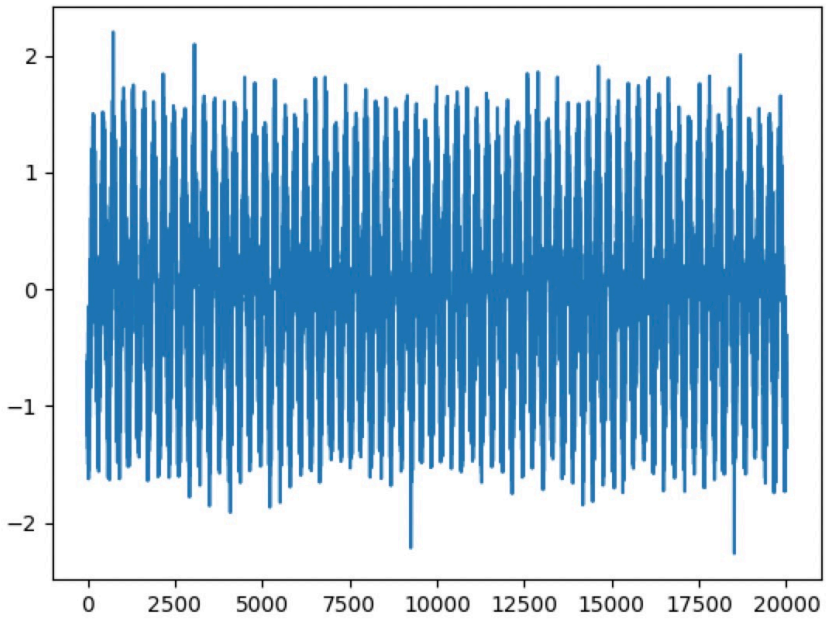


Code

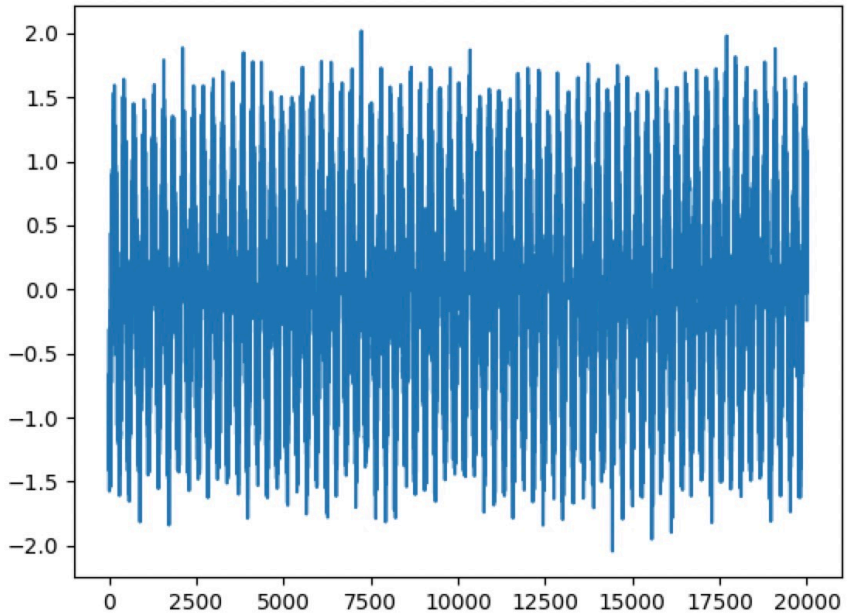
```
win_size = args.win_size  # windowSize:[10,30,60]
```

mances on three types of anomalies, *i.e.*, short, medium, and long with the duration of 10, 30, and 60, respectively.

Data[0]



Data[1]



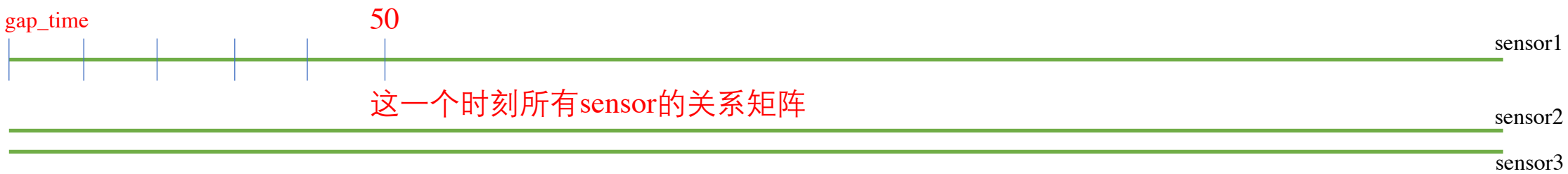
30个序列，每个序列有20000个时序点

数据预处理

```
for i in range(sensor_n):  i: 0
    for j in range(i, sensor_n):  j: 0
        # 计算该时刻，每个sensor之间的相似关系
        matrix_t[i][j] = np.inner(data[i, t - win:t], data[j, t - win:t])/(win)
        matrix_t[j][i] = matrix_t[i][j]  # 对称矩阵
```

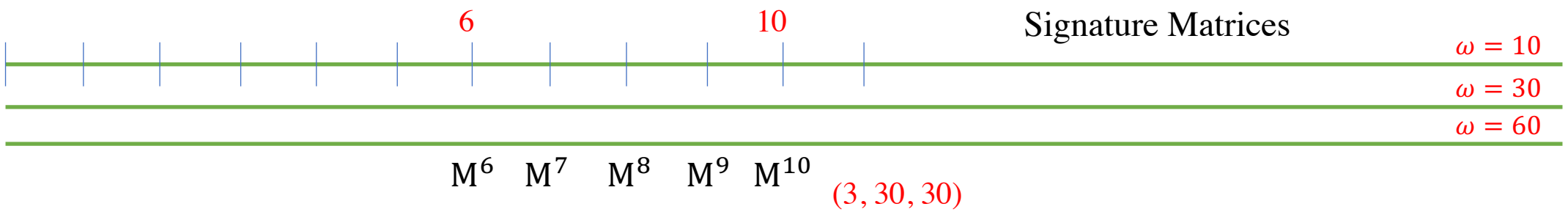
$$\mathbf{x}_i^w = (x_i^{t-w}, x_i^{t-w-1}, \cdots, x_i^t)$$
$$(x_j^{t-w}, x_j^{t-w-1}, \cdots, x_j^t)$$

$$m_{ij}^t = \frac{\sum_{\delta=0}^{w} x_i^{t-\delta} x_j^{t-\delta}}{\kappa} \quad (1)$$



因为在回溯win窗口时，最小窗口和gap相等，所以这段时间内的异常有表示

时序预处理



$$\mathcal{X}^{10} = (M^6, M^7, M^8, M^9, M^{10})$$

(5, 3, 30, 30)

```
# multi-scale signature matix generation
for w in range(len(win_size)):  # 多尺度特征生成
    matrix_all = []
    win = win_size[w]  # 向前推w时间点
    print ("generating signature with window " + str(win) + "...")
    for t in range(min_time, max_time, gap_time):  # ??? 造数据，构建相似矩阵
        #print t
        matrix_t = np.zeros((sensor_n, sensor_n))
        if t >= 60:
            for i in range(sensor_n):
                for j in range(i, sensor_n):
                    #if np.var(data[i, t - win:t]) and np.var(data[j, t - win:t]):
                    matrix_t[i][j] = np.inner(data[i, t - win:t], data[j, t - win:t])/(win)
                    matrix_t[j][i] = matrix_t[i][j]
        matrix_all.append(matrix_t)  # 每个时刻的Signature Matrices
    path_temp = matrix_data_path + "matrix_win_" + str(win)
    np.save(path_temp, matrix_all)
    del matrix_all[:]
```

(30, 30)

计算30个sensor的两两关系

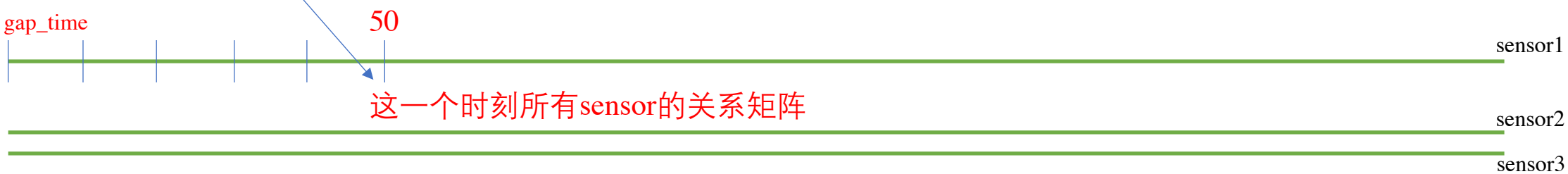
of time series in a multivariate time series segment from $t - w$ to t, we construct an $n \times n$ signature matrix M^t

```
In[6]: data.shape
Out[6]: (30, 20000)
```

$$\mathbf{x}_i^w = (x_i^{t-w}, x_i^{t-w-1}, \cdots, x_i^t)$$

$$\mathbf{x}_j^w = (x_j^{t-w}, x_j^{t-w-1}, \cdots, x_j^t)$$

$$m_{ij}^t = \frac{\sum_{\delta=0}^{w} x_i^{t-\delta} x_j^{t-\delta}}{\kappa}$$



```
np.inner(data[i, t - win:t], data[j, t - win:t])/(win)
```

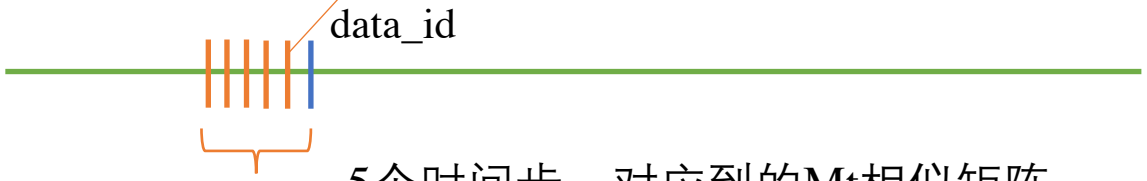
表示计算相似性考虑的时间长度

Train: 0,800

```
for i in range(len(train_test_time)):  # 训练，测试  i: 2
    for data_id in range(int(train_test_time[i][0]/gap_time), int(train_test_time[i][1]/gap_time)):
        #print data_id
        step_multi_matrix = []  step_multi_matrix: <class 'list'>: [[array([[0.10633768, 0.24020458
        for step_id in range(step_max, 0, -1):  step_id: 5
            multi_matrix = []  multi_matrix: <class 'list'>: [array([[0.10633768, 0.24020458, 0.087
            # for k in range(len(value_colnames)):
            for i in range(len(win_size)):
                multi_matrix.append(data_all[i][data_id - step_id])  # 每个win下的Mt矩阵;
            step_multi_matrix.append(multi_matrix)
```

表示lstm时，使用的时间序列长度

multi_matrix: 这个时间步下，不同win的Mt矩阵



5个时间步，对应到的Mt相似矩阵

step_multi_matrix: 这5个时间步，lstm时使用

```
In[19]: len(step_multi_matrix)
Out[19]: 5
In[20]: len(step_multi_matrix[0])
Out[20]: 3
In[21]: step_multi_matrix[0][0].shape
Out[21]: (30, 30)
```

5个时间步

3个win_size

每个win_size下Mt矩阵为(30, 30)

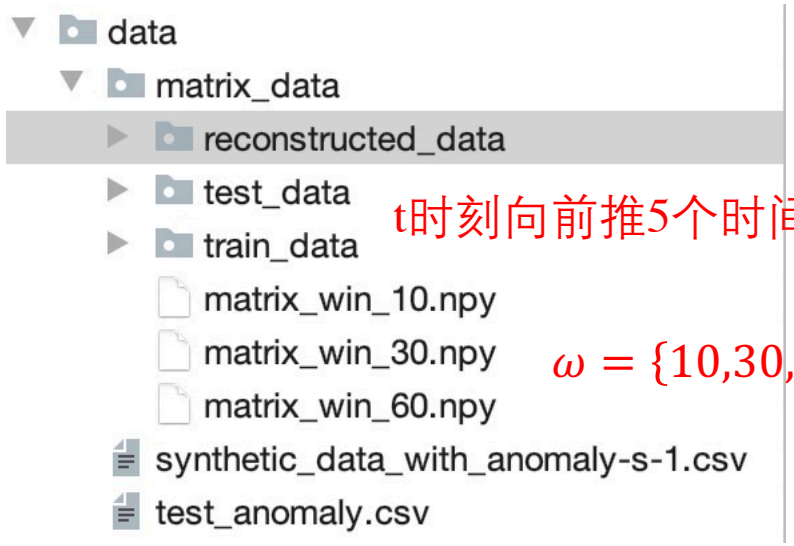
step_multi_matrix

5个时间步(时间间隔 gap_time=10)

win_size=10：15这个id下，计算该时刻下，

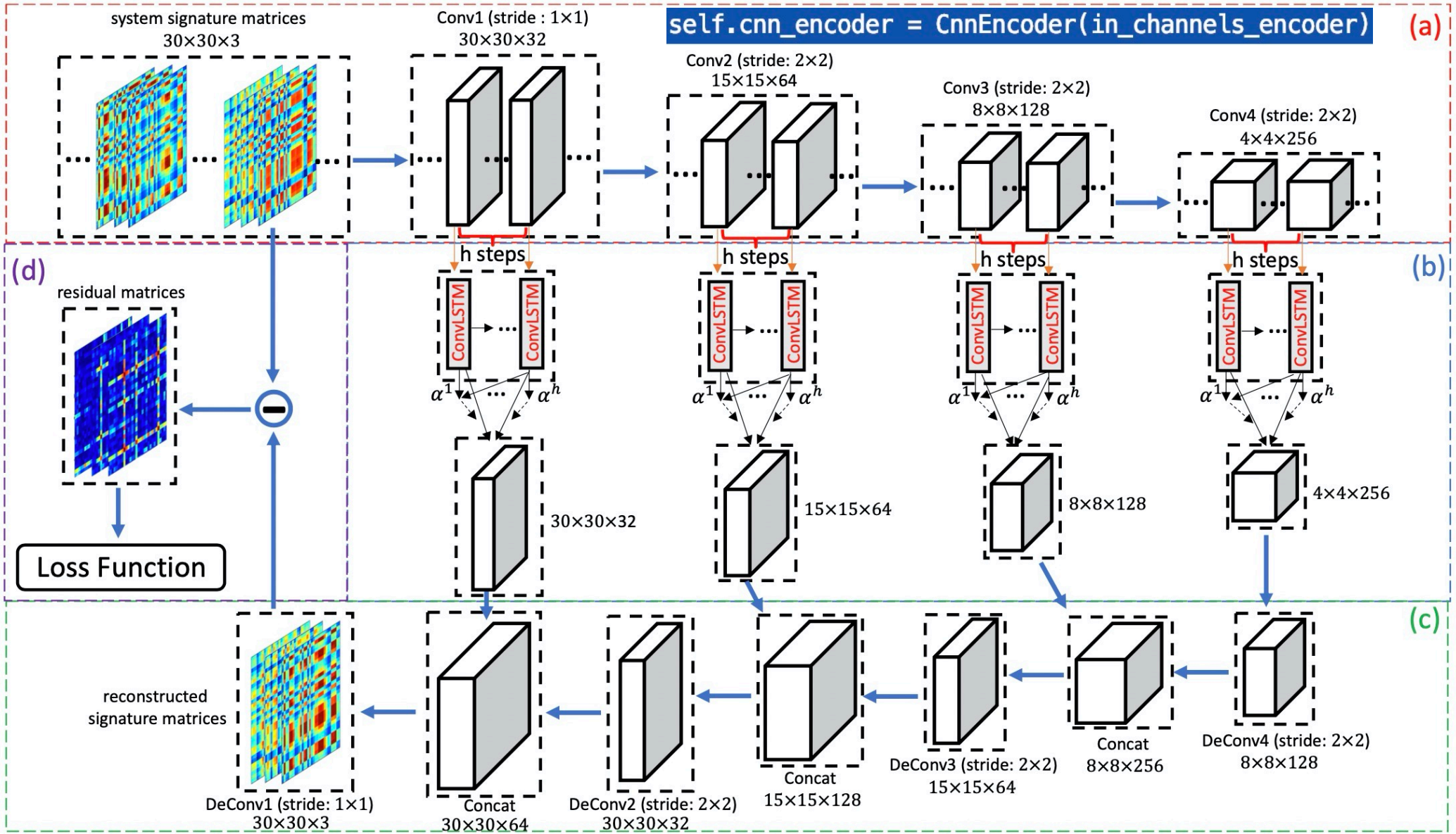
win_size=30： 所有sensor相似度

win_size=60：



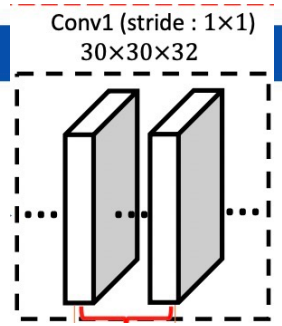
t时刻向前推5个时间步长的data

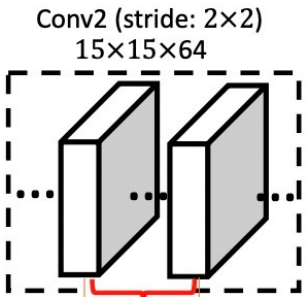
$\omega = \{10,30,60\}$: 形成的相似矩阵

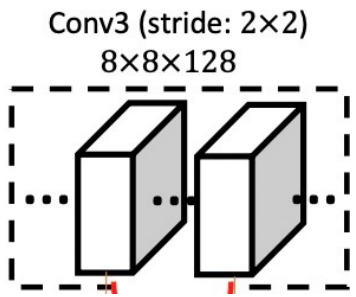

system signature matrices
30×30×3
Conv1 (stride : 1×1)
30×30×32
self.cnn_encoder = CnnEncoder(in_channels_encoder)
(a)
Conv2 (stride: 2×2)
15×15×64
Conv3 (stride: 2×2)
8×8×128
Conv4 (stride: 2×2)
4×4×256
(b)
h steps
ConvLSTM
α^1
α^h
30×30×32
15×15×64
8×8×128
4×4×256
(d)
residual matrices
Loss Function
(c)
reconstructed
signature matrices
DeConv1 (stride: 1×1)
30×30×3
Concat
30×30×64
DeConv2 (stride: 2×2)
30×30×32
Concat
15×15×128
DeConv3 (stride: 2×2)
15×15×64
Concat
8×8×256
DeConv4 (stride: 2×2)
8×8×128

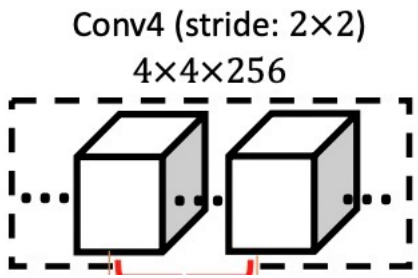
```python
class CnnEncoder(nn.Module):
    def __init__(self, in_channels_encoder):  self: CnnEncoder()  in_channels_encoder: 3
        super(CnnEncoder, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels_encoder, 32, 3, (1, 1), 1),
            nn.SELU()
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, 3, (2, 2), 1),
            nn.SELU()
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, 2, (2, 2), 1),
            nn.SELU()
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(128, 256, 2, (2, 2), 0),
            nn.SELU()
        )
    def forward(self, X):
        conv1_out = self.conv1(X)
        conv2_out = self.conv2(conv1_out)
        conv3_out = self.conv3(conv2_out)
        conv4_out = self.conv4(conv3_out)
        return conv1_out, conv2_out, conv3_out, conv4_out
```

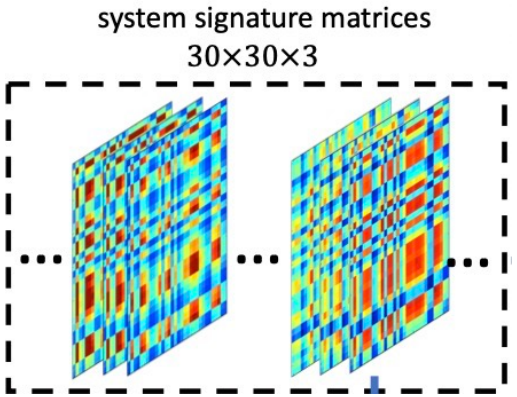
kernel_size
padding
in_channels
out_channels
stride





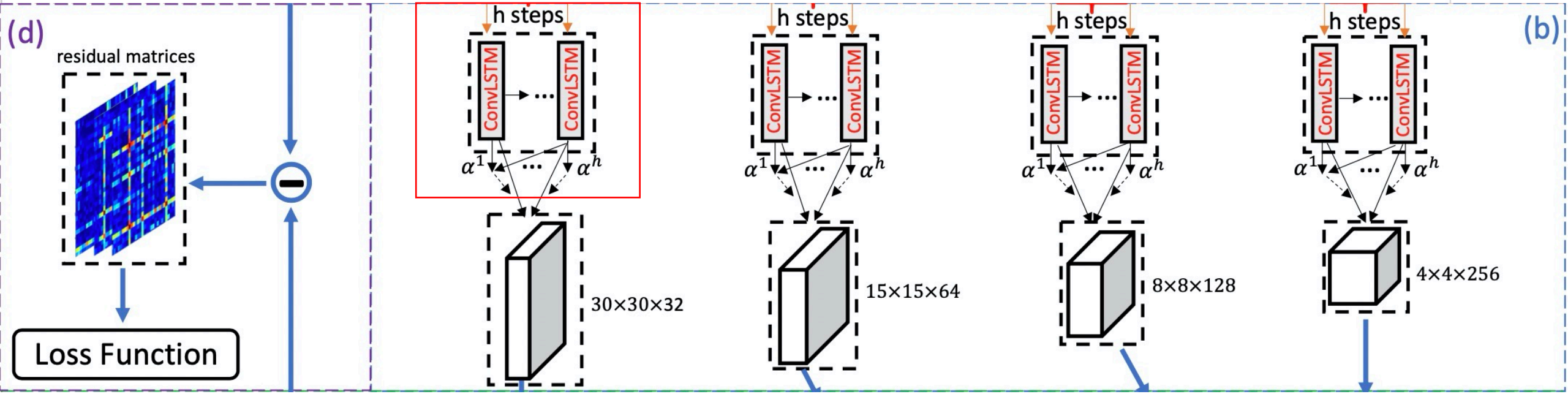




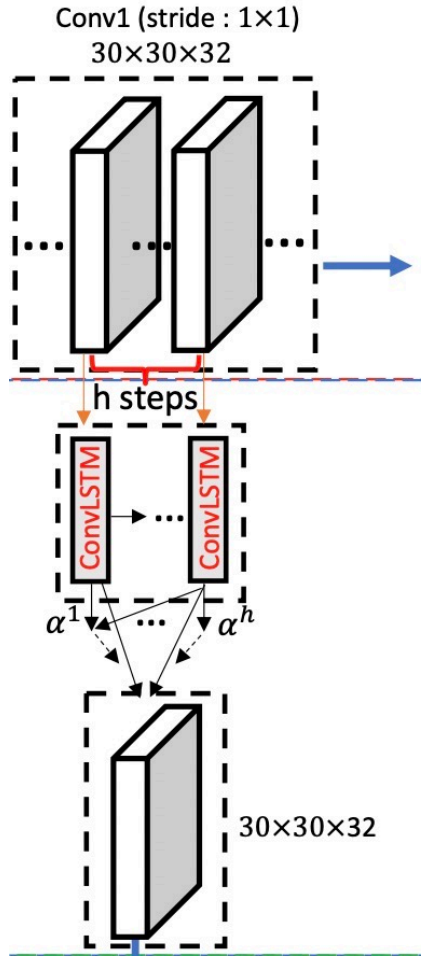


原始矩阵30*30*3

```python
class Conv_LSTM(nn.Module):
    def __init__(self):
        super(Conv_LSTM, self).__init__()
        self.conv1_lstm = ConvLSTM(input_channels=32, hidden_channels=[32],
                                   kernel_size=3, step=5, effective_step=[4])
        self.conv2_lstm = ConvLSTM(input_channels=64, hidden_channels=[64],
                                   kernel_size=3, step=5, effective_step=[4])
        self.conv3_lstm = ConvLSTM(input_channels=128, hidden_channels=[128],
                                   kernel_size=3, step=5, effective_step=[4])
        self.conv4_lstm = ConvLSTM(input_channels=256, hidden_channels=[256],
                                   kernel_size=3, step=5, effective_step=[4])
```



```python
class ConvLSTM(nn.Module):
    # input_channels corresponds to the first input feature map
    # hidden state is a list of succeeding lstm layers.
    def __init__(self, input_channels, hidden_channels, kernel_size, step=1, effective_step=[1]):
        super(ConvLSTM, self).__init__()
        self.input_channels = [input_channels] + hidden_channels
        self.hidden_channels = hidden_channels
        self.kernel_size = kernel_size
        self.num_layers = len(hidden_channels)
        self.step = step
        self.effective_step = effective_step
        self._all_layers = []
        for i in range(self.num_layers):
            name = 'cell{}'.format(i)
            cell = ConvLSTMCell(self.input_channels[i], self.hidden_channels[i], self.kernel_size)
            setattr(self, name, cell)
            self._all_layers.append(cell)
```



$$\mathbf{z}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XZ}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HZ}} * \mathcal{H}^{t-1,l} + \tilde{W}^{k}_{\mathcal{CZ}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^{l}_{\mathcal{Z}})$$

$$\mathbf{r}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XR}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HR}} * \mathcal{H}^{t-1,l} + \tilde{W}^{l}_{\mathcal{CR}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^{l}_{\mathcal{R}})$$

$$\mathcal{C}^{t,l} = \mathbf{z}^{t,l} \circ \tanh(\tilde{W}^{l}_{\mathcal{XC}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HC}} * \mathcal{H}^{t-1,l} + \tilde{b}^{l}_{\mathcal{C}}) + \mathbf{r}^{t,l} \circ \mathcal{C}^{t-1,l}$$

$$\mathbf{o}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XO}} * \mathcal{X}^{t,l} + \tilde{W}_{\mathcal{HO}} * \mathcal{H}^{t-1,l} + \tilde{W}_{\mathcal{CO}} \circ \mathcal{C}^{t,l} + \tilde{b}^{l}_{\mathcal{O}})$$

$$\mathcal{H}^{t,l} = \mathbf{o}^{t,l} \circ \tanh(\mathcal{C}^{t,l})$$

```python
class ConvLSTMCell(nn.Module):
    def __init__(self, input_channels, hidden_channels, kernel_size):
        super(ConvLSTMCell, self).__init__()
```

w^l_{XZ} x, input

```python
        self.Wxi = nn.Conv2d(self.input_channels, self.hidden_channels, self.kernel_size, 1, self.padding, bias=True)
```

w^l_{HZ}

```python
        self.Whi = nn.Conv2d(self.hidden_channels, self.hidden_channels, self.kernel_size, 1, self.padding, bias=False)
        self.Wxf = nn.Conv2d(self.input_channels, self.hidden_channels, self.kernel_size, 1, self.padding, bias=True)
        self.Whf = nn.Conv2d(self.hidden_channels, self.hidden_channels, self.kernel_size, 1, self.padding, bias=False)
        self.Wxc = nn.Conv2d(self.input_channels, self.hidden_channels, self.kernel_size, 1, self.padding, bias=True)
        self.Whc = nn.Conv2d(self.hidden_channels, self.hidden_channels, self.kernel_size, 1, self.padding, bias=False)
        self.Wxo = nn.Conv2d(self.input_channels, self.hidden_channels, self.kernel_size, 1, self.padding, bias=True)
        self.Who = nn.Conv2d(self.hidden_channels, self.hidden_channels, self.kernel_size, 1, self.padding, bias=False)
```

w^l_{CZ} cell, input

```python
        self.Wci = None
        self.Wcf = None
        self.Wco = None
```

input: x, h, c

```python
    def forward(self, x, h, c):
        ci = torch.sigmoid(self.Wxi(x) + self.Whi(h) + c * self.Wci)
        cf = torch.sigmoid(self.Wxf(x) + self.Whf(h) + c * self.Wcf)
        cc = cf * c + ci * torch.tanh(self.Wxc(x) + self.Whc(h))
        co = torch.sigmoid(self.Wxo(x) + self.Who(h) + cc * self.Wco)
        ch = co * torch.tanh(cc)
        return ch, cc
```

input $\mathbf{z}^{t,l} = \sigma(\tilde{W}^l_{\mathcal{XZ}} * \mathcal{X}^{t,l} + \tilde{W}^l_{\mathcal{HZ}} * \mathcal{H}^{t-1,l} + \tilde{W}^k_{\mathcal{CZ}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^l_{\mathcal{Z}})$

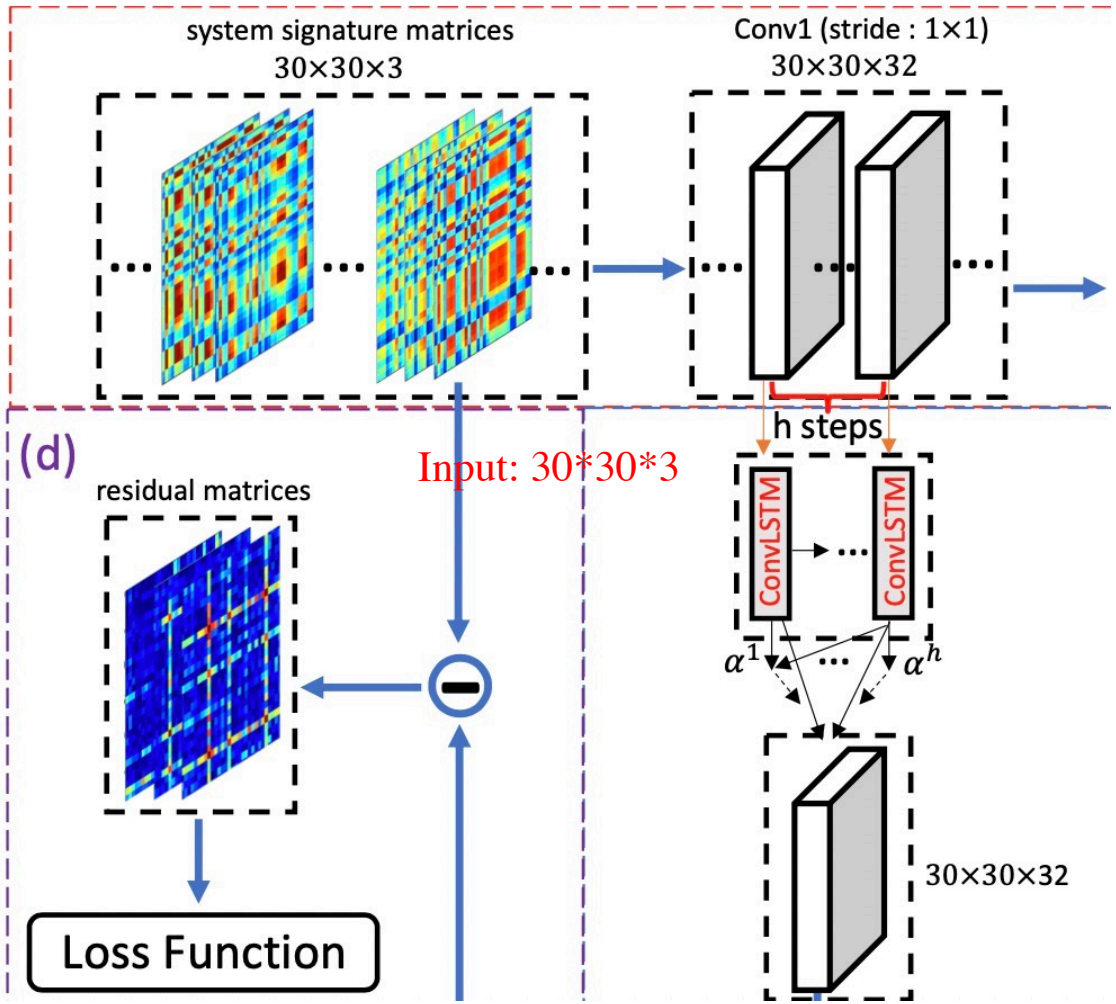
forget $\mathbf{r}^{t,l} = \sigma(\tilde{W}^l_{\mathcal{XR}} * \mathcal{X}^{t,l} + \tilde{W}^l_{\mathcal{HR}} * \mathcal{H}^{t-1,l} + \tilde{W}^l_{\mathcal{CR}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^l_{\mathcal{R}})$

cell $\mathcal{C}^{t,l} = \mathbf{z}^{t,l} \circ \tanh(\tilde{W}^l_{\mathcal{XC}} * \mathcal{X}^{t,l} + \tilde{W}^l_{\mathcal{HC}} * \mathcal{H}^{t-1,l} + \tilde{b}^l_{\mathcal{C}}) + \mathbf{r}^{t,l} \circ \mathcal{C}^{t-1,l}$

output $\mathbf{o}^{t,l} = \sigma(\tilde{W}^l_{\mathcal{XO}} * \mathcal{X}^{t,l} + \tilde{W}_{\mathcal{HO}} * \mathcal{H}^{t-1,l} + \tilde{W}_{\mathcal{CO}} \circ \mathcal{C}^{t,l} + \tilde{b}^l_{\mathcal{O}})$

$\mathcal{H}^{t,l} = \mathbf{o}^{t,l} \circ \tanh(\mathcal{C}^{t,l})$

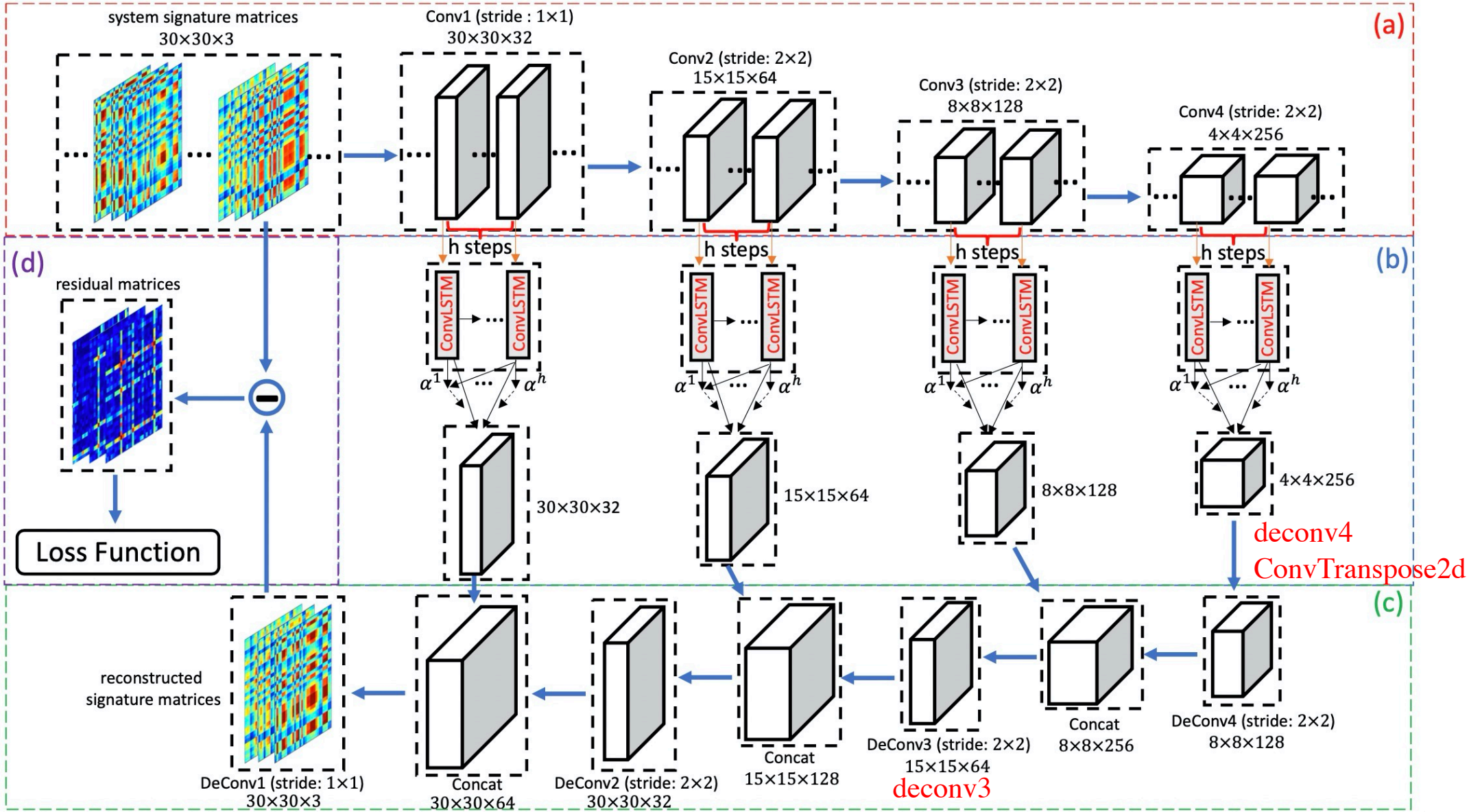
ConvLSTM



LSTM的参数是由Conv得到的

$$\mathbf{z}^{t,l} = \sigma(\tilde{W}^l_{\mathcal{XZ}} * \mathcal{X}^{t,l} + \tilde{W}^l_{\mathcal{HZ}} * \mathcal{H}^{t-1,l} + \tilde{W}^k_{\mathcal{CZ}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^l_{\mathcal{Z}})$$
$$\mathbf{r}^{t,l} = \sigma(\tilde{W}^l_{\mathcal{XR}} * \mathcal{X}^{t,l} + \tilde{W}^l_{\mathcal{HR}} * \mathcal{H}^{t-1,l} + \tilde{W}^l_{\mathcal{CR}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^l_{\mathcal{R}})$$
$$\mathcal{C}^{t,l} = \mathbf{z}^{t,l} \circ \tanh(\tilde{W}^l_{\mathcal{XC}} * \mathcal{X}^{t,l} + \tilde{W}^l_{\mathcal{HC}} * \mathcal{H}^{t-1,l} + \tilde{b}^l_{\mathcal{C}}) + \mathbf{r}^{t,l} \circ \mathcal{C}^{t-1,l}$$
$$\mathbf{o}^{t,l} = \sigma(\tilde{W}^l_{\mathcal{XO}} * \mathcal{X}^{t,l} + \tilde{W}_{\mathcal{HO}} * \mathcal{H}^{t-1,l} + \tilde{W}_{\mathcal{CO}} \circ \mathcal{C}^{t,l} + \tilde{b}^l_{\mathcal{O}})$$
$$\mathcal{H}^{t,l} = \mathbf{o}^{t,l} \circ \tanh(\mathcal{C}^{t,l})$$

ture maps) across different time steps. Specifically, given the feature maps $\mathcal{X}^{t,l}$ from the l-th convolutional layer and previous hidden state $\mathcal{H}^{t-1,l} \in \mathbb{R}^{n_l \times n_l \times d_l}$, the current hidden state $\mathcal{H}^{t,l}$ is updated with $\mathcal{H}^{t,l} = \text{ConvLSTM}(\mathcal{X}^{t,l}, \mathcal{H}^{t-1,l})$,



kernel_size dilation

nn.ConvTranspose2d(in_channels, 128, 2, 2, 0, 0)

in_channel out_channel stride padding

forward

```
self.Wci = Variable(torch.zeros(1, hidden, shape[0], shape[1])).to("cpu")  # 定义为全0变量; [1, 32, 30, 30]
self.Wcf = Variable(torch.zeros(1, hidden, shape[0], shape[1])).to("cpu")
self.Wco = Variable(torch.zeros(1, hidden, shape[0], shape[1])).to("cpu")
```

```
return (Variable(torch.zeros(batch_size, hidden, shape[0], shape[1])).to("cpu"),  # [5, 32, 30, 30]
        Variable(torch.zeros(batch_size, hidden, shape[0], shape[1])).to("cpu"))  # [5, 32, 30, 30]
```

h

初始h和c

c

```
def forward(self, x, h, c):  # x: 上层Cnn输出的conv1  self:    x: t
    ci = torch.sigmoid(self.Wxi(x) + self.Whi(h) + c * self.Wci)
    cf = torch.sigmoid(self.Wxf(x) + self.Whf(h) + c * self.Wcf)
    cc = cf * c + ci * torch.tanh(self.Wxc(x) + self.Whc(h))
    co = torch.sigmoid(self.Wxo(x) + self.Who(h) + cc * self.Wco)
    ch = co * torch.tanh(cc)
    return ch, cc
```

$[5, 32, 30, 30]$ ->

```
In[39]: self.Wxi
Out[39]: Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

-> $[5, 32, 30, 30]$

$$\mathbf{z}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XZ}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HZ}} * \mathcal{H}^{t-1,l} + \tilde{W}^{k}_{\mathcal{CZ}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^{l}_{\mathcal{Z}})$$

$$\mathbf{r}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XR}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HR}} * \mathcal{H}^{t-1,l} + \tilde{W}^{l}_{\mathcal{CR}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^{l}_{\mathcal{R}})$$

$$\mathcal{C}^{t,l} = \mathbf{z}^{t,l} \circ \tanh(\tilde{W}^{l}_{\mathcal{XC}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HC}} * \mathcal{H}^{t-1,l} + \tilde{b}^{l}_{\mathcal{C}}) + \mathbf{r}^{t,l} \circ \mathcal{C}^{t-1,l}$$

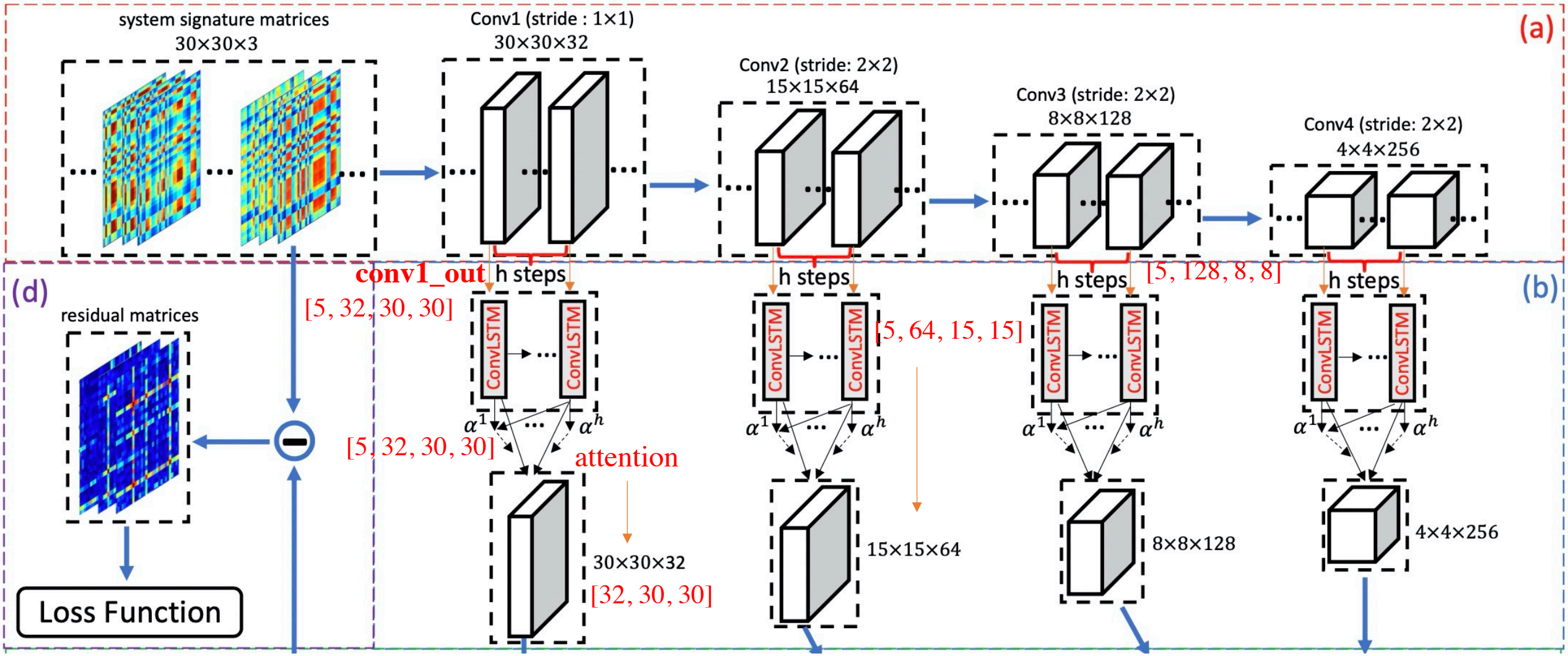
$$\mathbf{o}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XO}} * \mathcal{X}^{t,l} + \tilde{W}_{\mathcal{HO}} * \mathcal{H}^{t-1,l} + \tilde{W}_{\mathcal{CO}} \circ \mathcal{C}^{t,l} + \tilde{b}^{l}_{\mathcal{O}})$$

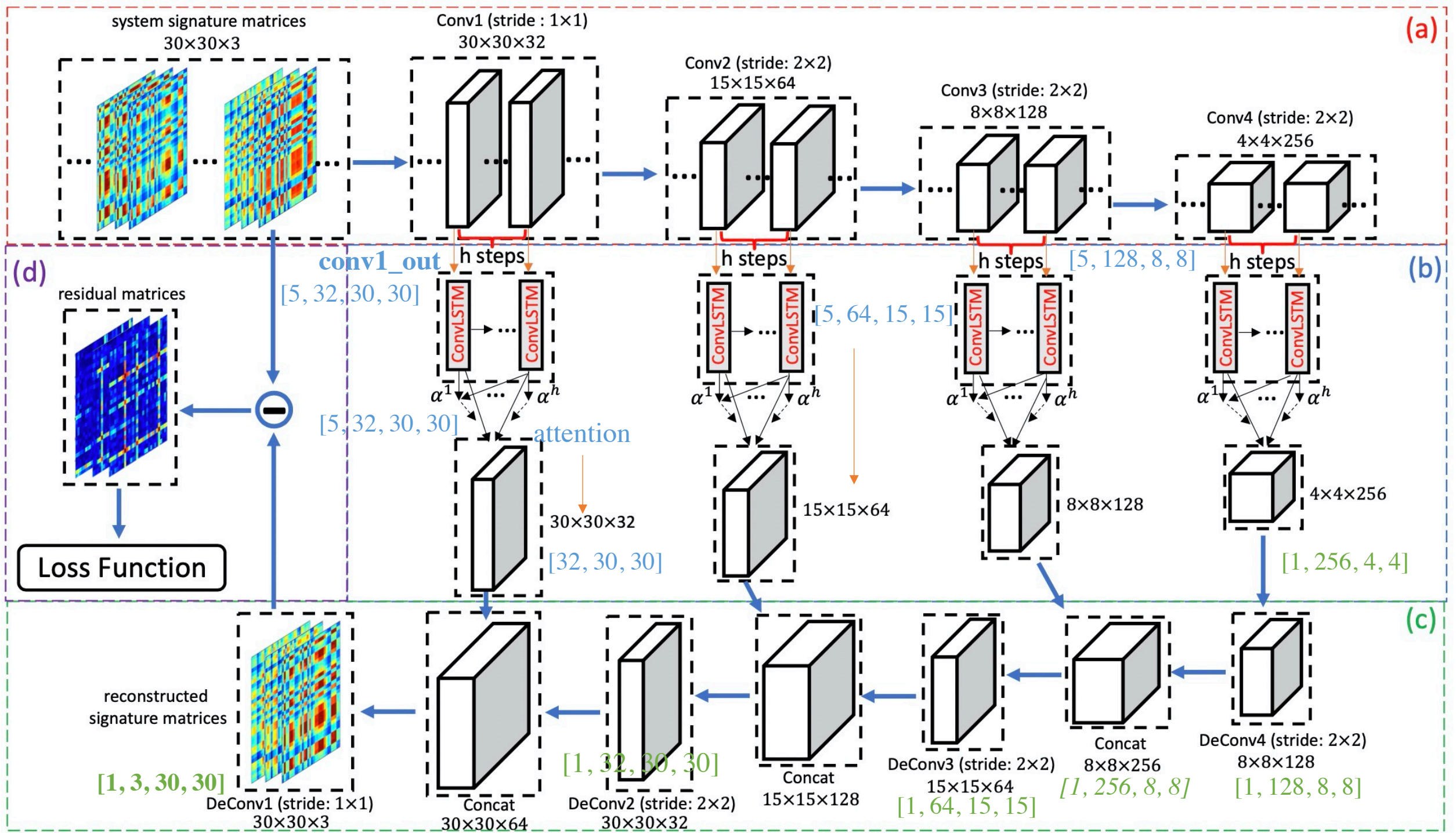
$$\mathcal{H}^{t,l} = \mathbf{o}^{t,l} \circ \tanh(\mathcal{C}^{t,l})$$

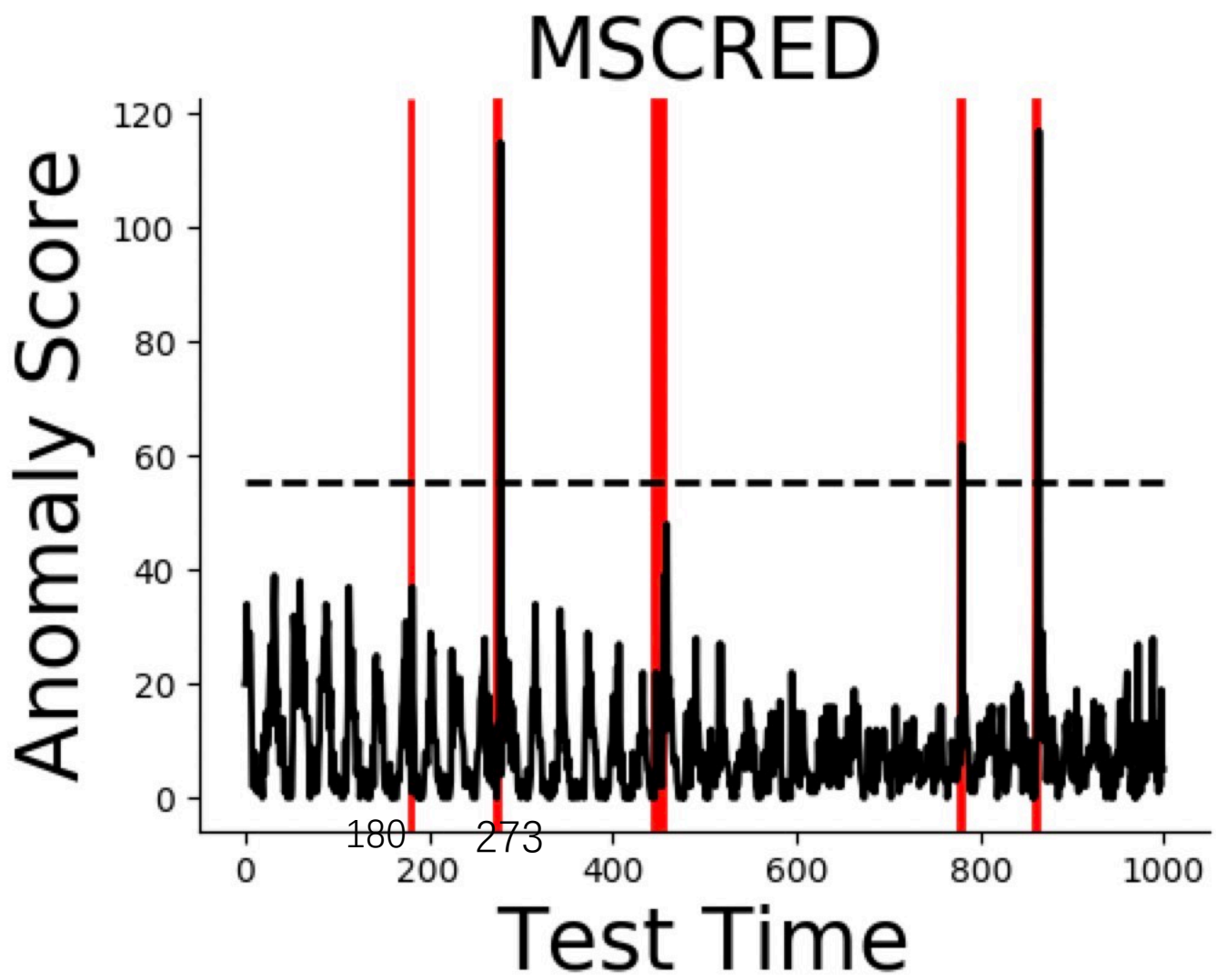
```
for k in range(5):  k: 0
    attention_w.append(torch.sum(torch.mul(ConvLstm_out[k], ConvLstm_out[-1]))/5)
```

lstm最后一个时间步

$$\hat{\mathcal{H}}^{t,l} = \sum_{i \in (t-h,t)} \alpha^i \mathcal{H}^{i,l}, \alpha^i = \frac{\exp\{\frac{\mathrm{Vec}(\mathcal{H}^{t,l})^{\mathrm{T}} \mathrm{Vec}(\mathcal{H}^{i,l})}{\chi}\}}{\sum_{i \in (t-h,t)} \exp\{\frac{\mathrm{Vec}(\mathcal{H}^{t,l})^{\mathrm{T}} \mathrm{Vec}(\mathcal{H}^{i,l})}{\chi}\}} \tag{4}$$


(a)
system signature matrices
30×30×3
Conv1 (stride : 1×1)
30×30×32
Conv2 (stride: 2×2)
15×15×64
Conv3 (stride: 2×2)
8×8×128
Conv4 (stride: 2×2)
4×4×256
conv1_out
h steps
[5, 32, 30, 30]
[5, 64, 15, 15]
[5, 128, 8, 8]
(b)
(d)
residual matrices
ConvLSTM
α^1
α^h
[5, 32, 30, 30]
attention
30×30×32
[32, 30, 30]
15×15×64
8×8×128
4×4×256
Loss Function

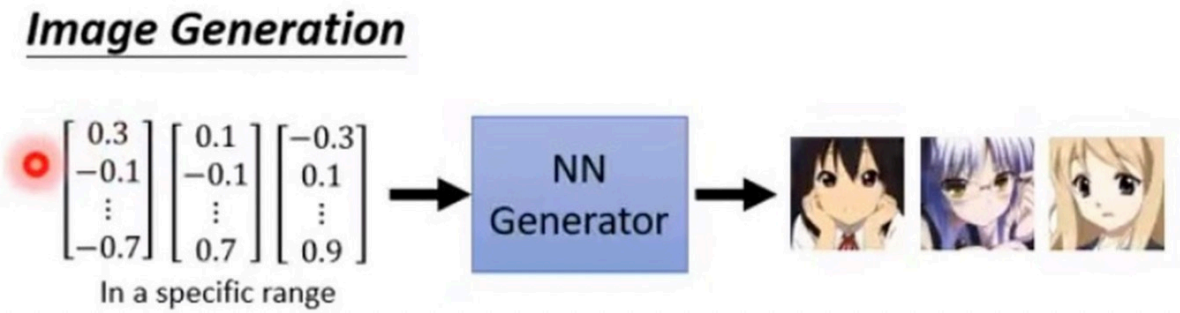

(a)
system signature matrices
30×30×3
Conv1 (stride : 1×1)
30×30×32
Conv2 (stride: 2×2)
15×15×64
Conv3 (stride: 2×2)
8×8×128
Conv4 (stride: 2×2)
4×4×256
(b)
conv1_out
[5, 32, 30, 30]
h steps
ConvLSTM
attention
[5, 32, 30, 30]
30×30×32
[32, 30, 30]
[5, 64, 15, 15]
15×15×64
[5, 128, 8, 8]
8×8×128
4×4×256
[1, 256, 4, 4]
(c)
DeConv4 (stride: 2×2)
8×8×128
[1, 128, 8, 8]
Concat
8×8×256
[1, 256, 8, 8]
DeConv3 (stride: 2×2)
15×15×64
[1, 64, 15, 15]
Concat
15×15×128
DeConv2 (stride: 2×2)
30×30×32
[1, 32, 30, 30]
Concat
30×30×64
DeConv1 (stride: 1×1)
30×30×3
[1, 3, 30, 30]
reconstructed
signature matrices
(d)
residual matrices
Loss Function


MSCRED
Anomaly Score
120
100
80
60
40
20
0
180
273
0
200
400
600
800
1000
Test Time

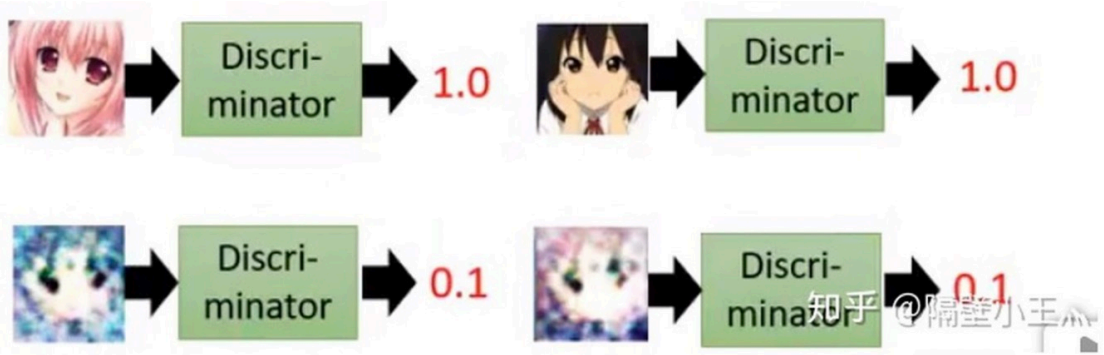
TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks

之前我们讲的autoEncoder方式进行异常检测，这里作者认为autoEncoder的方式可能会存在过拟合现象，采用GAN的方式去生成序列，作者认为可能会带来更好的效果。
其次，在评估异常时，鉴别器和重构误差可以配合来进行异常识别，识别效果会有所提升。
所以提出一种新颖的方式去进行异常检测。

生成器：



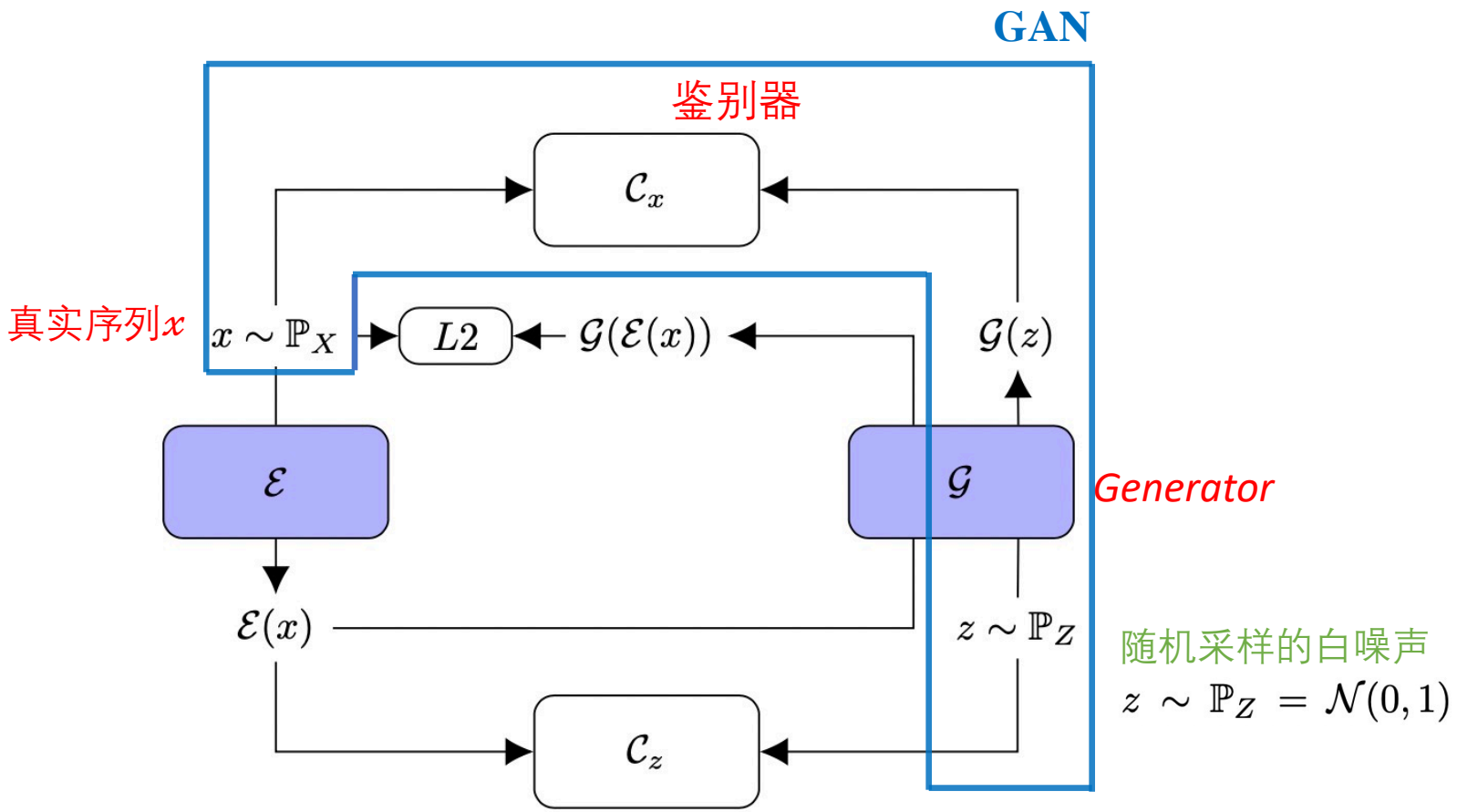
鉴别器：

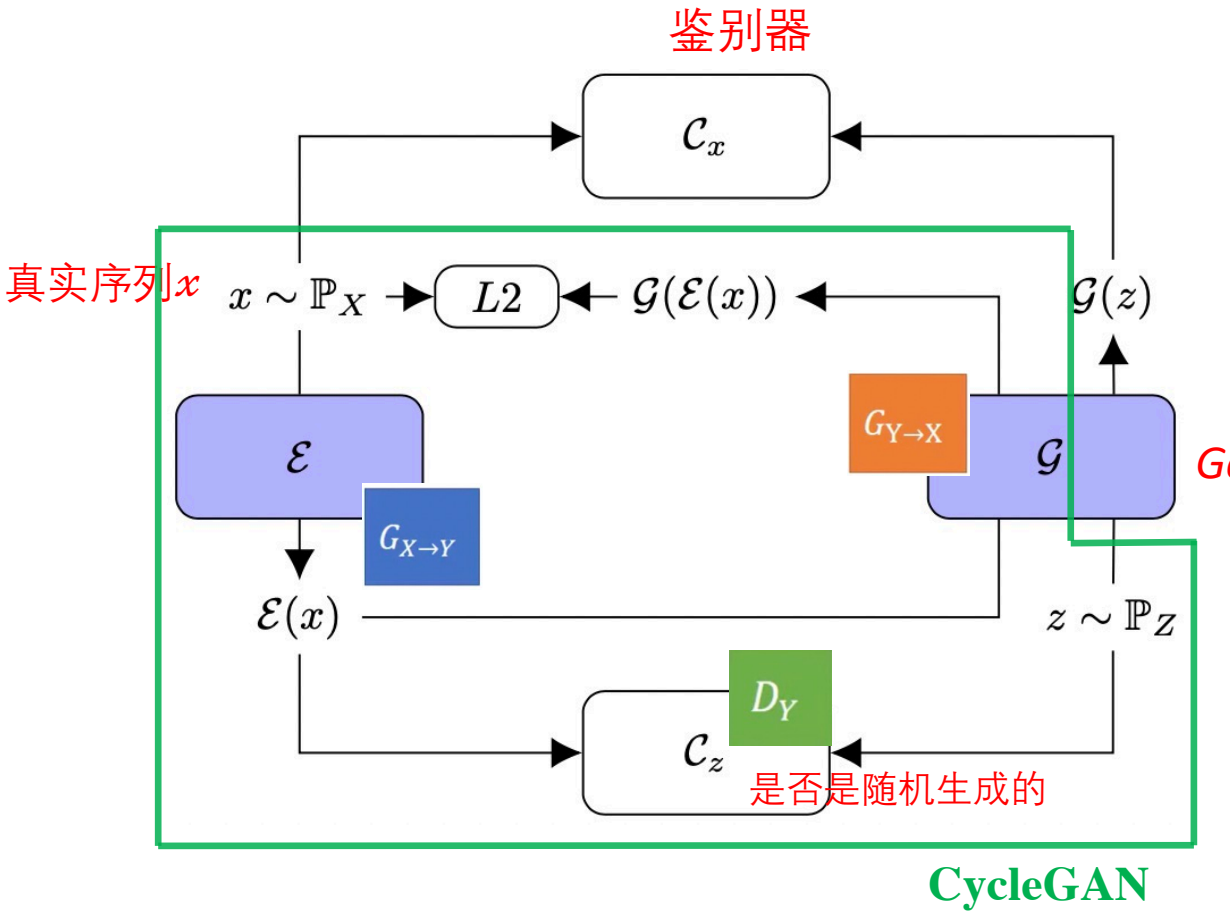


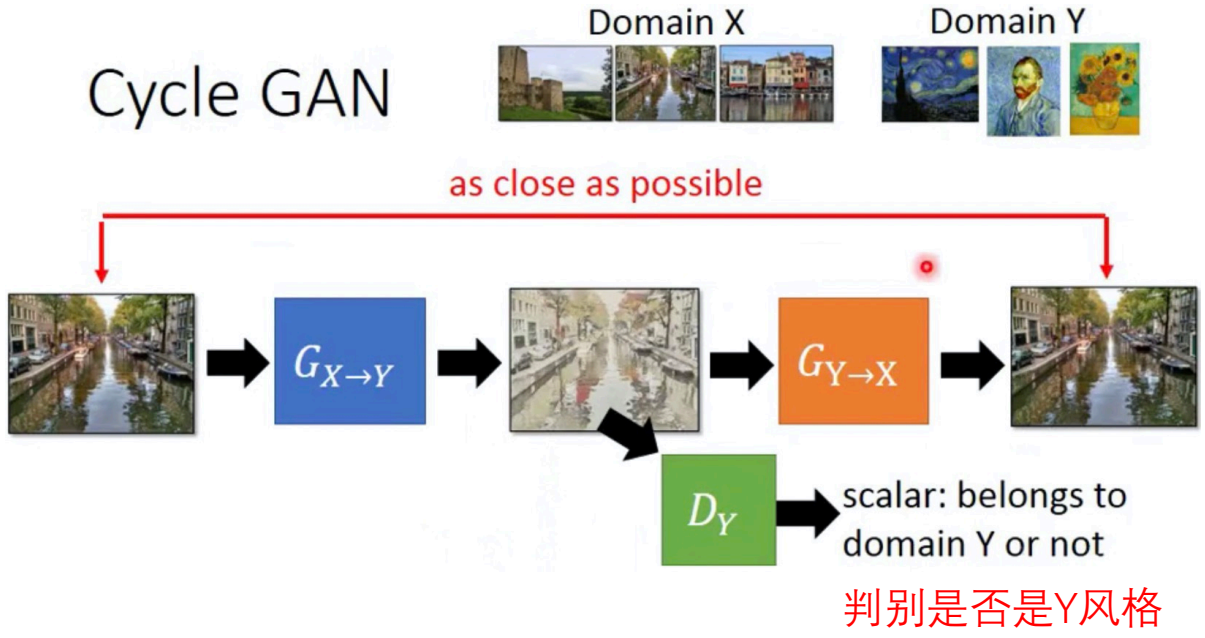
GAN：https://www.bilibili.com/video/BV1Up411R7Lk?p=1

IV. Adversarial Learning for Time Series Reconstruction

$\mathcal{E} : X \rightarrow Z$ and $\mathcal{G} : Z \rightarrow X$

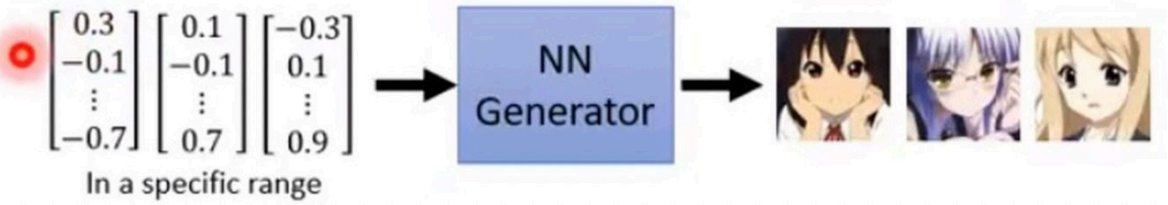




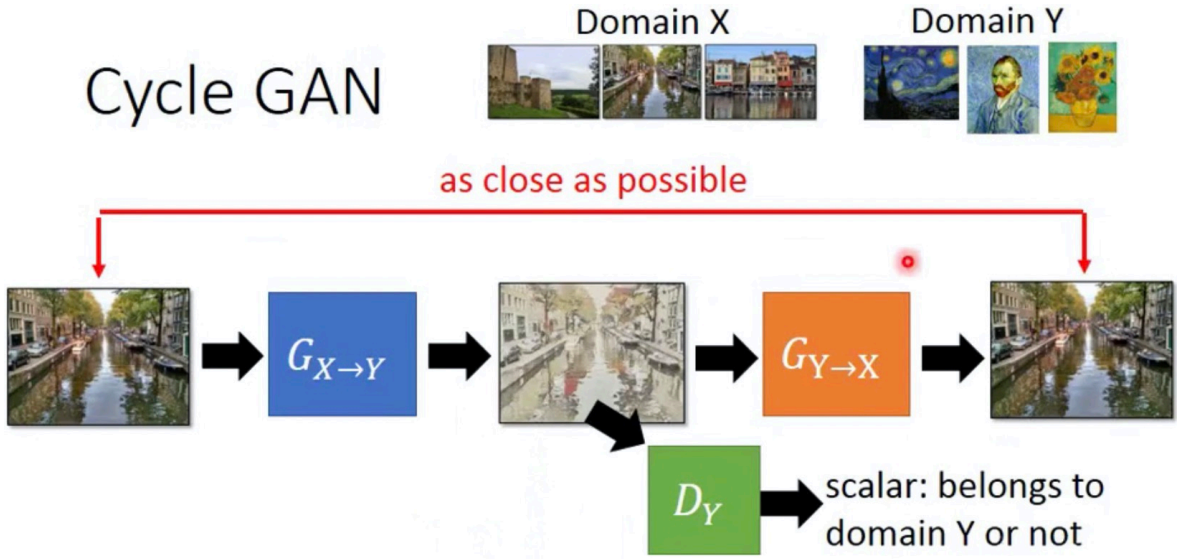


CycleGAN：https://www.bilibili.com/video/BV1FW411b7ux?from=search&seid=9703283865686077945&spm_id_from=333.337.0.0

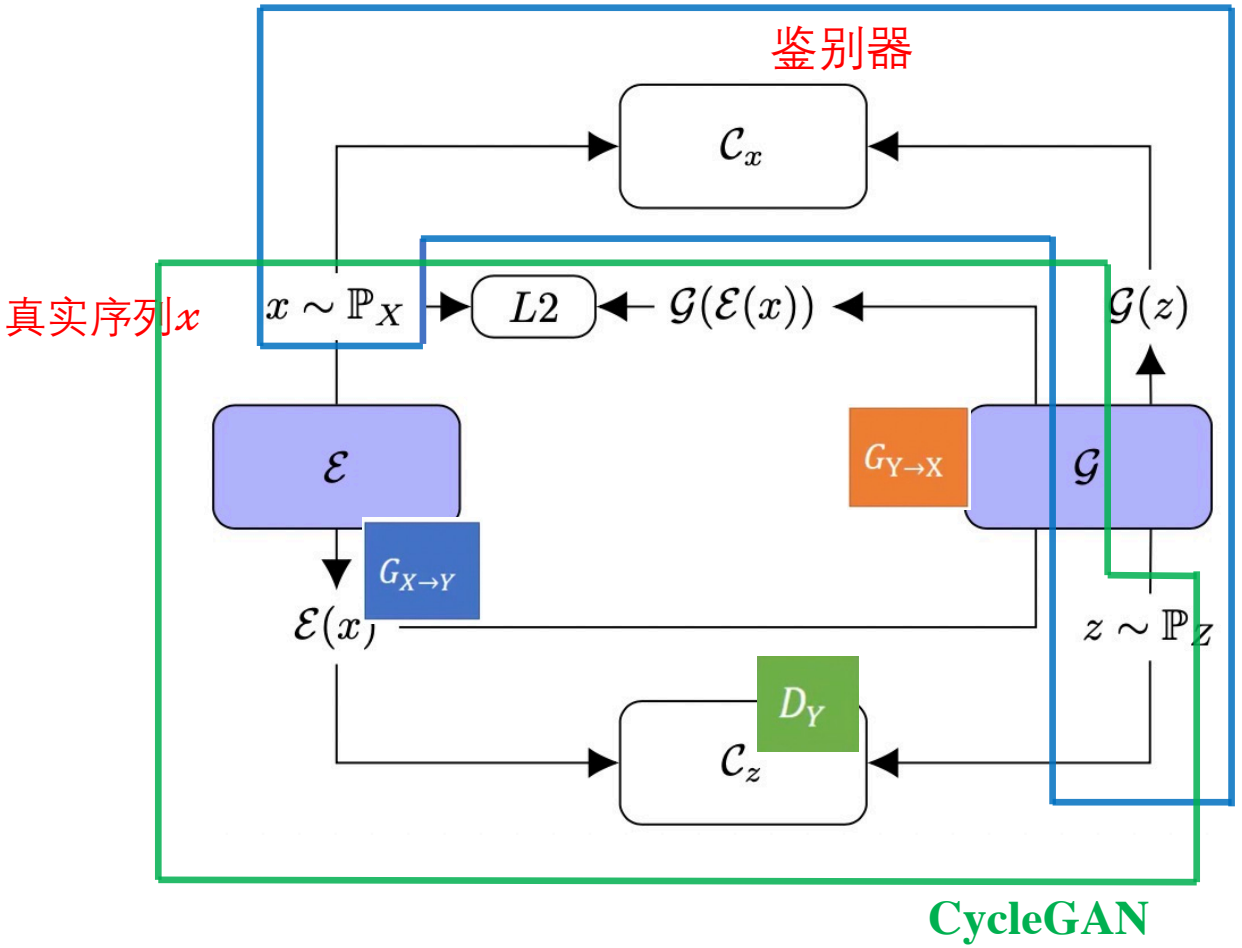
Image Generation



Cycle GAN



GAN



Generator

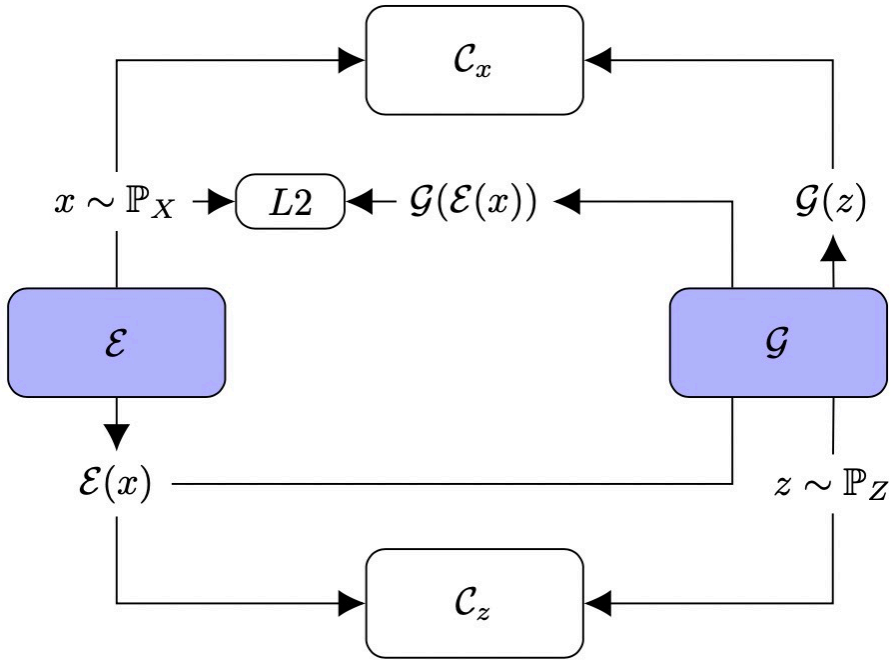
随机采样的白噪声

$z \sim \mathbb{P}_Z = \mathcal{N}(0,1)$

训练方法

真实:1　　随机:0

鉴别真实 或者 生成的



鉴别随机 或者 正常样本生成的

随机:1　　正常:0

x_i: 正常样本

z_i: 随机样本

$gp(x_i, \mathcal{G}(z_i))$: gradient penalty

other words, $\mathcal{G}$ is trying to fool $\mathcal{C}_x$ by generating real-looking sequences. Thus, our high-level objective consists of two

鉴别真实 或者 生成的

The purpose of the second *Critic* C_z is to distinguish between random latent samples $z \sim \mathbb{P}_Z$ and encoded samples $\mathcal{E}(x)$ with $x \sim \mathbb{P}_X$. We present the model type and architecture for

鉴别随机 或者 正常样本生成的

1. 固定生成器，训练鉴别器

cost function越大越好

$$g_{w_{\mathcal{C}_x}} = \nabla_{w_{\mathcal{C}_x}} [\frac{1}{m} \sum_{i=1}^{m} \mathcal{C}_x(x_i) - \frac{1}{m} \sum_{i=1}^{m} \mathcal{C}_x(\mathcal{G}(z_i)) + gp(x_i, \mathcal{G}(z_i))]$$

$$w_{\mathcal{C}_x} = w_{\mathcal{C}_x} + \eta \cdot \mathrm{adam}(w_{\mathcal{C}_x}, g_{w_{\mathcal{C}_x}})$$　梯度上升法

鉴别随机:1

正常样本生成:0

$$g_{w_{\mathcal{C}_z}} = \nabla_{w_{\mathcal{C}_z}} [\frac{1}{m} \sum_{i=1}^{m} \mathcal{C}_z(z_i) - \frac{1}{m} \sum_{i=1}^{m} \mathcal{C}_z(\mathcal{E}(x_i)) + gp(z_i, \mathcal{E}(x_i))]$$

$$w_{\mathcal{C}_z} = w_{\mathcal{C}_z} + \eta \cdot \mathrm{adam}(w_{\mathcal{C}_z}, g_{w_{\mathcal{C}_z}})$$

2. 固定鉴别器，训练生成器

损失函数，越小越好

$$g_{w_{\mathcal{G},\mathcal{E}}} = \nabla_{w_{\mathcal{G}}, w_{\mathcal{E}}} [\frac{1}{m} \sum_{i=1}^{m} \mathcal{C}_x(x_i) - \frac{1}{m} \sum_{i=1}^{m} \mathcal{C}_x(\mathcal{G}(z_i)) + \frac{1}{m} \sum_{i=1}^{m} \mathcal{C}_z(z_i) - \frac{1}{m} \sum_{i=1}^{m} \mathcal{C}_z(\mathcal{E}(x_i)) + \frac{1}{m} \sum_{i=1}^{m} \|x_i - \mathcal{G}(\mathcal{E}(x_i))\|_2]$$

$$w_{\mathcal{G},\mathcal{E}}, = w_{\mathcal{G},\mathcal{E}} - \eta \cdot \mathrm{adam}(w_{\mathcal{G},\mathcal{E}}, g_{w_{\mathcal{G},\mathcal{E}}})$$

V. TIME-SERIES GAN FOR ANOMALY DETECTION (TADGAN)

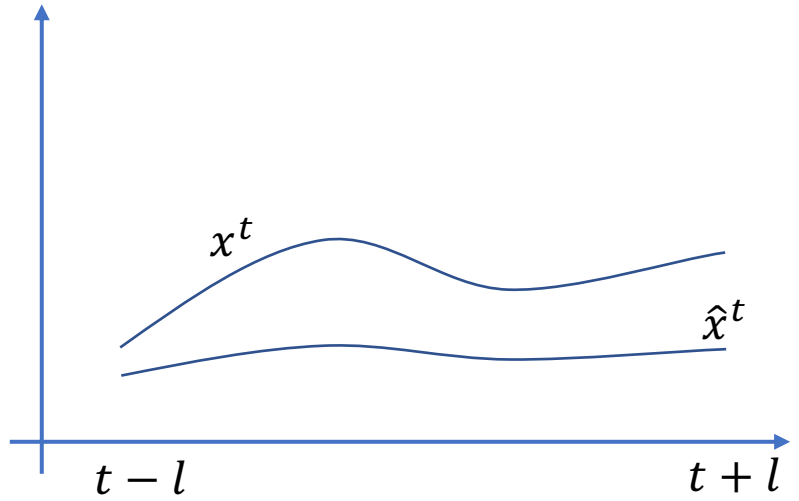
训练完模型后，我们得到两种异常评判方法：
重构误差 和 鉴别器C_x

A. *Estimating Anomaly Scores using Reconstruction Errors*

评估结构误差的三种方式

$$s_t = \left| x^t - \hat{x}^t \right| \tag{7}$$

计算重构误差



$$s_t = \frac{1}{2*l} \left| \int_{t-l}^{t+l} x^t - \hat{x}^t \, dx \right| \tag{8}$$

它被定义为两条长度为l的曲线下面积之间的平均差

区域差异善于识别长期存在小差异的区域

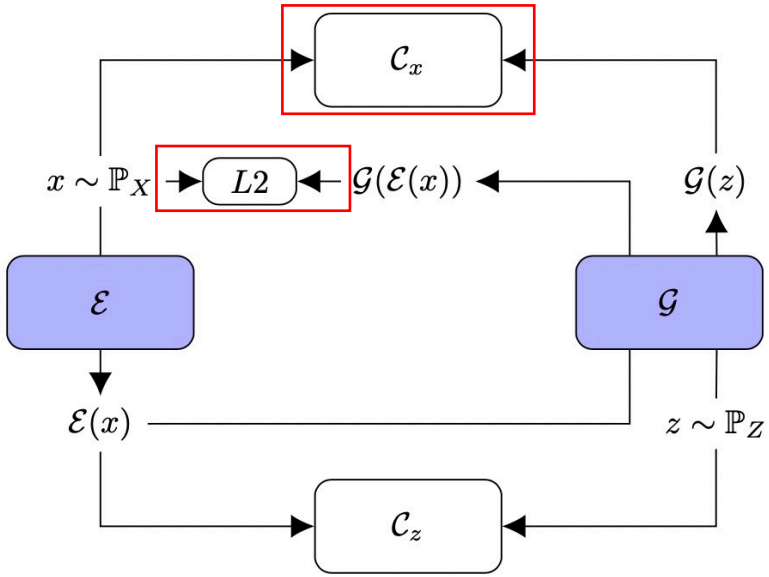
$$s_t = W^* = \text{DTW}(X, \hat{X}) = \min_W \left[\frac{1}{K} \sqrt{\sum_{k=1}^{K} w_k} \right] \tag{9}$$

动态时间规整

DTW能够在很长一段时间内识别出微小差异的区域，也可以处理时间转换问题

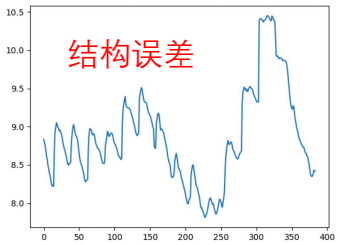
DTW: https://www.bilibili.com/video/BV1K5411H7EQ?p=6 4:10

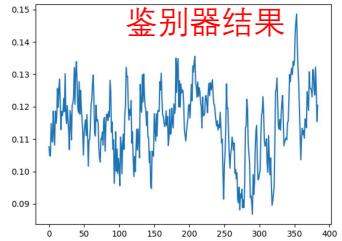
B. Estimating Anomaly Scores with Critic Outputs

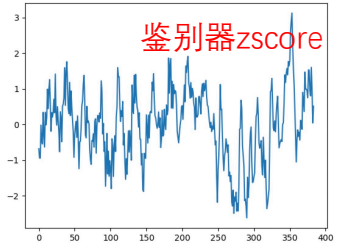


根据GAN模型，我们可以通过两个方面去评估异常性：

1. 重构误差，值越大表示越异常
2. 鉴别器C_x，越小表示越异常







C. Combining Both Scores

$Z_{RE}(x)$: *reconstruction errors*

$Z_{C_x}(x)$: *critic output*

计算各自的Z-score得分，大的Z-score指示着异常

$$\boldsymbol{a}(x) = \alpha Z_{RE}(x) + (1-\alpha) Z_{C_x}(x) \quad (10)$$

alpha = 0.5

$$\boldsymbol{a}(x) = \alpha Z_{RE}(x) \odot Z_{C_x}(x) \quad (11)$$

alpha = 1

D. *Identifying Anomalous Sequences*

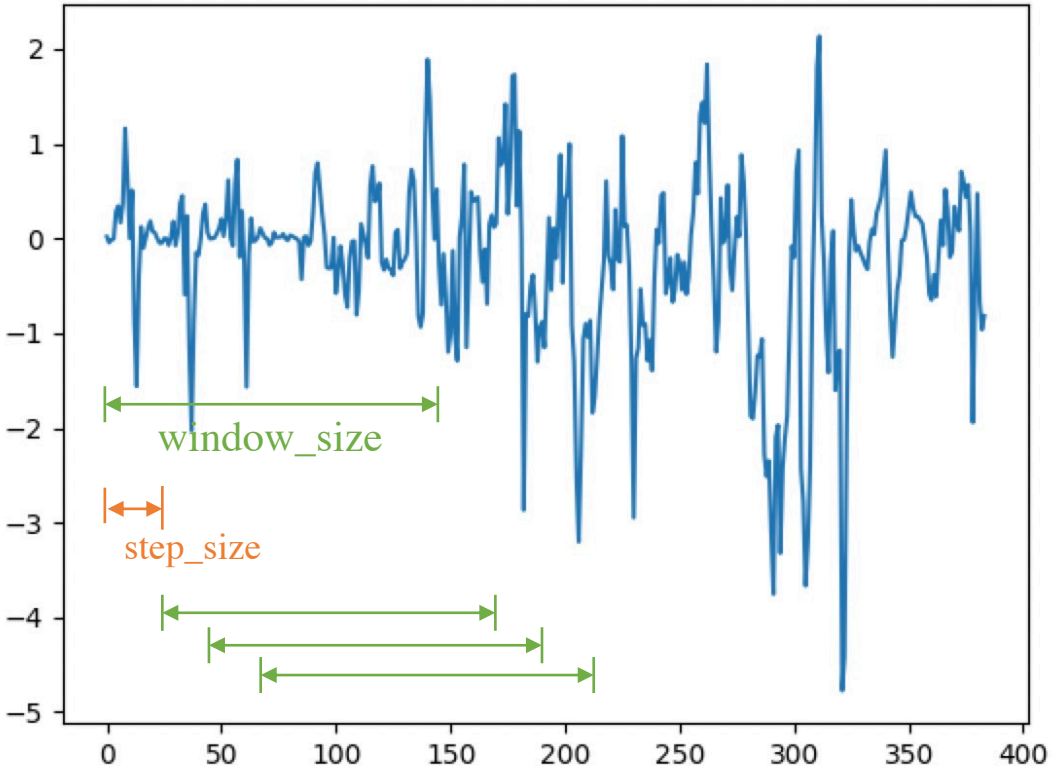
采用滑动窗口的方式来评估异常值，以增加对于异常的召回率。

$$window\ size = \frac{T}{3}$$

滑动窗口大小

$$step\ size = \frac{T}{3 * 10}$$

时间步



在*window size*时间窗口中计算均值和方差

Mitigating false positives

滑动窗口，会增加对于异常的召回，但是也会增加误报。作者采用了一种误判召回的方法，降低误判。

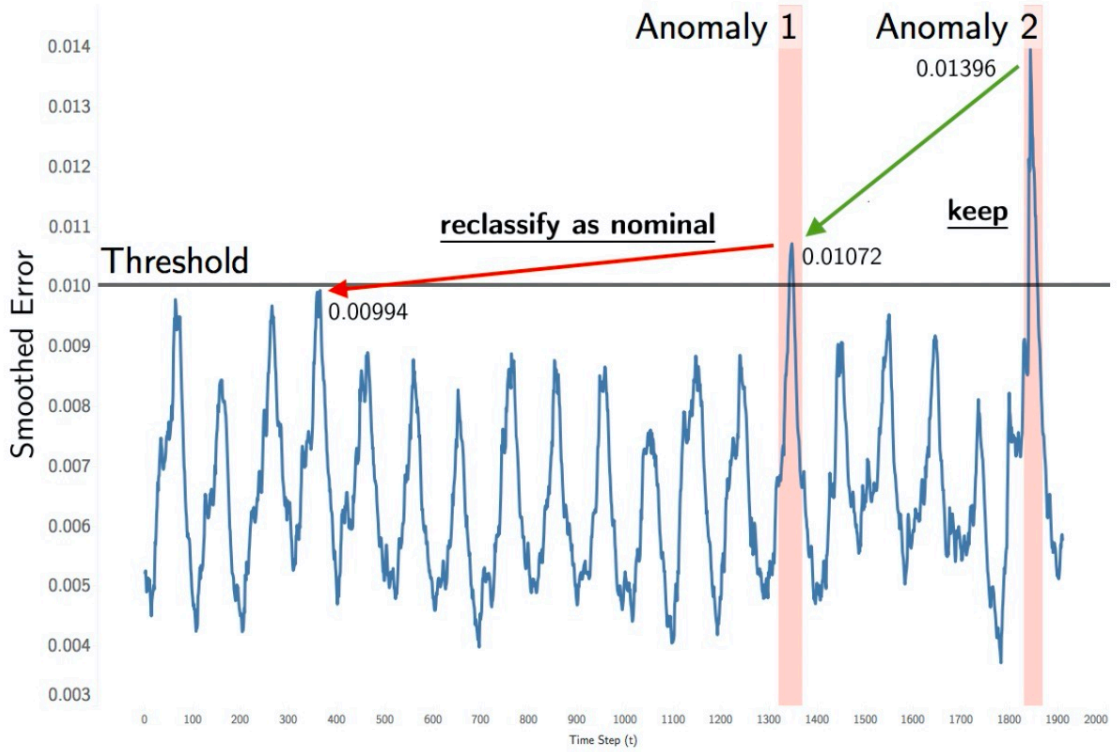


Figure 2: An example demonstrating the anomaly pruning process. In this scenario $\mathbf{e}_{max} = [0.01396, 0.01072, 0.00994]$ and the minimum percent decrease $p = 0.1$. The decrease from Anomaly 2 to Anomaly 1 $d^{(1)} = 0.23 > p$ and this sequence retains its classification as anomalous. From Anomaly 1 to the next highest smoothed error ($e_s = 0.0099$) $d^{(2)} = .07 < p$ so this sequence is re-classified as nominal.

$$p^i = (\boldsymbol{a}_{max}^{i-1} - \boldsymbol{a}_{max}^{i}) / \boldsymbol{a}_{max}^{i-1}$$

我们定义超过一定阈值为异常(0.1)，不超过认为是正常的。

$$\alpha = \{0.01396, 0.01072, 0.00994\}$$

正常值中最大值

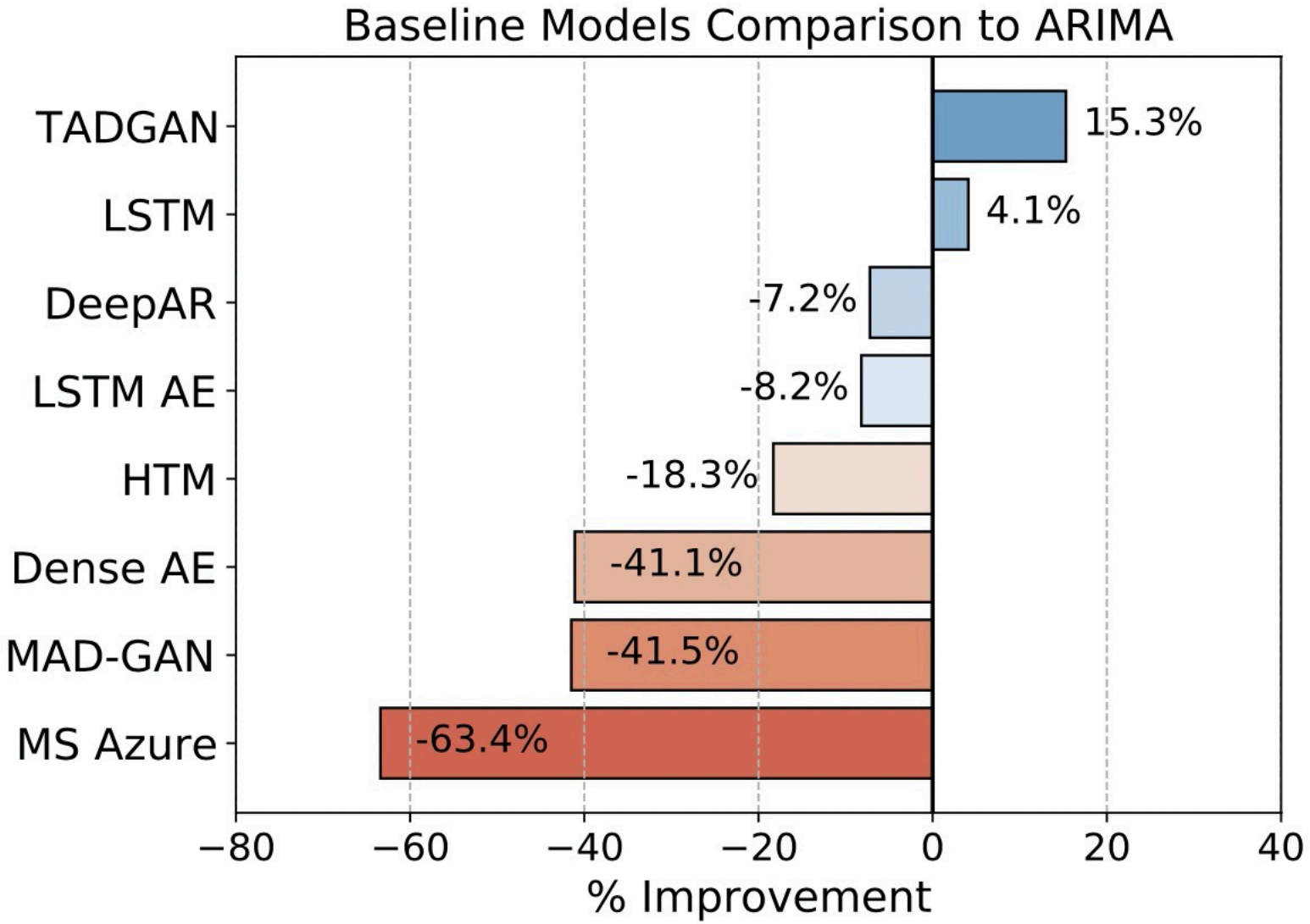
$$p^1 = \frac{0.01396 - 0.01072}{0.01396} = 0.23$$

Anomaly2

$$p^2 = \frac{0.01072 - 0.00994}{0.01072} = 0.07$$

Anomaly1

	NASA		Yahoo S5				NAB					
Baseline	MSL	SMAP	A1	A2	A3	A4	Art	AdEx	AWS	Traf	Tweets	Mean±SD
TadGAN	**0.623**	**0.704**	**0.8**	0.867	0.685	0.6	**0.8**	**0.8**	0.644	0.486	**0.609**	**0.700±0.123**
(P) LSTM	0.46	0.69	0.744	**0.98**	0.772	0.645	0.375	0.538	0.474	**0.634**	0.543	0.623±0.163
(P) Arima	0.492	0.42	0.726	0.836	**0.815**	**0.703**	0.353	0.583	0.518	0.571	0.567	0.599±0.148
(C) DeepAR	0.583	0.453	0.532	0.929	0.467	0.454	0.545	0.615	0.39	0.6	0.542	0.555±0.130
(R) LSTM AE	0.507	0.672	0.608	0.871	0.248	0.163	0.545	0.571	**0.764**	0.552	0.542	0.549±0.193
(P) HTM	0.412	0.557	0.588	0.662	0.325	0.287	0.455	0.519	0.571	0.474	0.526	0.489±0.108
(R) Dense AE	0.507	0.7	0.472	0.294	0.074	0.09	0.444	0.267	0.64	0.333	0.057	0.353±0.212
(R) MAD-GAN	0.111	0.128	0.37	0.439	0.589	0.464	0.324	0.297	0.273	0.412	0.444	0.35±0.137
(C) MS Azure	0.218	0.118	0.352	0.612	0.257	0.204	0.125	0.066	0.173	0.166	0.118	0.219±0.145



和ARIMA的效果比较

采用了不同的误差评估和特征组合方法

	NASA		Yahoo S5				NAB					
Variation	MSL	SMAP	A1	A2	A3	A4	Art	AdEx	AWS	Traf	Tweets	Mean+SD
Critic	0.393	0.672	0.285	0.118	0.008	0.024	0.625	0	0.35	0.167	0.548	0.290±0.237
Point	0.585	0.588	0.674	0.758	0.628	**0.6**	0.588	0.611	0.551	0.383	0.571	0.594±0.086
Area	0.525	0.655	0.681	0.82	0.567	0.523	0.625	0.645	0.59	0.435	0.559	0.602±0.096
DTW	0.514	0.581	0.697	0.794	0.613	0.547	0.714	0.69	0.633	0.455	0.559	0.618±0.095
Critic×Point	0.619	0.675	0.703	0.75	**0.685**	0.536	0.588	0.579	0.576	0.4	0.59	0.609±0.091
Critic+Point	0.529	0.653	**0.8**	0.78	0.571	0.44	0.625	0.595	**0.644**	0.439	0.592	0.606±0.111
Critic×Area	0.578	**0.704**	0.719	**0.867**	0.587	0.46	**0.8**	0.6	0.6	0.4	0.571	0.625±0.131
Critic+Area	0.493	0.692	0.789	0.847	0.483	0.367	0.75	0.75	0.607	0.474	0.6	0.623±0.148
Critic×DTW	**0.623**	0.68	0.667	0.82	0.631	0.497	0.667	0.667	0.61	0.455	0.605	**0.629±0.091**
Critic+DTW	0.462	0.658	0.735	0.857	0.523	0.388	0.667	**0.8**	0.632	**0.486**	**0.609**	0.620±0.139
Mean	0.532	0.655	0.675	0.741	0.529	0.438	0.664	0.593	0.579	0.409	0.580	
SD	0.068	0.039	0.137	0.211	0.182	0.154	0.067	0.209	0.081	0.087	0.02	

不同的误差组合选择

Point $s_t = \left| x^t - \hat{x}^t \right|$

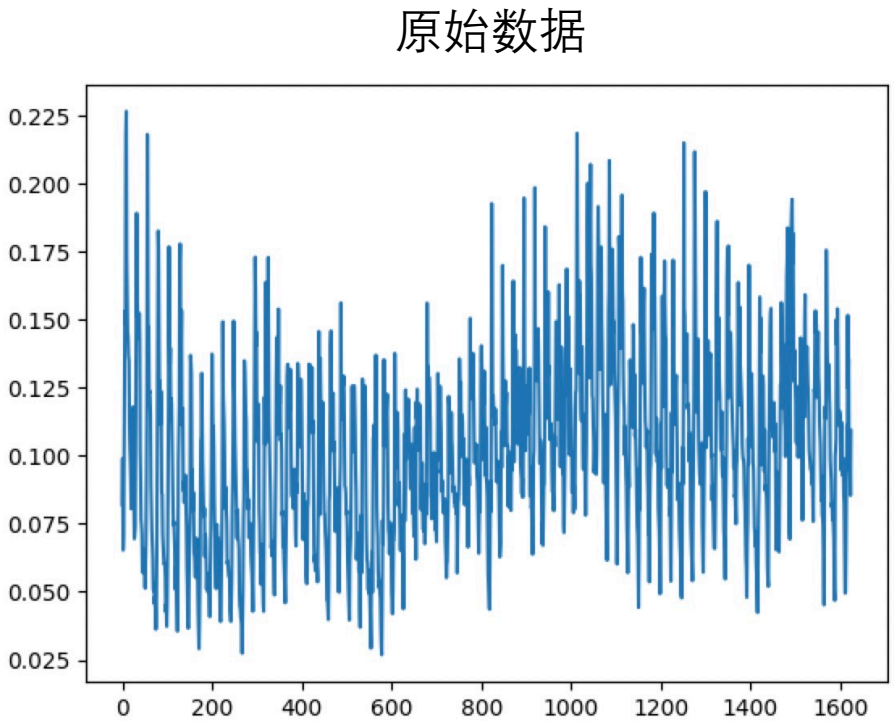
Area $s_t = \frac{1}{2 * l} \left| \int_{t-l}^{t+l} x^t - \hat{x}^t \, dx \right|$

DTW $s_t = W^* = \mathrm{DTW}(X, \hat{X}) = \min_W \left[\frac{1}{K} \sqrt{\sum_{k=1}^{K} w_k} \right]$

$$\boldsymbol{a}(x) = \alpha Z_{RE}(x) + (1 - \alpha) Z_{\mathcal{C}_x}(x) \qquad (10)$$

$$\boldsymbol{a}(x) = \alpha Z_{RE}(x) \odot Z_{\mathcal{C}_x}(x) \qquad (11)$$

Code



Signal-50

	timestamp	signal	anomaly	...	signal47	signal48	signal49
0	2011-07-03 01:00:01	0.058761	0	...	0.065314	0.098972	0.081965
1	2011-07-03 02:00:01	0.059879	0	...	0.070663	0.065314	0.098972
2	2011-07-03 03:00:01	0.051283	0	...	0.102490	0.070663	0.065314
3	2011-07-03 04:00:01	0.059109	0	...	0.123395	0.102490	0.070663
4	2011-07-03 05:00:01	0.069140	0	...	0.153045	0.123395	0.102490

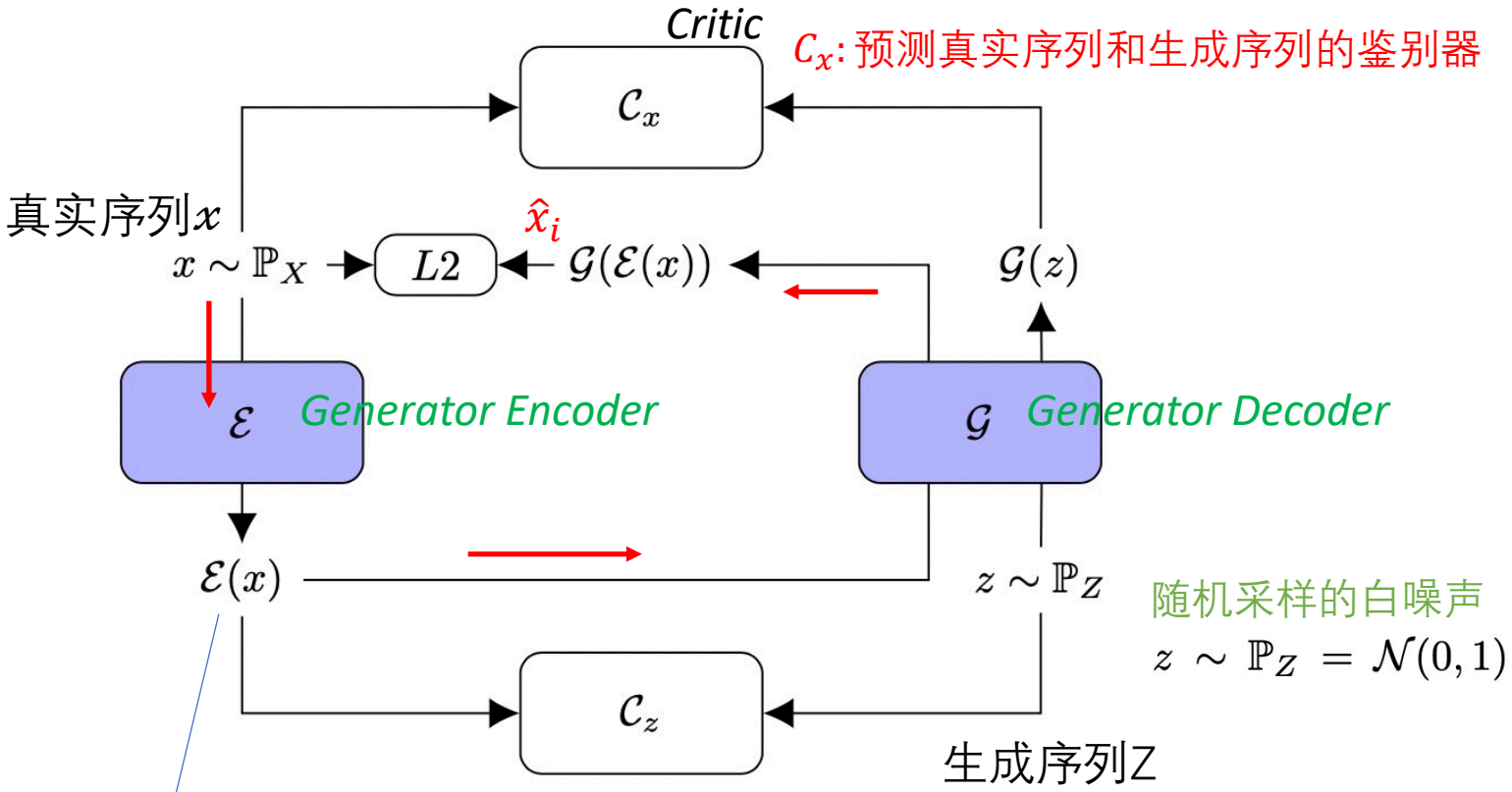
signal-1

signal+1

	timestamp	signal	anomaly	0.081965	0.081965
0	2011-07-01 00:00:01	0.081965	0	0.098972	0.098972
1	2011-07-01 01:00:01	0.098972	0	0.065314	0.065314
2	2011-07-01 02:00:01	0.065314	0	0.070663	0.070663
3	2011-07-01 03:00:01	0.070663	0	0.102490	0.102490
4	2011-07-01 04:00:01	0.102490	0	0.123395	0.123395
5	2011-07-01 05:00:01	0.123395	0	0.153045	0.153045
6	2011-07-01 06:00:01	0.153045	0	0.148322	0.148322
7	2011-07-01 07:00:01	0.148322	0	0.218258	0.218258
8	2011-07-01 08:00:01	0.218258	0	0.226598	0.226598
9	2011-07-01 09:00:01	0.226598	0	0.174046	0.174046
10	2011-07-01 10:00:01	0.174046	0	0.159902	0.159902
11	2011-07-01 11:00:01	0.159902	0	0.145626	0.145626
12	2011-07-01 12:00:01	0.145626	0	0.143829	0.143829
13	2011-07-01 13:00:01	0.143829	0	0.136605	0.136605
14	2011-07-01 14:00:01	0.136605	0	shift=-1	shift=+1

得到这个时间点，向后50个时间步长，向前50个时间步长

Code



a 1-layer bidirectional Long Short-Term Memory (LSTM) with 100 hidden units

```python
class Encoder(nn.Module):

    def __init__(self, encoder_path, signal_shape=100):
        super(Encoder, self).__init__()
        self.signal_shape = signal_shape
        self.lstm = nn.LSTM(input_size=self.signal_shape, hidden_size=20, num_layers=1, bidirectional=True)
        self.dense = nn.Linear(in_features=40, out_features=20)
        self.encoder_path = encoder_path
```

latent space (domain Z) is 20-dimensional. We use a 1-layer bidirectional Long Short-Term Memory (LSTM) with 100 hidden units as *Generator* $\mathcal{E}$, and a 2-layer bidirectional LSTM

```python
class Decoder(nn.Module):
    def __init__(self, decoder_path, signal_shape=100):
        super(Decoder, self).__init__()
        self.signal_shape = signal_shape
        self.lstm = nn.LSTM(input_size=20, hidden_size=64, num_layers=2, bidirectional=True)
        self.dense = nn.Linear(in_features=128, out_features=self.signal_shape)
        self.decoder_path = decoder_path
```

hidden units as *Generator* $\mathcal{E}$, and a 2-layer bidirectional LSTM with 64 hidden units each as *Generator* $\mathcal{G}$, where dropout is

训练鉴别器

只训练critic_x

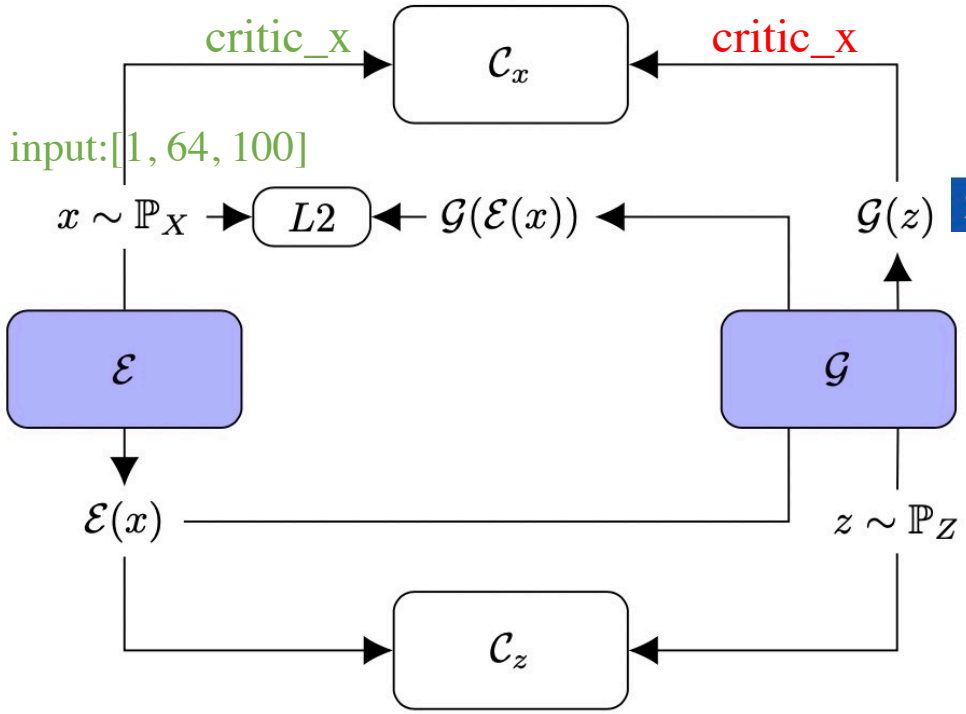
```
critic_score_valid_x = torch.mean(torch.ones(valid_x.shape) * valid_x) # Wasserstein Loss
```

$$V_X(\mathcal{C}_x, \mathcal{G}) = \mathbb{E}_{x\sim\mathbb{P}_X}[\mathcal{C}_x(x)] - \mathbb{E}_{z\sim\mathbb{P}_Z}[\mathcal{C}_x(\mathcal{G}(z))] \quad (3)$$

随机输入 z

```
z = torch.empty(1, batch_size, latent_space_dim).uniform_(0, 1)  # [1, 64, 20]; 随机生成向量
x_ = decoder(z)  # z -> decoder -> x_
fake_x = critic_x(x_)
fake_x = torch.squeeze(fake_x)
critic_score_fake_x = torch.mean(torch.ones(fake_x.shape) * fake_x)  # Wasserstein Loss

#Gradient Penalty Loss
gp_loss = torch.sqrt(torch.sum(torch.square(gradients).view(-1)))  # 均方误差
```



[1, 64, 100]

```
x_ = decoder(z)
```

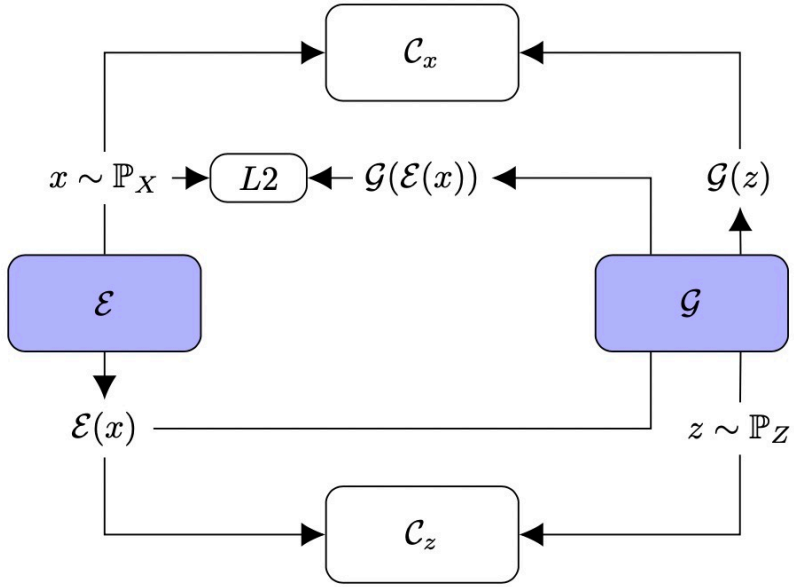
随机输入:[1, 64, 20]

```
def forward(self, x):
    x, (hn, cn) = self.lstm(x)  # 0->128
    x = self.dense(x)  # 128->100
    return (x)
```

Algorithm 1: TadGAN

Require: m, batch size.
$epoch$, number of iterations over the data.
n_{critic}, number of iterations of the critic per epoch.
η, step size.

for *each epoch* **do**
  for $\kappa = 0, \ldots, n_{critic}$ **do**
    Sample $\{(x_i^{1 \ldots t})\}_{i=1}^m$ from real data.
    Sample $\{(z_i^{1 \ldots k})\}_{i=1}^m$ from random.
    $g_{w_{\mathcal{C}_x}} = \nabla_{w_{\mathcal{C}_x}} [\frac{1}{m} \sum_{i=1}^m \mathcal{C}_x(x_i) - \frac{1}{m} \sum_{i=1}^m \mathcal{C}_x(\mathcal{G}(z_i)) + gp(x_i, \mathcal{G}(z_i))]$
    $w_{\mathcal{C}_x} = w_{\mathcal{C}_x} + \eta \cdot \texttt{adam}(w_{\mathcal{C}_x}, g_{w_{\mathcal{C}_x}})$
    $g_{w_{\mathcal{C}_z}} = \nabla_{w_{\mathcal{C}_z}} [\frac{1}{m} \sum_{i=1}^m \mathcal{C}_z(z_i) - \frac{1}{m} \sum_{i=1}^m \mathcal{C}_z(\mathcal{E}(x_i)) + gp(z_i, \mathcal{E}(x_i))]$
    $w_{\mathcal{C}_z} = w_{\mathcal{C}_z} + \eta \cdot \texttt{adam}(w_{\mathcal{C}_z}, g_{w_{\mathcal{C}_z}})$
  end



所以，论文作者就非常机智地提出，我们其实没必要在整个样本空间上施加Lipschitz限制，只要重点抓住生成样本集中区域、真实样本集中区域以及夹在它们中间的区域就行了。具体来说，我们先随机采一对真假样本，还有一个0-1的随机数：

$$x_r \sim P_r, x_g \sim P_g, \epsilon \sim Uniform[0,1] \quad \text{(公式7)}$$

然后在 x_r 和 x_g 的连线上随机插值采样：

$$\hat{x} = \epsilon x_r + (1-\epsilon) x_g \quad \text{(公式8)}$$

真实样本　生成样本

把按照上述流程采样得到的 $\hat{x}$ 所满足的分布记为 $P_{\hat{x}}$ ，就得到最终版本的判别器loss：

$$L(D) = -\mathbb{E}_{x \sim P_r}[D(x)] + \mathbb{E}_{x \sim P_g}[D(x)] + \lambda \mathbb{E}_{x \sim \mathcal{P}_{\hat{x}}}[||\nabla_x D(x)||_p - 1]^2 \quad \text{(公式9)}$$

$\hat{x}$的梯度

这就是新论文所采用的gradient penalty方法，相应的新WGAN模型简称为WGAN-GP。我们可以做一个对比：

生成样本　　正常样本

```
wl = critic_score_fake_x - critic_score_valid_x  # Vx loss; critic_score_fake_x越小越好, critic_score_valid_x越大越好
loss = wl + gp_loss  #
loss.backward()
optim_cx.step()
```

Loss越小越好

cost function越大越好

正常样本

$$g_{w_{C_x}} = \nabla_{w_{C_x}} [\frac{1}{m} \sum_{i=1}^{m} C_x(x_i) - \frac{1}{m} \sum_{i=1}^{m} C_x(G(z_i)) + gp(x_i, G(z_i))]$$

生成样本

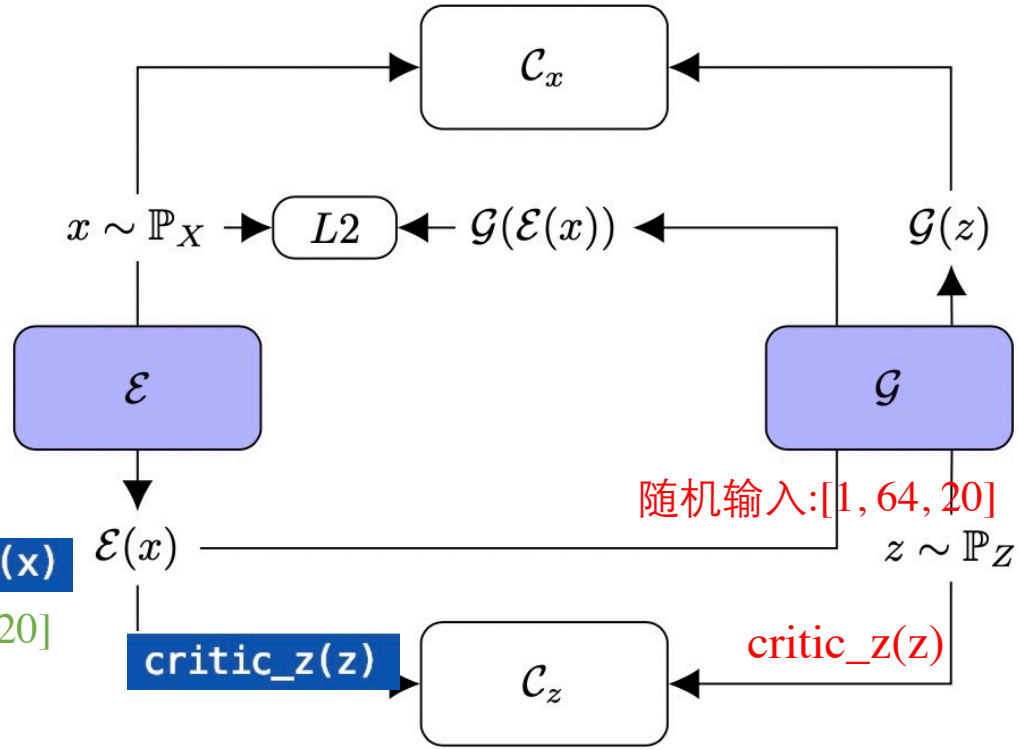
$$w_{C_x} = w_{C_x} + \eta \cdot \mathrm{adam}(w_{C_x}, g_{w_{C_x}})$$

梯度上升法

```
critic_score_valid_z = torch.mean(torch.ones(valid_z.shape) * valid_z)
```

$$\min_{\mathcal{E}} \max_{\mathcal{C}_z \in \mathbf{C_z}} V_Z(\mathcal{C}_z, \mathcal{E}) \tag{4}$$

```
critic_score_fake_z = torch.mean(torch.ones(fake_z.shape) * fake_z) #Wasserstein Loss
```




```
z = encoder(x)
```

[1, 64, 20]

```
def forward(self, x):
    x = self.dense1(x)
    return (x)
```

[1, 64, 1]

正常样本　　随机样本

```
wl = critic_score_valid_z - critic_score_fake_z
loss = wl + gp_loss
loss.backward()
optim_cz.step()
```

判断是否是随机样本

Loss function越小越好

随机样本

$$g_{w_{\mathcal{C}_z}} = \nabla_{w_{\mathcal{C}_z}} [\frac{1}{m} \sum_{i=1}^{m} \mathcal{C}_z(z_i) - \frac{1}{m} \sum_{i=1}^{m} \mathcal{C}_z(\mathcal{E}(x_i)) + gp(z_i, \mathcal{E}(x_i))]$$

正常样本

cost function越大越好

$$w_{\mathcal{C}_z} = w_{\mathcal{C}_z} + \eta \cdot \mathrm{adam}(w_{\mathcal{C}_z}, g_{w_{\mathcal{C}_z}})$$

鉴别随机:1
正常样本生成:0

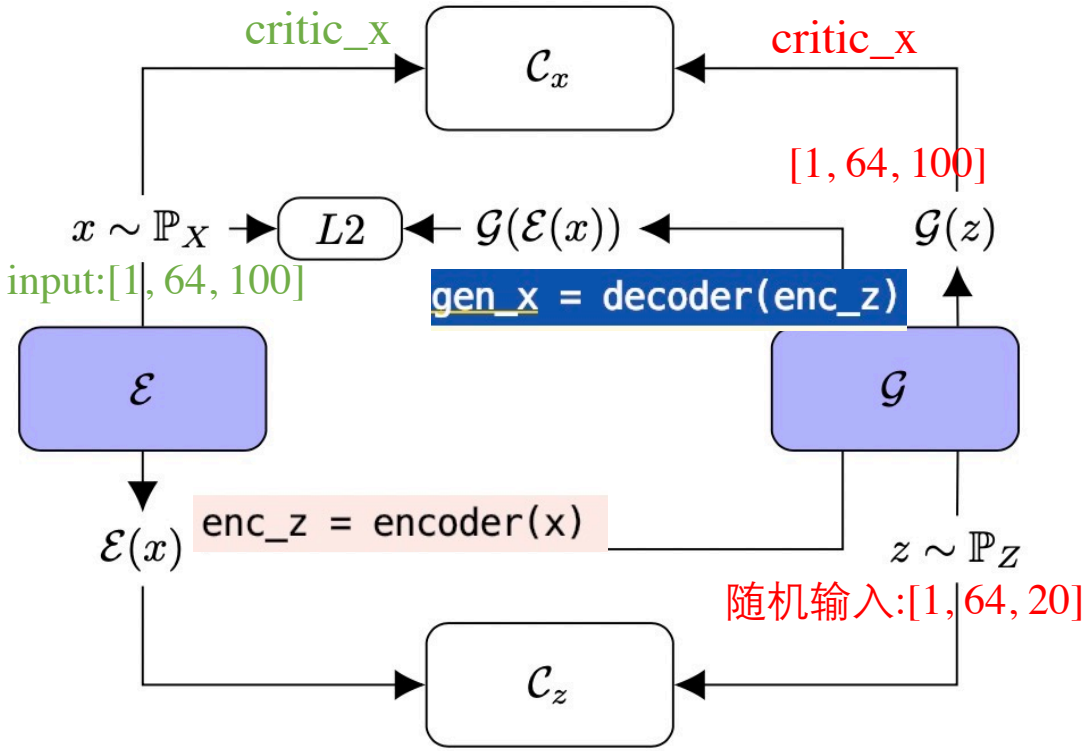
判断是否是随机样本

训练生成器

```
critic_score_valid_x = torch.mean(torch.ones(valid_x.shape) * valid_x)
critic_score_fake_x = torch.mean(torch.ones(fake_x.shape) * fake_x)
mse = mse_loss(x.float(), gen_x.float())  # mse误差
loss_enc = mse + critic_score_valid_x - critic_score_fake_x
```

$$V_{L2}(\mathcal{E},\mathcal{G}) = \mathbb{E}_{x\sim\mathbb{P}_X}[\|x-\mathcal{G}(\mathcal{E}(x))\|_2] \quad (5)$$

$$\min_{\{\mathcal{E},\mathcal{G}\}}\max_{\{\mathcal{C}_x\in\mathbf{C_x},\mathcal{C}_z\in\mathbf{C_z}\}} V_X(\mathcal{C}_x,\mathcal{G}) + V_Z(\mathcal{C}_z,\mathcal{E}) + V_{L2}(\mathcal{E},\mathcal{G}) \quad (6)$$




```
mse = mse_loss(x.float(), gen_x.float())  # mse误差; 越小越好
loss_enc = mse + critic_score_valid_x - critic_score_fake_x
loss_enc.backward(retain_graph=True)
optim_enc.step()
```

真实样本

生成样本

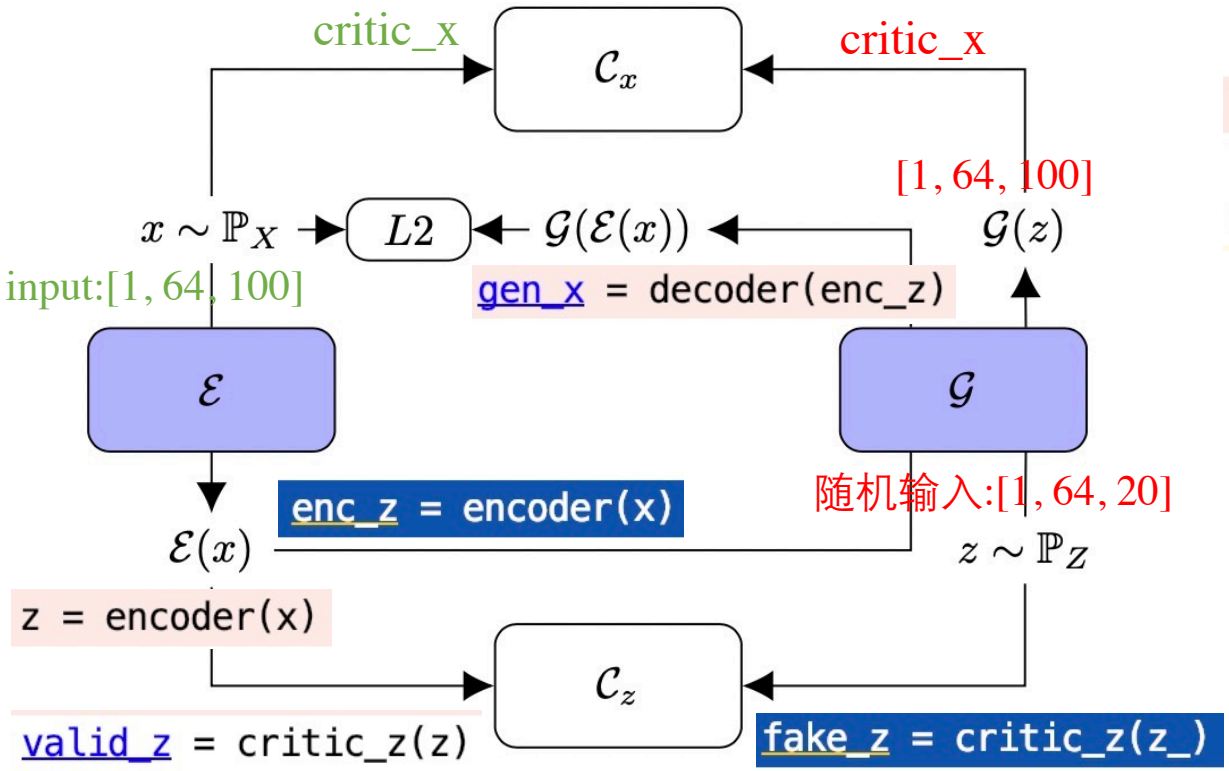
$$g_{w_{\mathcal{G},\mathcal{E}}} = \nabla_{w_{\mathcal{G}},w_{\mathcal{E}}}\Big[\frac{1}{m}\sum_{i=1}^{m}\mathcal{C}_x(x_i) - \frac{1}{m}\sum_{i=1}^{m}\mathcal{C}_x(\mathcal{G}(z_i)) + \frac{1}{m}\sum_{i=1}^{m}\mathcal{C}_z(z_i) - \frac{1}{m}\sum_{i=1}^{m}\mathcal{C}_z(\mathcal{E}(x_i)) + \frac{1}{m}\sum_{i=1}^{m}\|x_i - \mathcal{G}(\mathcal{E}(x_i))\|_2\Big]$$

损失函数，越小越好

$$w_{\mathcal{G},\mathcal{E},} = w_{\mathcal{G},\mathcal{E}} - \eta\cdot\mathrm{adam}(w_{\mathcal{G},\mathcal{E}}, g_{w_{\mathcal{G},\mathcal{E}}})$$

```
mse = mse_loss(x.float(), gen_x.float())
loss_dec = mse + critic_score_valid_z - critic_score_fake_z
loss_dec.backward(retain_graph=True)
optim_dec.step()
```

$$\min_{\{\mathcal{E},\mathcal{G}\}} \max_{\{\mathcal{C}_x \in \mathbf{C_x}, \mathcal{C}_z \in \mathbf{C_z}\}} V_X(\mathcal{C}_x, \mathcal{G}) + V_Z(\mathcal{C}_z, \mathcal{E}) + V_{L2}(\mathcal{E}, \mathcal{G}) \quad (6)$$



随机样本　　正常样本

```
loss_dec = mse + critic_score_fake_z - critic_score_valid_z
loss_dec.backward(retain_graph=True)
optim_dec.step()
```

$$g_{w_{\mathcal{G},\mathcal{E}}} = \nabla_{w_{\mathcal{G}},w_{\mathcal{E}}} [\frac{1}{m}\sum_{i=1}^{m} \mathcal{C}_x(x_i) - \frac{1}{m}\sum_{i=1}^{m} \mathcal{C}_x(\mathcal{G}(z_i)) + \frac{1}{m}\sum_{i=1}^{m} \mathcal{C}_z(z_i) - \frac{1}{m}\sum_{i=1}^{m} \mathcal{C}_z(\mathcal{E}(x_i)) + \frac{1}{m}\sum_{i=1}^{m} \|x_i - \mathcal{G}(\mathcal{E}(x_i))\|_2]$$

损失函数，越小越好

$$w_{\mathcal{G},\mathcal{E}}, = w_{\mathcal{G},\mathcal{E}} - \eta \cdot \mathrm{adam}(w_{\mathcal{G},\mathcal{E}}, g_{w_{\mathcal{G},\mathcal{E}}})$$

```
cx_epoch_loss.append(torch.mean(torch.tensor(cx_nc_loss)))
cz_epoch_loss.append(torch.mean(torch.tensor(cz_nc_loss)))
encoder_epoch_loss.append(torch.mean(torch.tensor(encoder_loss)))
decoder_epoch_loss.append(torch.mean(torch.tensor(decoder_loss)))
```

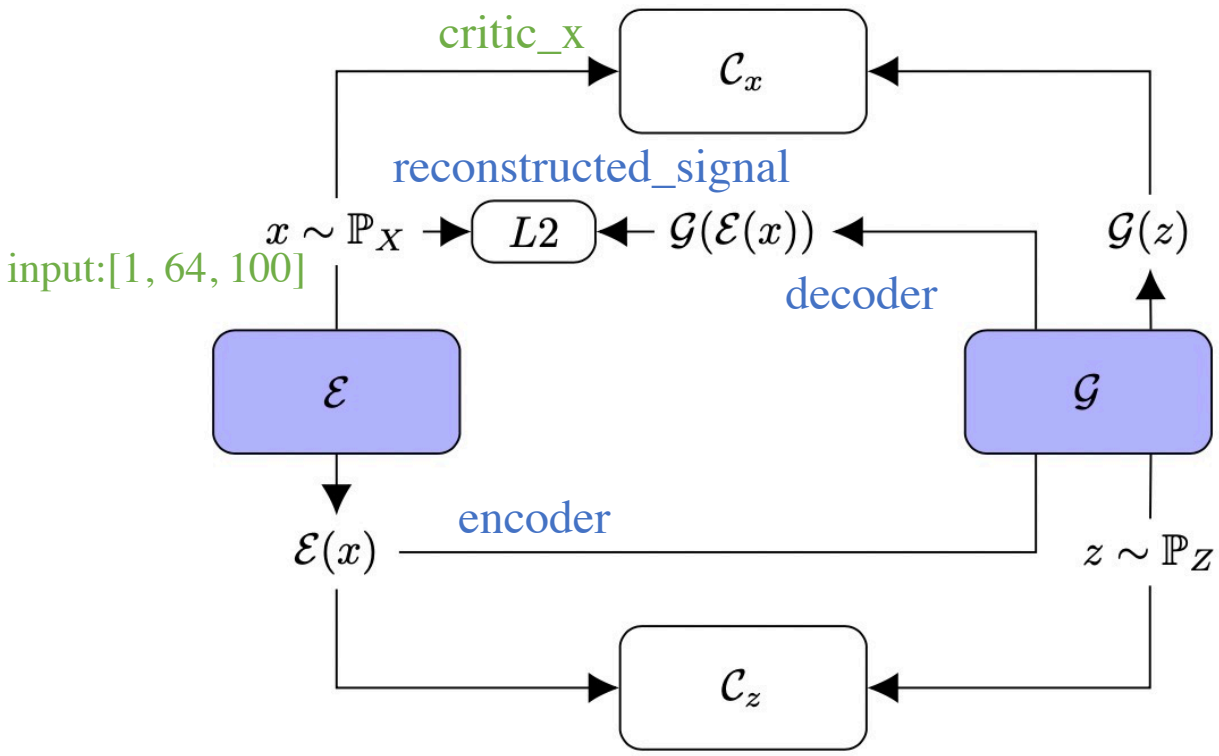
C_x loss

C_z loss

encoder loss

decoder loss

```
critic_score.extend(torch.squeeze(critic_x(sample['signal'])).detach().numpy())  # 计算鉴别器结果
```



```
reconstruction_error = stats.zscore(reconstruction_error)  # 计算zscore
critic_score = stats.zscore(critic_score)
anomaly_score = reconstruction_error * critic_score
```

Zscore：标准差和均值的距离

$$\boldsymbol{a}(x) = \alpha Z_{RE}(x) + (1-\alpha) Z_{C_x}(x) \qquad (10)$$

$$\boldsymbol{a}(x) = \alpha Z_{RE}(x) \odot Z_{C_x}(x) \qquad (11)$$

Detect_anomaly

```
window_size = len(anomaly_score) // 3  wind
step_size = len(anomaly_score) // (3 * 10)
```

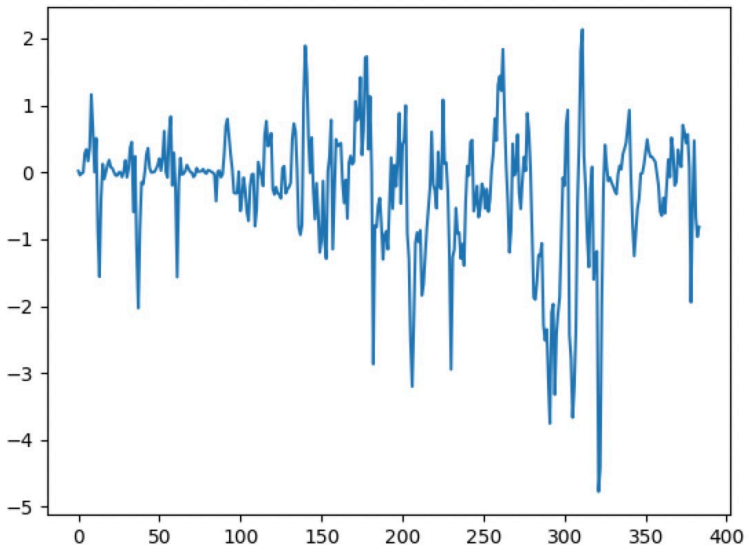
$$window\ size = \frac{T}{3}$$

$$step\ size = \frac{T}{3 * 10}$$

滑动窗口大小决定了用于评估当前阈值的历史异常分数的数量

```
for j, elt in enumerate(window_elts):
    if (window_mean - 3 * window_std) < elt < (window_mean + 3 * window_std):
        is_anomaly[i + j] = 0
    else:
        is_anomaly[i + j] = 1
```

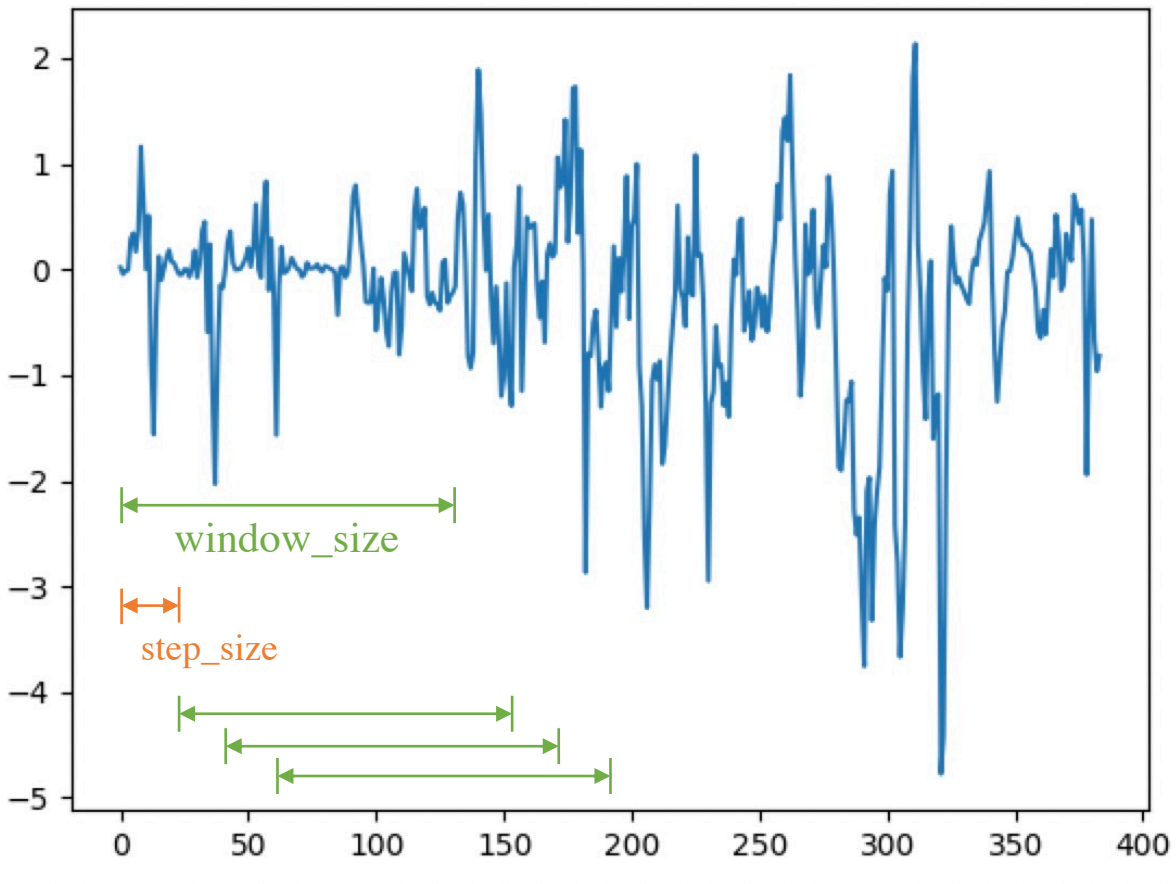
在3sigma之外认为是异常



anomaly_score=384

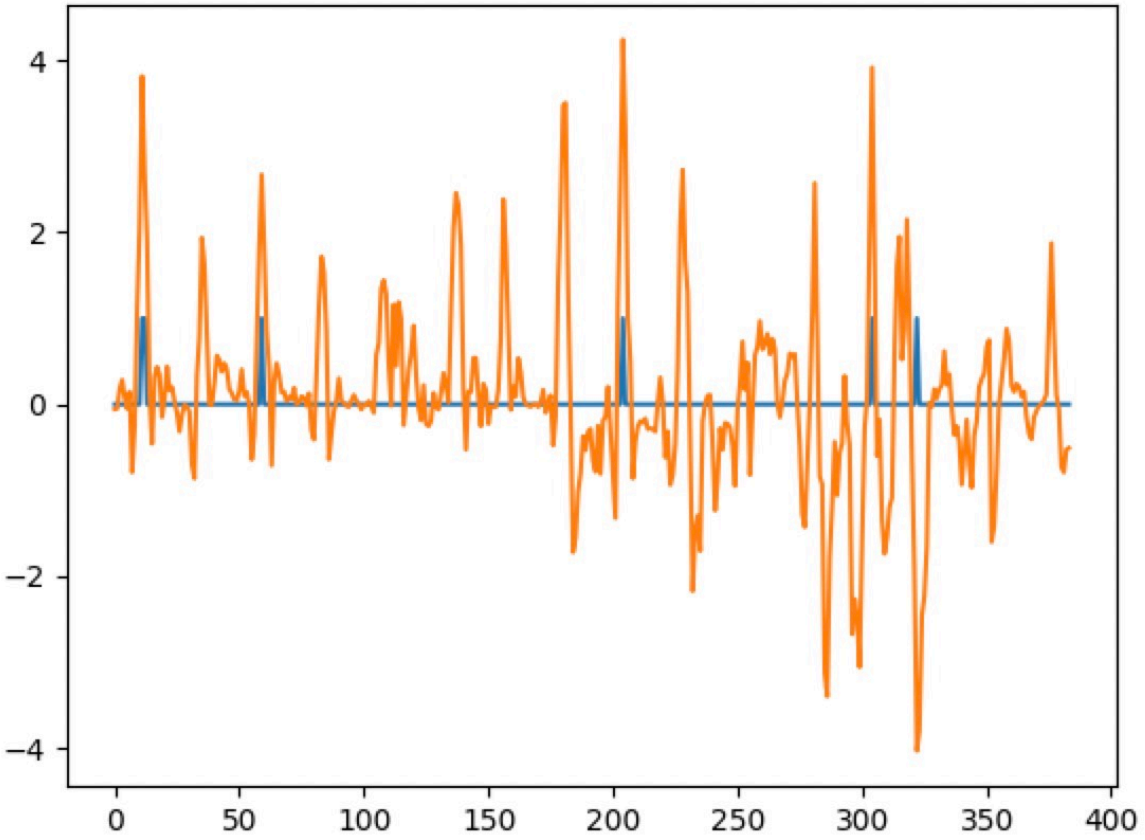
window_size=128

step_size=12



只有在多个时间窗口内，点都为异常，才定义为该点为异常值。

如果一个点在一周期内为异常，但新周期内，波动导致该异常没到异常阈值，则认为之前为误报异常。



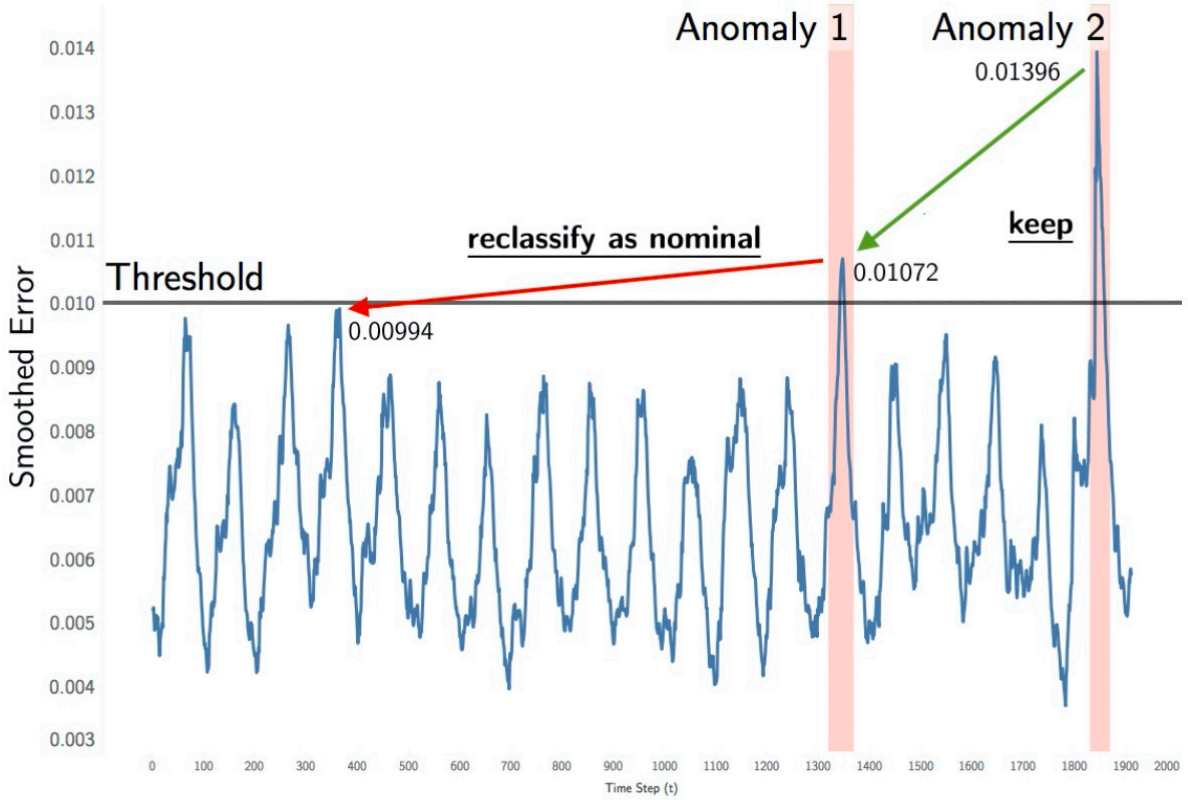


Figure 2: An example demonstrating the anomaly pruning process. In this scenario $\mathbf{e}_{max} = [0.01396, 0.01072, 0.00994]$ and the minimum percent decrease $p = 0.1$. The decrease from Anomaly 2 to Anomaly 1 $d^{(1)} = 0.23 > p$ and this sequence retains its classification as anomalous. From Anomaly 1 to the next highest smoothed error ($e_s = 0.0099$) $d^{(2)} = .07 < p$ so this sequence is re-classified as nominal.

减少误报

记录一个异常

```
for i in range(1, len(is_anomaly)):  i: 315
    if i+1 == len(is_anomaly):  # 倒数第二个元素
        seq_details.append([start_position, i, max_seq_element, delete_sequence])
    elif is_anomaly[i] == 1 and is_anomaly[i+1] == 0:  # 异常之后，为正常
        end_position = i  # 异常结束index
        seq_details.append([start_position, end_position, max_seq_element, delete_sequence])
    elif is_anomaly[i] == 1 and is_anomaly[i-1] == 0:  # 异常的上一个元素为，正常
        start_position = i  # 开始元素
        max_seq_element = anomaly_score[i]  # 异常值
    if is_anomaly[i] == 1 and is_anomaly[i-1] == 1 and anomaly_score[i] > max_seq_element:
        max_seq_element = anomaly_score[i]
```

```
In[2]: seq_details
Out[2]: [[0, 60, -0.025287094455741056, 0]]

Out[9]: [315, 316, 4.351647958551586, 0]

Out[10]: [315, 322, 4.351647958551586, 0]

Out[11]: [315, 383, 4.351647958551586, 0]
```

最后一个

```
change_percent = abs(max_elements[1:] - max_elements[:-1]) / max_elements[1:]
```

$$p^i = (a_{max}^{i-1} - a_{max}^{i}) / a_{max}^{i-1}$$

(下一个 - 前一个) / 前一个

is_anomaly: 多个滑动窗口判定为信号是否异常

anomaly_score: 计算z分数的乘积 $\boldsymbol{a}(x) = \alpha Z_{RE}(x) \odot Z_{C_x}(x)$ (11)

$\{\boldsymbol{a}_{max}^i, i = 1, 2, \ldots, K\}$ 每个序列的最大异常值

$p^i = (\boldsymbol{a}_{max}^{i-1} - \boldsymbol{a}_{max}^i)/\boldsymbol{a}_{max}^{i-1}$ 如果p^i不超过阈值$\theta(0.1)$，则认为是正常

首先，对于每个异常序列，我们使用最大值 异常评分来表示它

```python
for i in range(1, len(is_anomaly)):  i: 1
    if i+1 == len(is_anomaly):  # 最后一个元素
        seq_details.append([start_position, i, max_seq_element, delete_sequence])
    elif is_anomaly[i] == 1 and is_anomaly[i+1] == 0:  # 异常之后，为正常
        end_position = i  # 异常结束index
        seq_details.append([start_position, end_position, max_seq_element, delete_sequence])
    elif is_anomaly[i] == 1 and is_anomaly[i-1] == 0:  # 异常的上一个元素为，正常
        start_position = i  # 开始元素
        max_seq_element = anomaly_score[i]  # 异常值
    if is_anomaly[i] == 1 and is_anomaly[i-1] == 1 and anomaly_score[i] > max_seq_element:
        max_seq_element = anomaly_score[i]
```

先判断这个值是否为异常

异常之后为正常

[0,0,0,0,0,**1**,0,0,0……]

start end sql_details

异常，之前一个为正常，（之后一个为异常）

[0,0,0,0,0,**1**,1,0,0……]

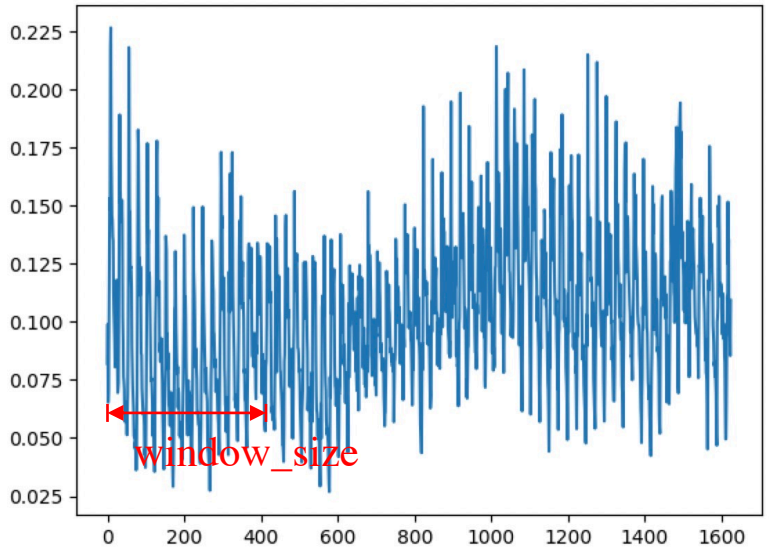
start max_seq_element更新

异常前一个也是异常，（后一个也是异常），并且大于最大异常分值

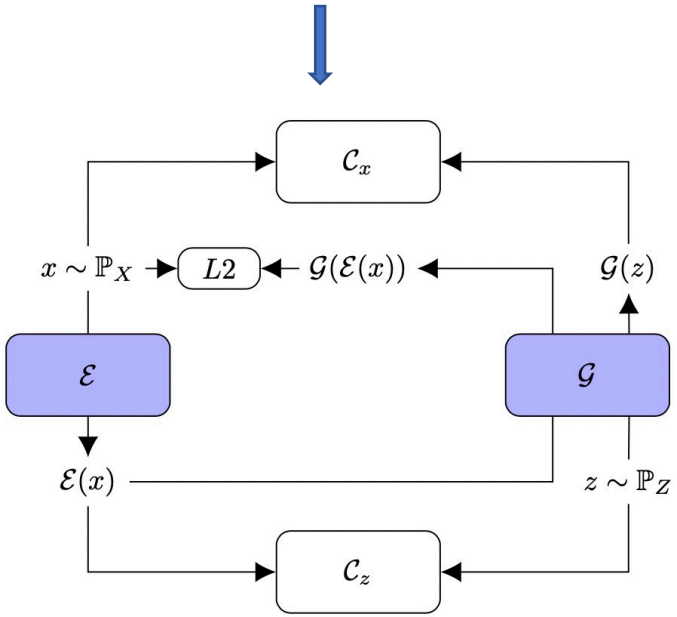
[0,0,0,0,**1**,**1**,1,0,0……]

max_seq_element更新

原始数据



[-50, -49, ,,, 0,,, 49]: 一共100个数据点

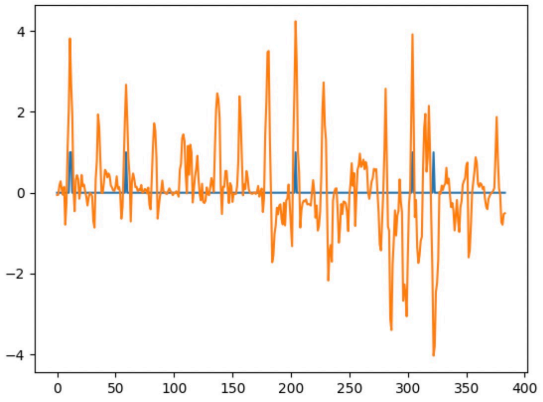


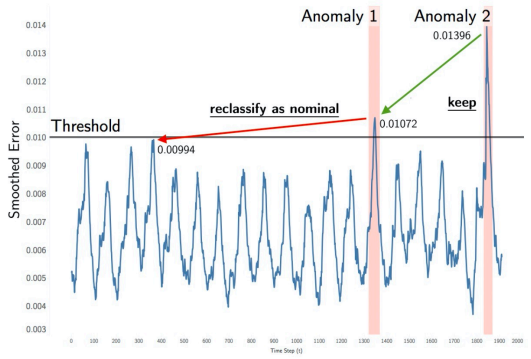
测试数据，计算两种误差

1）C_x误差

2）L_2误差

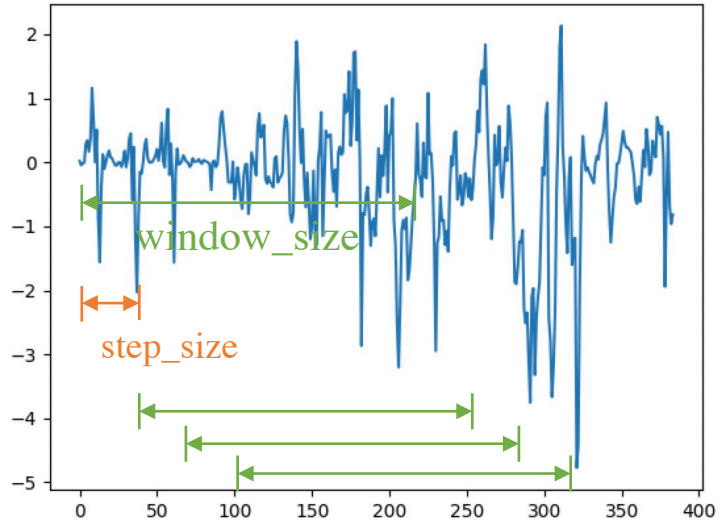
计算异常偏移标准差



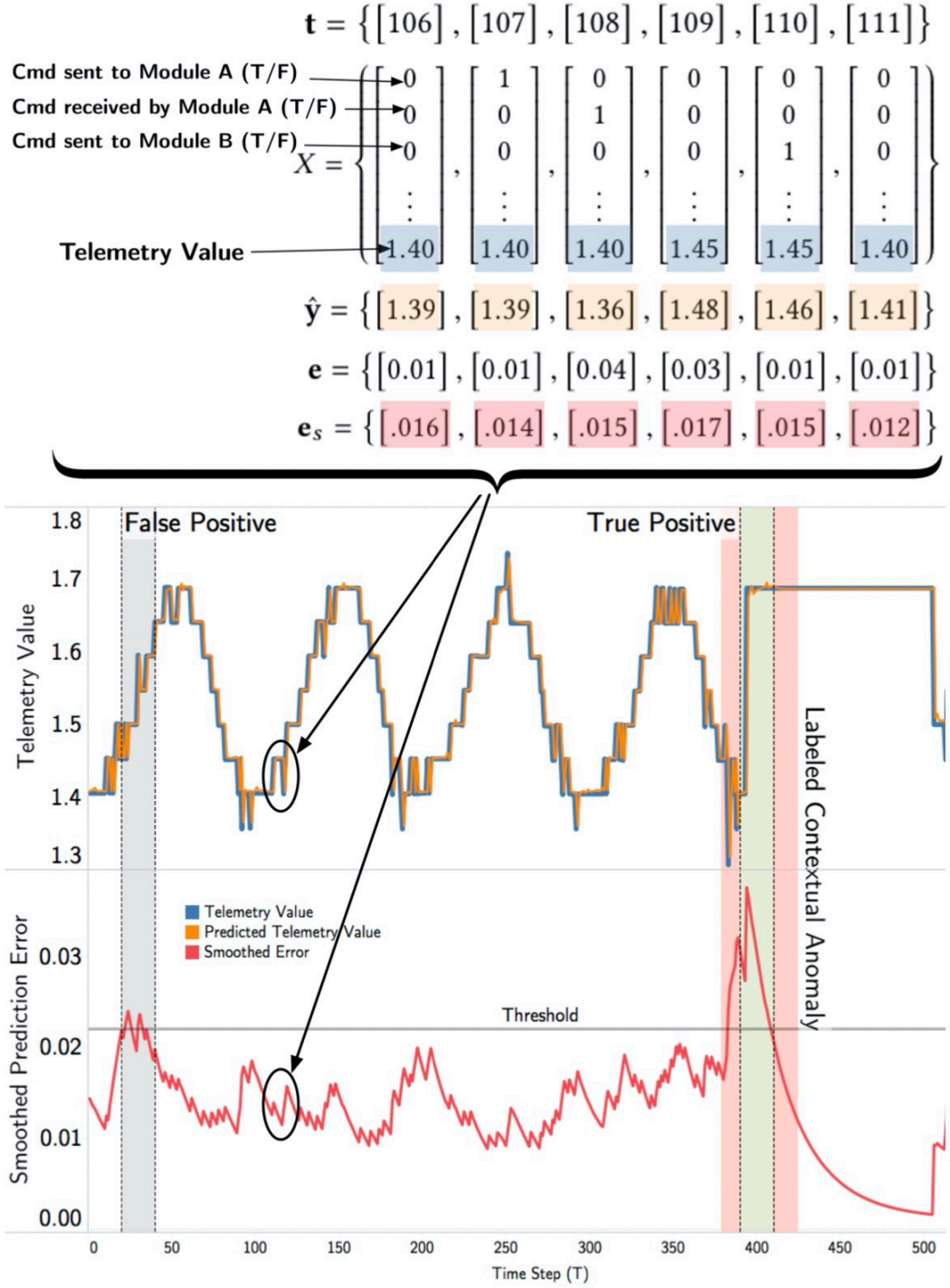


重新提纯误差

计算在每个window中，计算偏移标准差



计算两个z-score乘积，作为异常分值



对包含上下文异常的遥测流演示了命令信息的编码，使用基于限制或距离的方法不太可能识别该异常。 使用编码的命令信息和通道的先前遥测值，为下一个时间步长生成预测，并产生错误。 如顶部时间序列图所示，在此示例中，提前一步预测和实际遥测值非常接近。 使用第 3.2 节中详述的非参数阈值方法设置错误阈值，从而产生两个预测的异常序列——一个假阳性和一个真阳性位于标记的异常区域内。 误报表明需要在第 3.3 节中描述修剪，这将重新分类该序列为名义序列，因为它相对接近低于阈值的值（见图 2）。

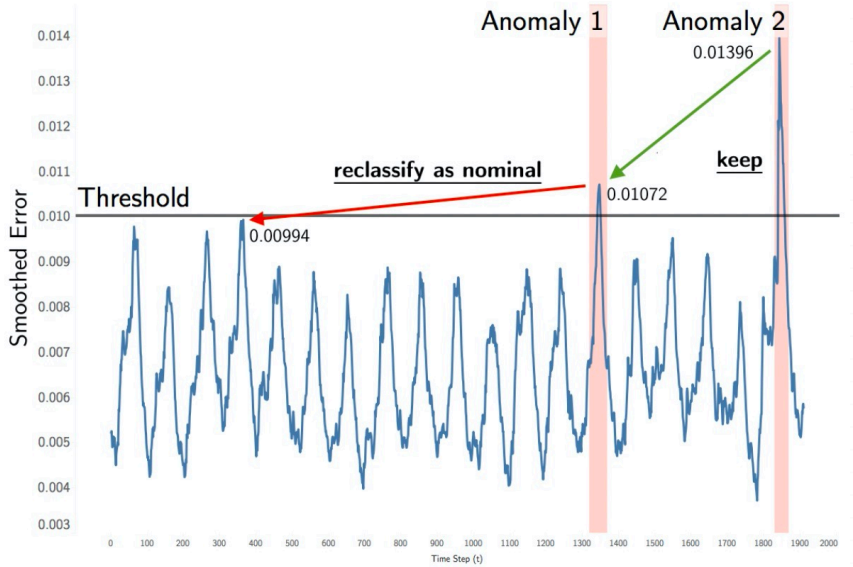
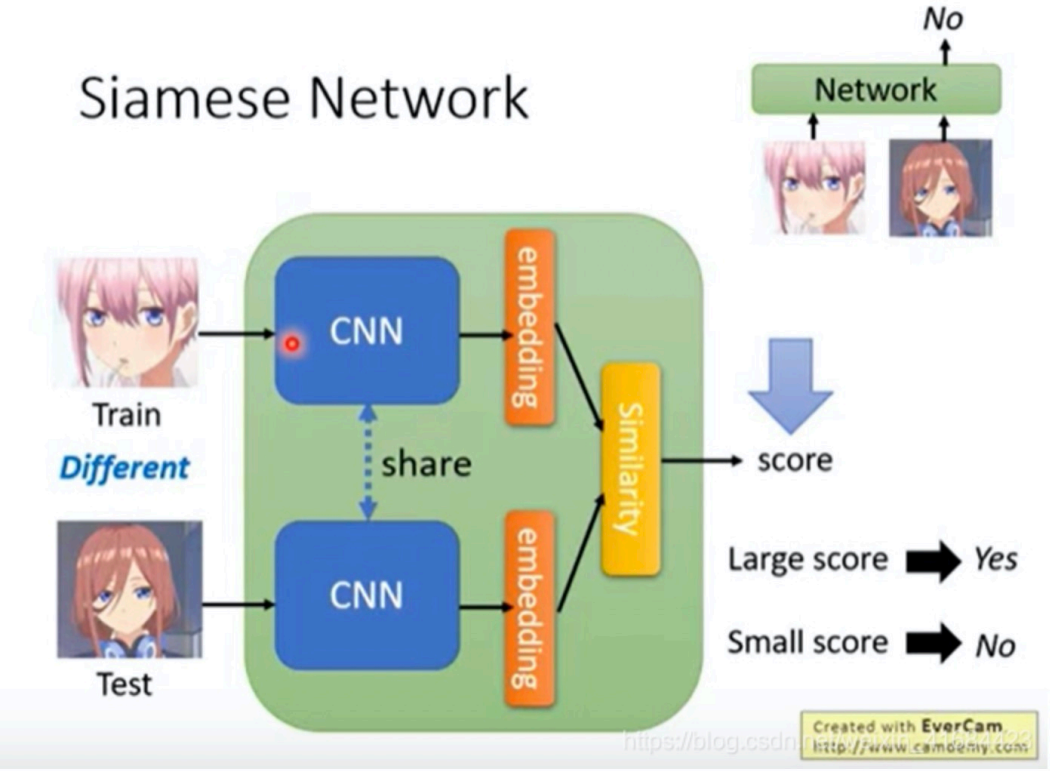


Figure 2: An example demonstrating the anomaly pruning process. In this scenario $\mathbf{e}_{max}$ = [0.01396, 0.01072, 0.00994] and the minimum percent decrease $p = 0.1$. The decrease from Anomaly 2 to Anomaly 1 $d^{(1)} = 0.23 > p$ and this sequence retains its classification as anomalous. From Anomaly 1 to the next highest smoothed error ($e_s = 0.0099$) $d^{(2)} = .07 < p$ so this sequence is re-classified as nominal.

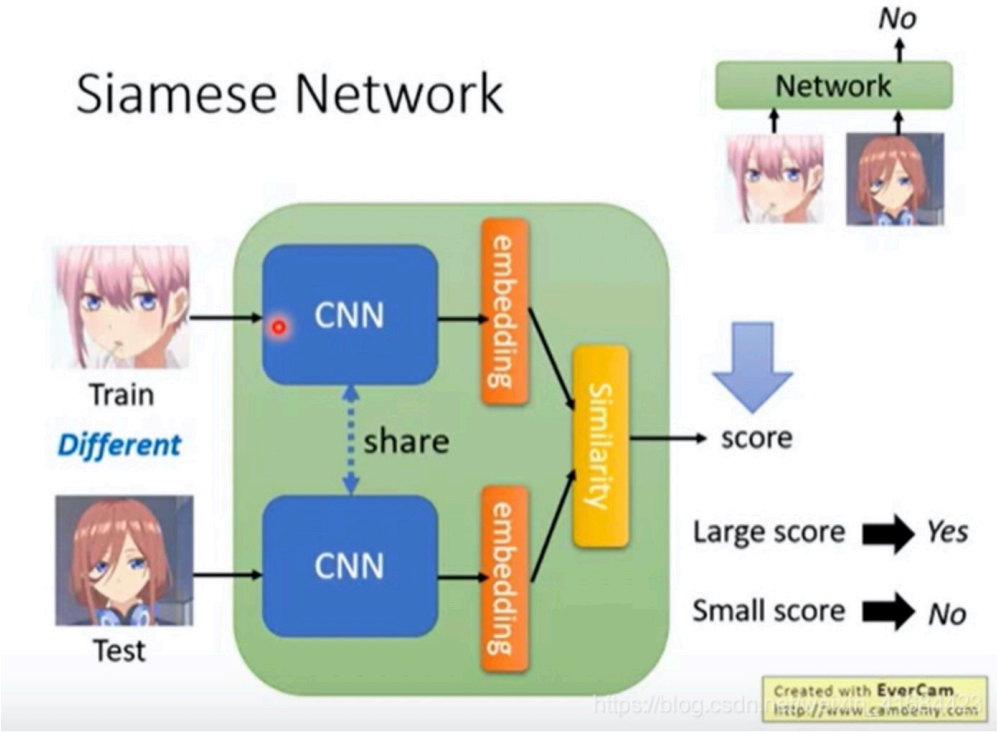
TS2Vec: Towards Universal Representation of Time Series

采用对比学习的架构，对每个节点生成通用的时间序列的表示方法。
该方法不仅可以应用于异常检测，可以应用于时间序列的预测(回归问题)，时间序列分类。

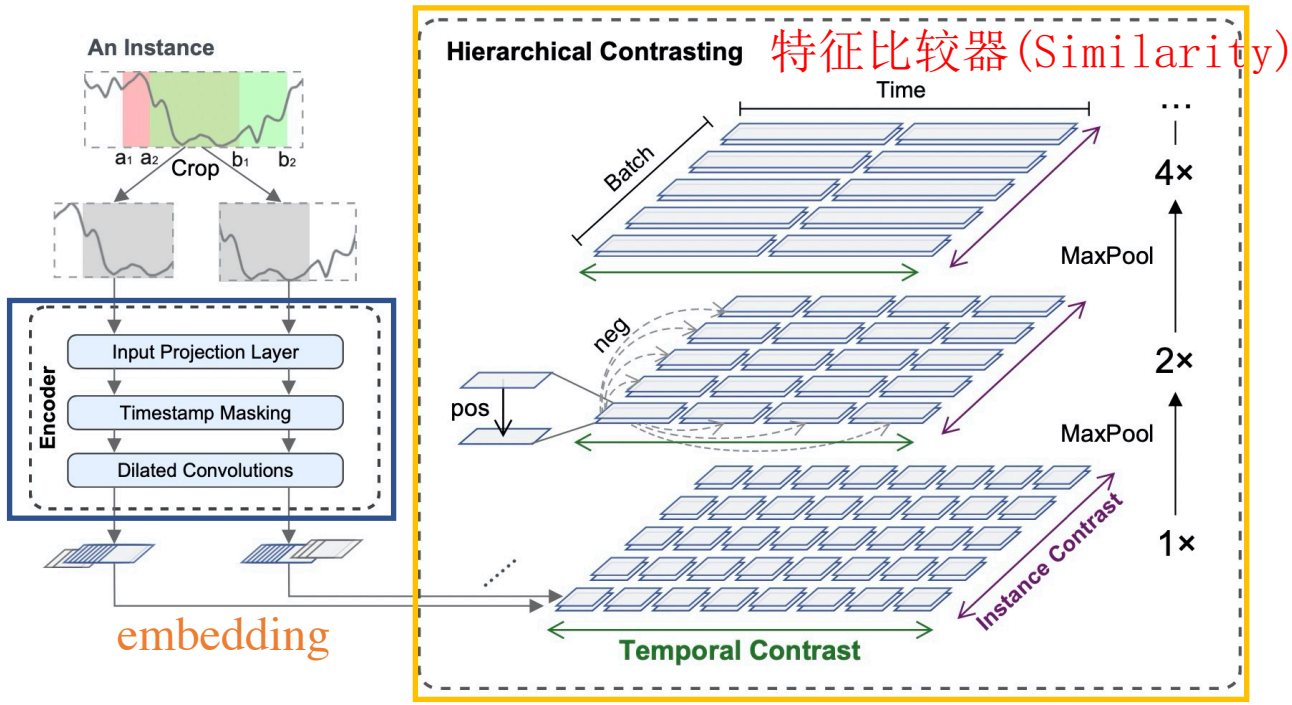


孪生网络：https://www.bilibili.com/video/BV1Gb411n7dE?p=44

Input: 两个图像(正样本和负样本)



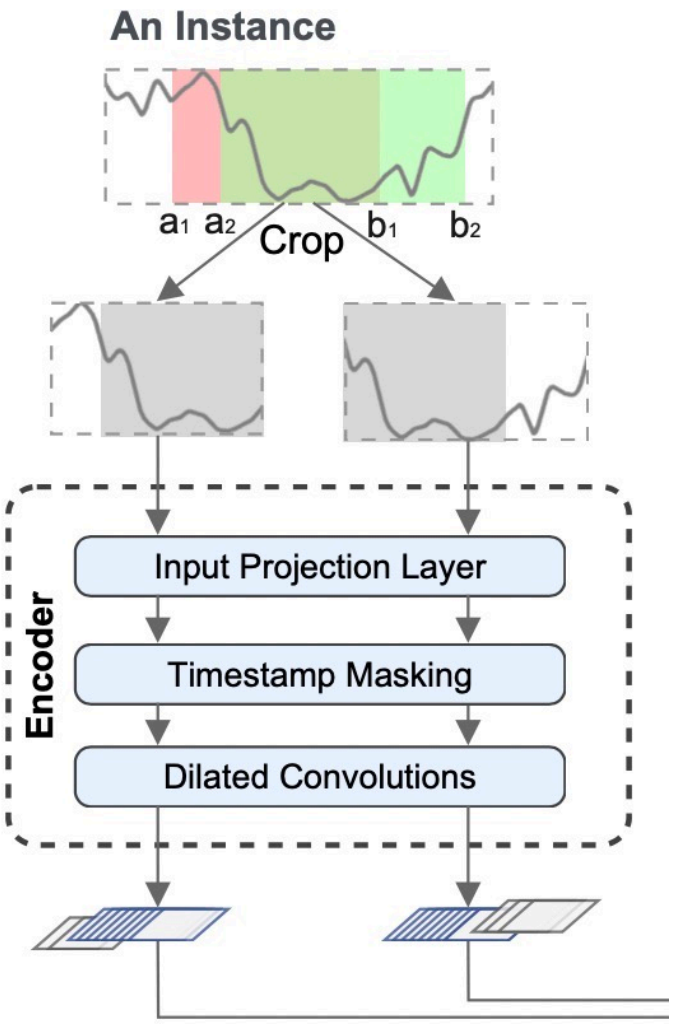
Input: 两个时间序列(正样本和负样本)



了解结构后，我们看下如何进行正样本和负样本的选择？

孪生网络：https://www.bilibili.com/video/BV1Gb411n7dE?p=44

增强采样



t_1 t_2 t_3 t_4 t_{13}

[-1.2, -1.16, -0.81, -0.77, -0.85, -0.62, -0.98, -1.23, -0.89, -0.89, -1.17, -1.26, -0.55, -0.74, -1.37]

时序特征: t_4~t_{13}

$t_{4\sim13}$: [-0.77, -0.85, -0.62, -0.98, -1.23, -0.89, -0.89, -1.17, -1.26, -0.55]

样本1

t_4 t_5 a_3

[-1.2, -1.16, -0.81, -0.77, -0.85, -0.62, -0.98, -1.23, -0.89, -0.89, -1.17, -1.26, -0.55, -0.74, -1.37]

a_1 a_2 a_4

样本2

样本1和**样本2**提取出来的特征，对每个时间点形成两种不同的特征表示，这样，我们就可以构建同时刻的成对正样本

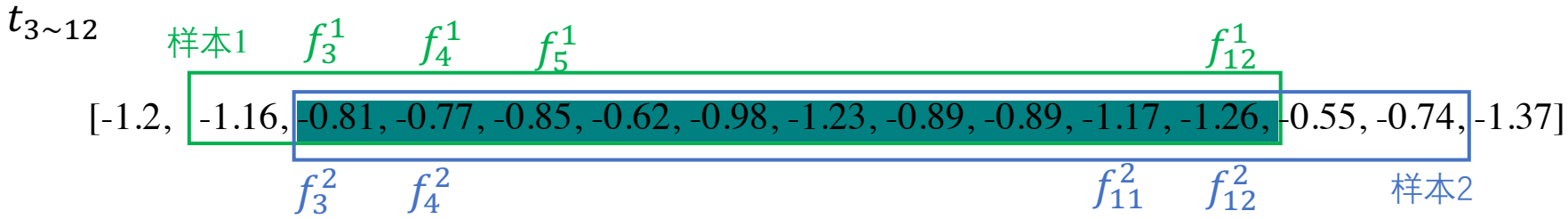
样本1经过模型后，t_4时刻会对应一个embedding1
样本2经过模型后，t_4时刻会对应一个embedding2

增强采样作用：

1. 构造出我们需要的正样本pair对
2. 因为时间序列具有连续性，前后随机增加一些时间，如果模型也能达到同样的效果，说明模型泛化能力也会更强。

作者使用了两种方式去构造正负样本。 Temporal Contrastive and Instance-wise

Temporal Contrastive：不同时刻的特征构造成负样本

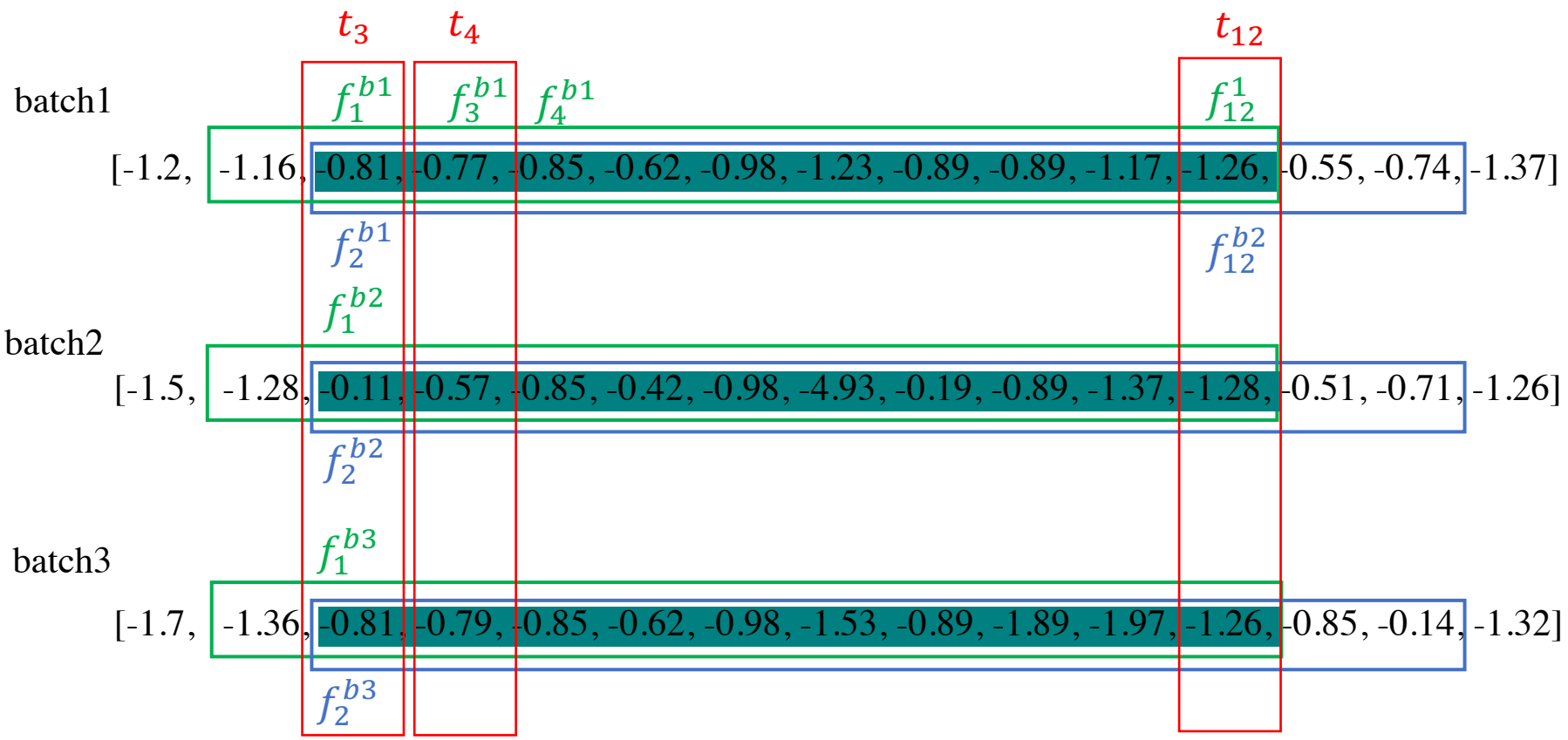


正样本：f_3^1和f_3^2； f_4^1和f_4^2； f_{12}^1和f_{12}^2

负样本：f_3^1和f_4^1； f_3^1和f_4^2； f_3^1和f_{12}^2

不是同时间的特征构造的都是负样本

Instance-wise ：同时刻，不同batch的特征构造出负样本



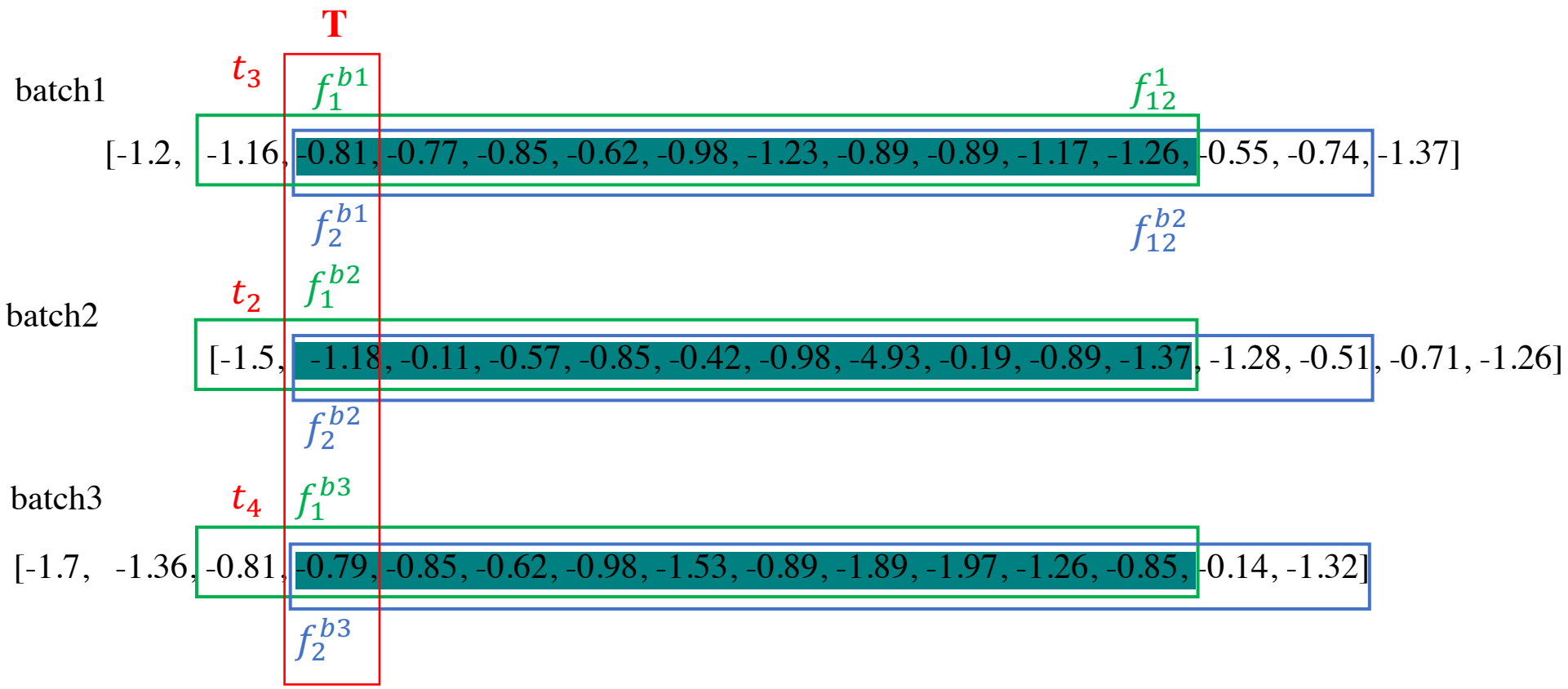
对于t_3时刻：

正样本：f_1^{b1}和f_2^{b1}; f_1^{b2}和f_2^{b2}; f_1^{b3}和f_2^{b3};　同batch中样本为正样本

负样本：f_1^{b1}和f_2^{b2}; f_1^{b1}和f_1^{b2}; f_1^{b3}和f_2^{b2};

只要不是同batch中(不同节点)，形成的组合都是负样本

Instance-wise ：对不同batch前后移动一个随机距离，这样更能使得负样本之间存在时间的差异



对于T时间：

正样本：f_1^{b1}和f_2^{b1}; f_1^{b2}和f_2^{b2}; f_1^{b3}和f_2^{b3};　同batch中样本为正样本

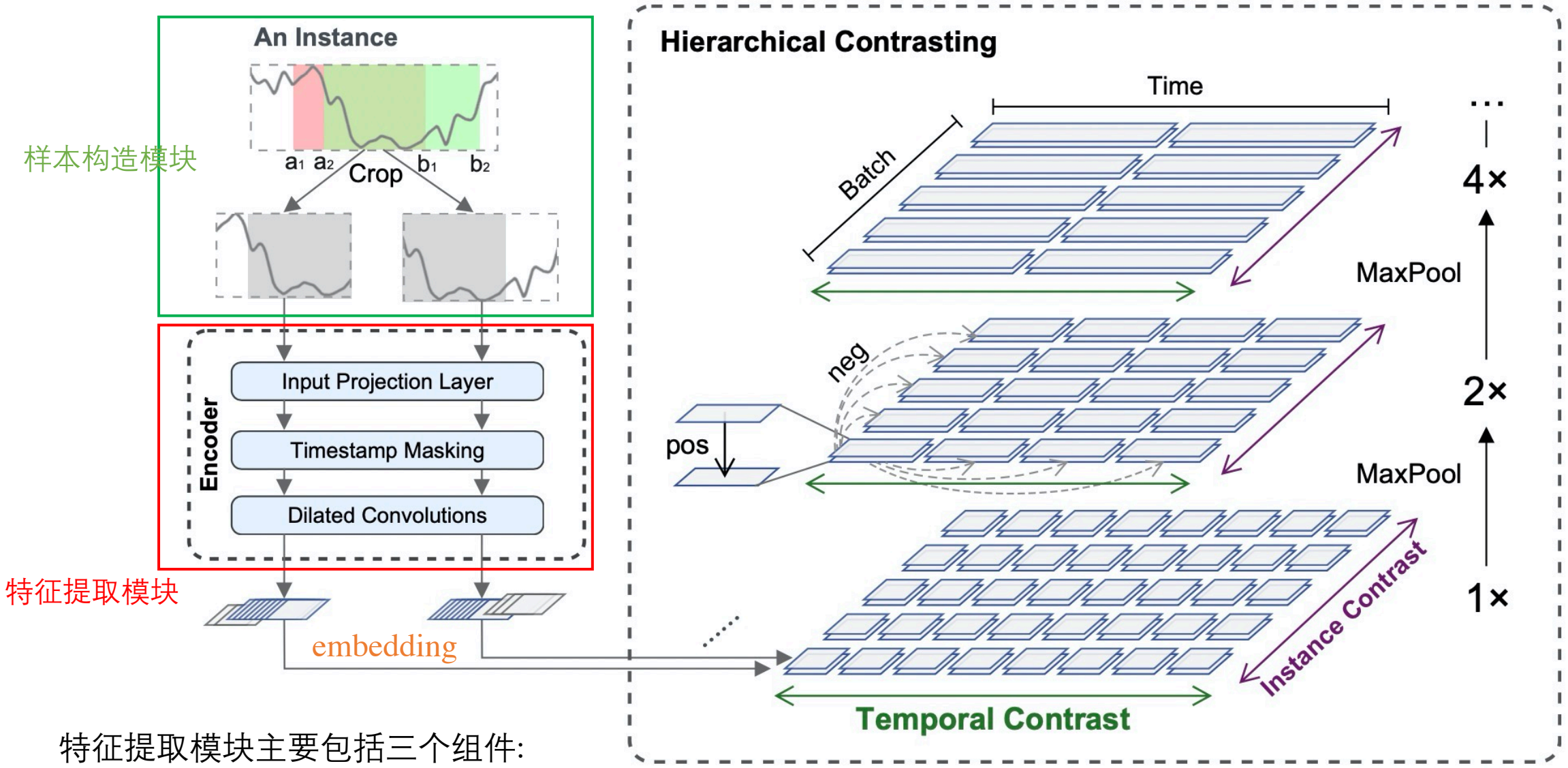
负样本：f_1^{b1}和f_2^{b2}; f_1^{b1}和f_1^{b2}; f_1^{b3}和f_2^{b2};

只要不是同batch中(不同节点)，形成的组合都是负样本

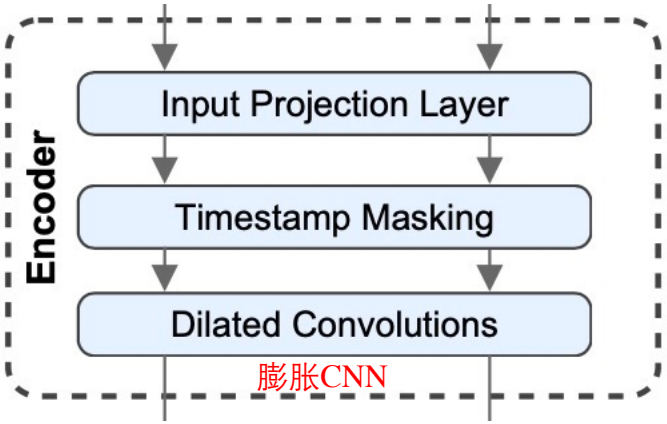
网络架构

样本构造模块

特征提取模块



特征提取模块主要包括三个组件:
an input projection layer
a timestamp masking module
a dilated CNN module.



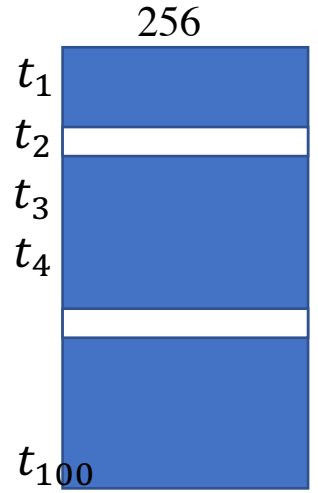
比如原始的时间序列长度为100，时间特征维度56，经过全连接层，转换成256维度，输出[100, 256]

$x_{i,t}$全链接层映射到隐层$z_{i,t}$

随机Mask隐层一部分时间序列，增强表征

膨胀卷积以扩大感受野

Mask:

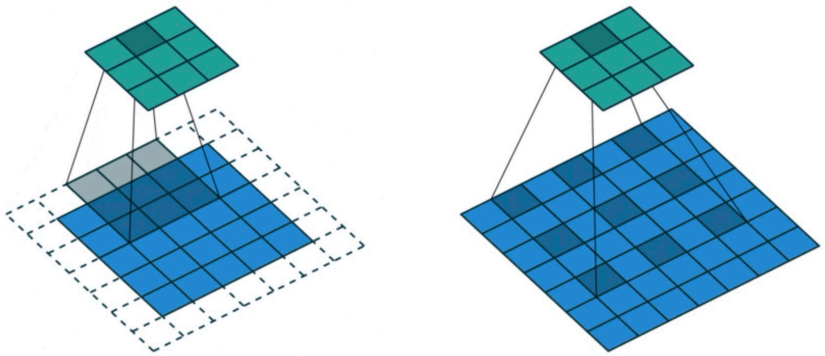


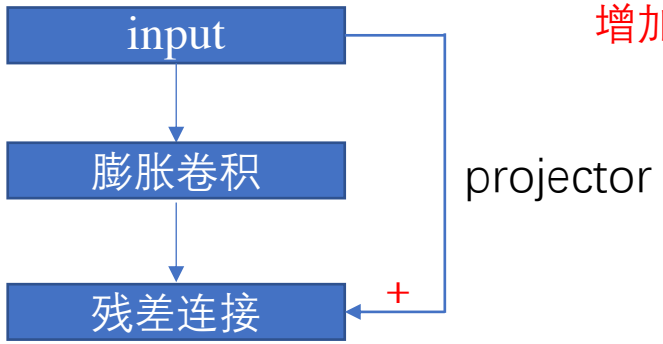
Input Projection Layer: 全连接层，将原始时序特征转换到高维特征

Timestamp Masking pos: 对Input Projection Layer输出的高维特征进行Mask操作

Dilated Convolutions: 膨胀卷积，增加感受野

模拟时间缺失情况，
增加模型对于时间特征的鲁棒性



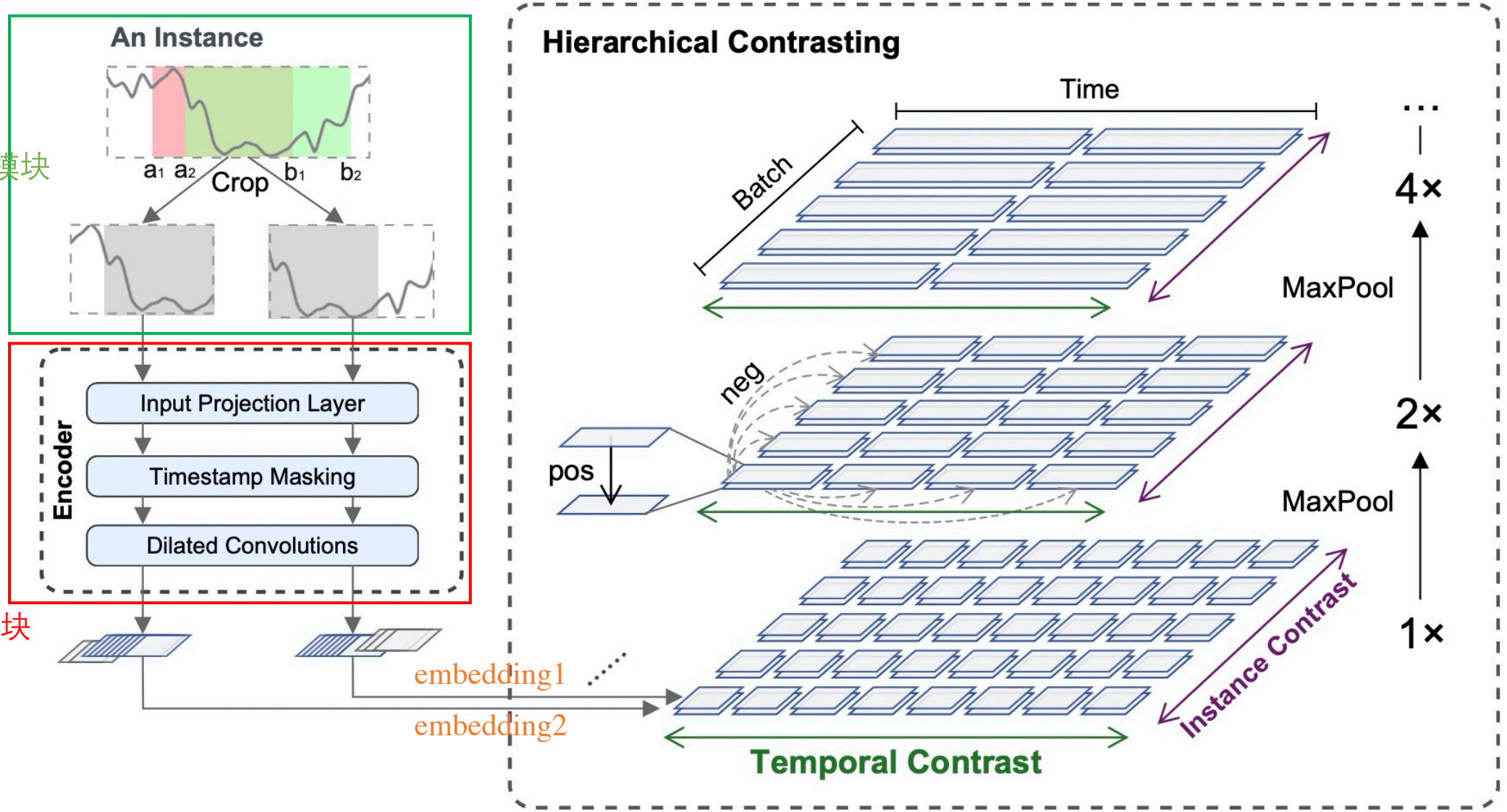


膨胀卷积: https://blog.csdn.net/qq_30241709/article/details/88080367

网络架构

样本构造模块

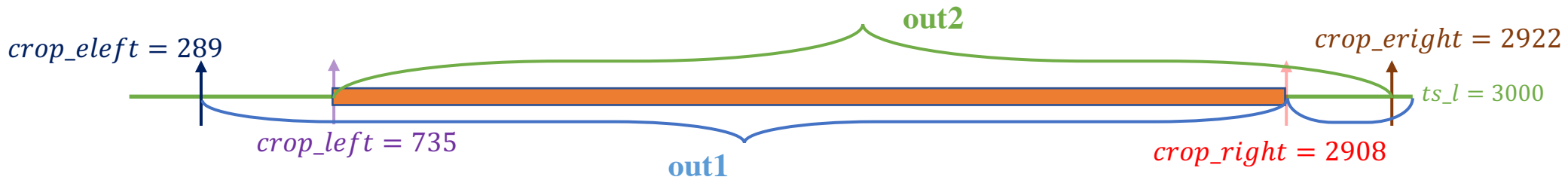
特征提取模块

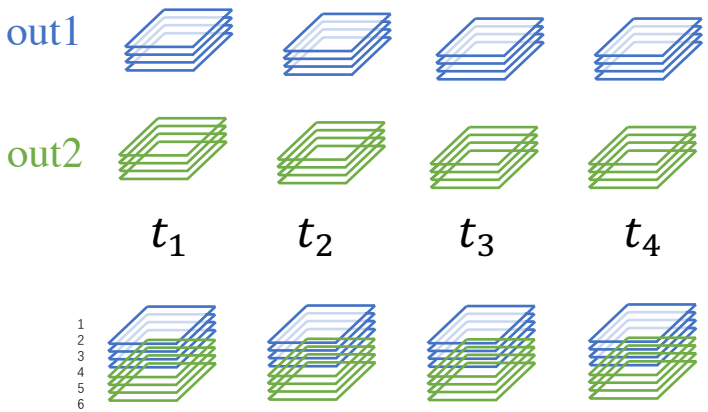


分层对比loss

Loss计算方法: instance_contrastive_loss and temporal_contrastive_loss

instance_contrastive_loss





3个batch

3个batch

t_1时刻不同batch相似性信息

s(1,2)	s(2,1)	s(6,1)
s(1,3)	s(2,3)	s(6,2)
s(1,4)	s(2,4)	s(6,3)
s(1,5)	s(2,5)	s(6,4)
s(1,6)	s(2,6)	s(6,5)

$$\ell_{inst}^{(i,t)} = -\log \frac{\exp(r_{i,t}\cdot r'_{i,t})}{\sum_{j=1}^{B}\left(\exp(r_{i,t}\cdot r'_{j,t})+\mathbb{1}_{[i\neq j]}\exp(r_{i,t}\cdot r_{j,t})\right)}, \quad (2)$$

$batch_i$在t时刻的loss

t时刻所有样本的和

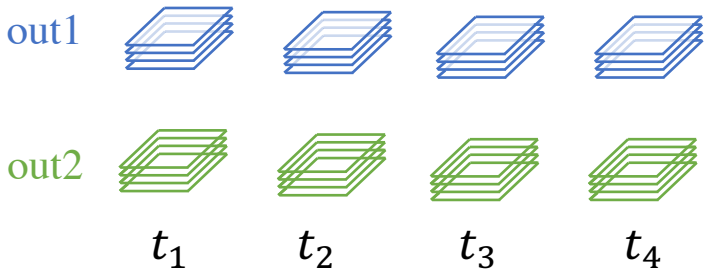
$$\ell_{inst}^{(1,t_1)} = -\log\left(\frac{\exp(s(1,4))}{\exp(s(1,2))+\exp(s(1,3))+\exp(s(1,4))+\exp(s(1,5))+\exp(s(1,6))}\right)$$

$batch_1$在t_1时刻的loss

对所有batch，所有时刻的loss求和

$$\mathcal{L}_{dual} = \frac{1}{NT}\sum_i\sum_t\left(\ell_{temp}^{(i,t)}+\ell_{inst}^{(i,t)}\right). \quad (3)$$

temporal_contrastive_loss



3个batch

3个batch

batch1

batch1中不同时刻的相似性

s(1,2)	s(2,1)	s(8,1)
s(1,3)	s(2,3)	s(8,2)
s(1,4)	s(2,4)	s(8,3)
s(1,5)	s(2,5)	s(8,4)
s(1,6)	s(2,6)	s(8,5)
s(1,7)	s(2,7)	s(8,6)
s(1,8)	s(2,8)	s(8,7)

$$\ell_{temp}^{(i,t)} = -\log \frac{\exp(r_{i,t} \cdot r'_{i,t})}{\sum_{t' \in \Omega} \left(\exp(r_{i,t} \cdot r'_{i,t'}) + \mathbb{1}_{[t \neq t']} \exp(r_{i,t} \cdot r_{i,t'}) \right)}, \quad (1)$$

$$\mathcal{L}_{dual} = \frac{1}{NT} \sum_i \sum_t \left(\ell_{temp}^{(i,t)} + \ell_{inst}^{(i,t)} \right). \quad (3)$$

对所有时刻，所有batch的loss求和

对比loss计算

同时刻

$$\ell_{temp}^{(i,t)} = -\log \frac{\exp(r_{i,t} \cdot r'_{i,t})}{\sum_{t' \in \Omega} \left(\exp(r_{i,t} \cdot r'_{i,t'}) + \mathbb{1}_{[t \neq t']} \exp(r_{i,t} \cdot r_{i,t'}) \right)}, \quad (1)$$

Temporal Contrastive loss

同batch

$$\ell_{inst}^{(i,t)} = -\log \frac{\exp(r_{i,t} \cdot r'_{i,t})}{\sum_{j=1}^{B} \left(\exp(r_{i,t} \cdot r'_{j,t}) + \mathbb{1}_{[i \neq j]} \exp(r_{i,t} \cdot r_{j,t}) \right)}, \quad (2)$$

Instance-wise loss

$$\mathcal{L}_{dual} = \frac{1}{NT} \sum_{i} \sum_{t} \left(\ell_{temp}^{(i,t)} + \ell_{inst}^{(i,t)} \right). \quad (3)$$

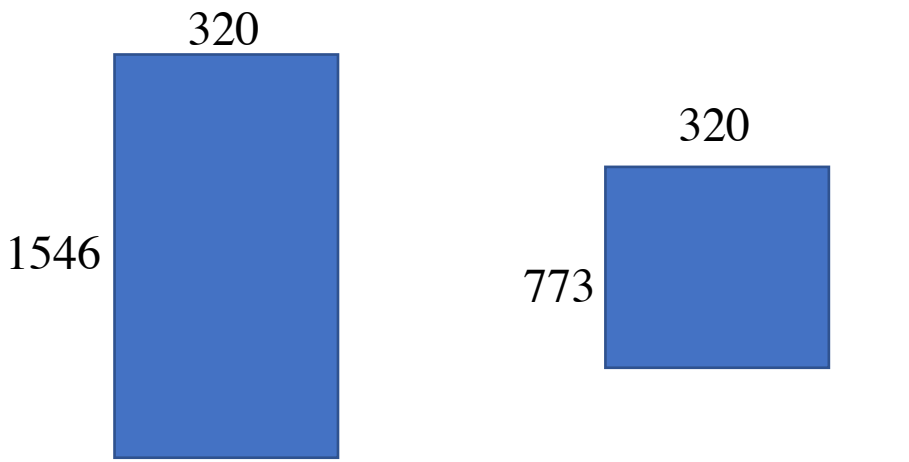
通过上诉方法，我们得到了对应模型的loss，
作者提出了层次对比loss，能够使编码器学习不同尺度的特征表示

Algorithm 1: Calculating the hierarchical contrastive loss

procedure HIERLOSS(r, r')
  $\mathcal{L}_{hier} \leftarrow \mathcal{L}_{dual}(r, r')$;
  $d \leftarrow 1$;
  while time_length(r) > 1 **do** 直到时间维度压缩到一维
    // The maxpool1d operates along the time axis.
    $r \leftarrow \text{maxpool1d}(r, \text{kernel\_size} = 2)$; 对时间轴进行压缩
    $r' \leftarrow \text{maxpool1d}(r', \text{kernel\_size} = 2)$;
    $\mathcal{L}_{hier} \leftarrow \mathcal{L}_{hier} + \mathcal{L}_{dual}(r, r')$;
    $d \leftarrow d + 1$;
  end while
  $\mathcal{L}_{hier} \leftarrow \mathcal{L}_{hier}/d$;
  return $\mathcal{L}_{hier}$
end procedure

$$\mathcal{L}_{dual} = \frac{1}{NT} \sum_i \sum_t \left(\ell_{temp}^{(i,t)} + \ell_{inst}^{(i,t)} \right). \quad (3)$$

$maxpooling: t_1 \sim t_2$
$maxpooling: t_3 \sim t_4$



[8, 1547, 320] -> [8, 773, 320]

对时间维度进行pooling，以获得更大的感受野，同时对缺失数据提升鲁棒性

3 Experiments

3个场景：

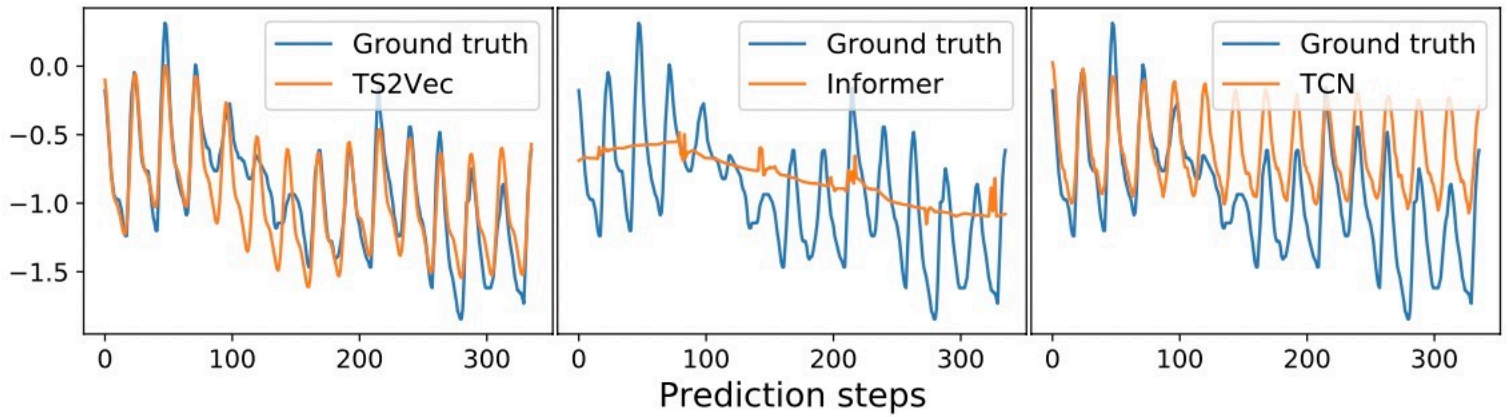
1. time series classification
2. forecasting
3. anomaly detection

3.1 Time Series Classification

	125 UCR datasets			29 UEA datasets		
Method	Avg. Acc.	Avg. Rank	Training Time (hours)	Avg. Acc.	Avg. Rank	Training Time (hours)
DTW	0.727	4.33	–	0.650	3.74	–
TNC	0.761	3.52	228.4	0.677	3.84	91.2
TST	0.641	5.23	17.1	0.635	4.36	28.6
TS-TCC	0.757	3.38	1.1	0.682	3.53	3.6
T-Loss	0.806	2.73	38.0	0.675	3.12	15.1
TS2Vec	**0.830 (+2.4%)**	**1.82**	**0.9**	**0.712 (+3.0%)**	**2.40**	**0.6**

3.2 Time Series Forecasting

Dataset	H	TS2Vec	Informer	LogTrans	N-BEATS	TCN	LSTnet
$ETTh_1$	24	**0.039**	0.098	0.103	0.094	0.075	0.108
	48	**0.062**	0.158	0.167	0.210	0.227	0.175
	168	**0.134**	0.183	0.207	0.232	0.316	0.396
	336	**0.154**	0.222	0.230	0.232	0.306	0.468
	720	**0.163**	0.269	0.273	0.322	0.390	0.659
$ETTh_2$	24	**0.090**	0.093	0.102	0.198	0.103	3.554
	48	**0.124**	0.155	0.169	0.234	0.142	3.190
	168	**0.208**	0.232	0.246	0.331	0.227	2.800
	336	**0.213**	0.263	0.267	0.431	0.296	2.753
	720	**0.214**	0.277	0.303	0.437	0.325	2.878
$ETTm_1$	24	**0.015**	0.030	0.065	0.054	0.041	0.090
	48	**0.027**	0.069	0.078	0.190	0.101	0.179
	96	**0.044**	0.194	0.199	0.183	0.142	0.272
	288	**0.103**	0.401	0.411	0.186	0.318	0.462
	672	**0.156**	0.512	0.598	0.197	0.397	0.639
Electric.	24	0.260	**0.251**	0.528	0.427	0.263	0.281
	48	**0.319**	0.346	0.409	0.551	0.373	0.381
	168	**0.427**	0.544	0.959	0.893	0.609	0.599
	336	**0.565**	0.713	1.079	1.035	0.855	0.823
	720	**0.861**	1.182	1.001	1.548	1.263	1.278
Avg.		**0.209**	0.310	0.370	0.399	0.338	1.099



长时间和周期性时序问题

3.3 Time Series Anomaly Detection

异常检测时，我们训练TS2Vec，测试时候，一次mask最后一个时间(x_t)的序列，一次完整时间序列

Mask

r_t^m [62.9, 66.4, 92.0, 68.4, 116.5, 116.7, 86.3, 64.6, 93.5, 70.3]

r_t^u [62.9, 66.4, 92.0, 68.4, 116.5, 116.7, 86.3, 64.6, 93.5, 70.3]

目标是对整个序列检测是否异常

$$\alpha_t = \|r_t^u - r_t^m\|_1 . \quad (4)$$

为了防止漂移，我们取前面Z点的局部平均值

$$\overline{\alpha}_t = \frac{1}{Z}\sum_{i=t-Z}^{t-1}\alpha_i$$

$$\alpha_t^{adj} = \frac{\alpha_t - \overline{\alpha}_t}{\overline{\alpha}_t}$$

$$\alpha_t^{adj} > \mu + \beta\sigma$$

异常值　　均值和方差

评估t时刻Mask前后的值

Z=3

t3

α_t

[62.9, 66.4, 92.0, 68.4, 116.5, 116.7, 86.3, 64.6, 93.5, 70.3]

$\bar{\alpha}_t$

[nan, nan, 73.8, 75.6, 92.3, 100.6, 106.5, 89.2, 81.5, 76.2]

移动平均

```
In[29]: (62.9+66.4+92.0)/3
Out[29]: 73.76666666666667
```

	Yahoo			KPI		
	F_1	Prec.	Rec.	F_1	Prec.	Rec.
SPOT	0.338	0.269	0.454	0.217	0.786	0.126
DSPOT	0.316	0.241	0.458	0.521	0.623	0.447
DONUT	0.026	0.013	0.825	0.347	0.371	0.326
SR	0.563	0.451	0.747	0.622	0.647	0.598
TS2Vec	**0.745**	0.729	0.762	**0.677**	0.929	0.533
Cold-start:						
FFT	0.291	0.202	0.517	0.538	0.478	0.615
Twitter-AD	0.245	0.166	0.462	0.330	0.411	0.276
Luminol	0.388	0.254	0.818	0.417	0.306	0.650
SR	0.529	0.404	0.765	0.666	0.637	0.697
TS2Vec†	**0.726**	0.692	0.763	**0.676**	0.907	0.540

Table 4: Univariate time series anomaly detection results.

异常检测的效果

removes the temporal contrastive loss

removes the instance-wise contrastive loss

移除层次对比

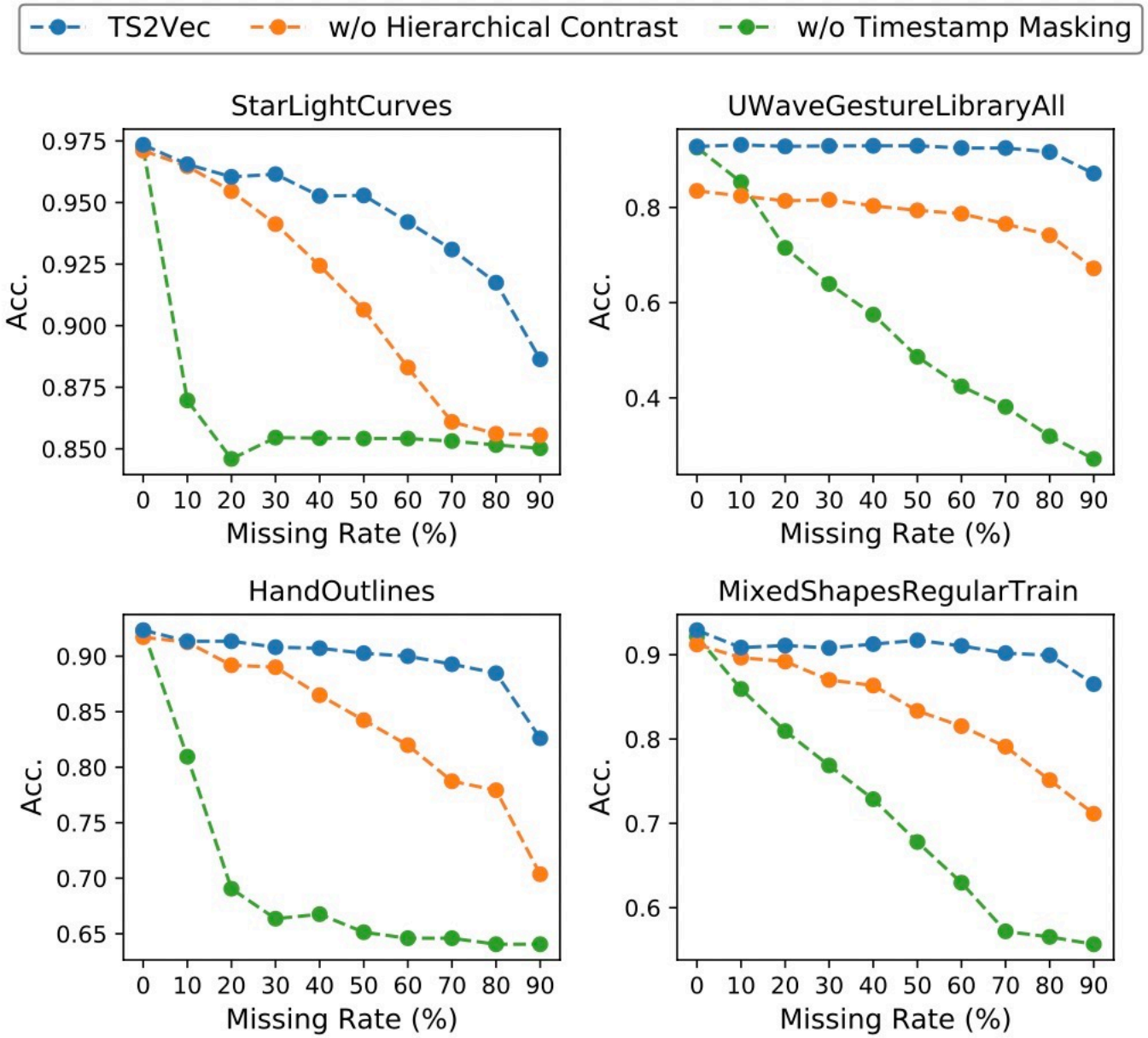
不使用增强采样

不采用mask

移除初始全连接层

	Avg. Accuracy
TS2Vec	**0.829**
w/o Temporal Contrast	0.819 (-1.0%)
w/o Instance Contrast	0.824 (-0.5%)
w/o Hierarchical Contrast	0.812 (-1.7%)
w/o Random Cropping	0.808 (-2.1%)
w/o Timestamp Masking	0.820 (-0.9%)
w/o Input Projection Layer	0.817 (-1.2%)
Positive Pair Selection	
Contextual Consistency	
→ Temporal Consistency	0.807 (-2.2%)
→ Subseries Consistency	0.780 (-4.9%)
Augmentations	
+ Jitter	0.814 (-1.5%)
+ Scaling	0.814 (-1.5%)
+ Permutation	0.796 (-3.3%)
Backbone Architectures	
Dilated CNN	
→ LSTM	0.779 (-5.0%)
→ Transformer	0.647 (-18.2%)

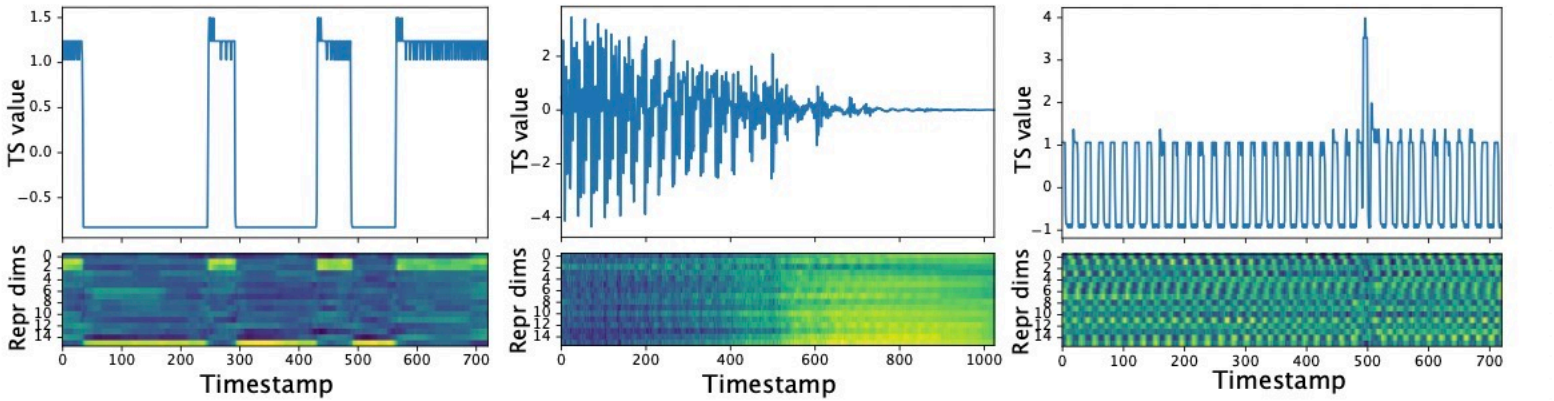
层次方法　　Mask方法



缺失值（train and test）对于效果的影响

指示远程信息对于处理大量缺失值的重要性。

可视化效果

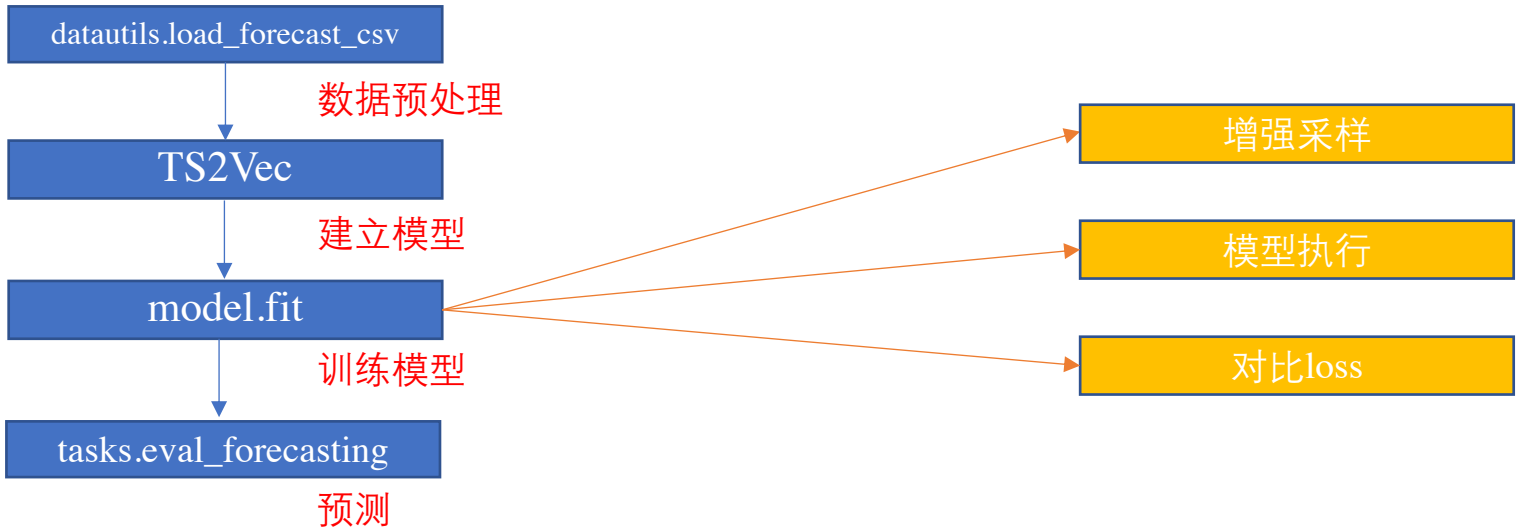


(a) ScreenType. (b) Phoneme. (c) RefrigerationDevices.

区分高低值 时间戳变化趋势 周期性异常值，明显不同

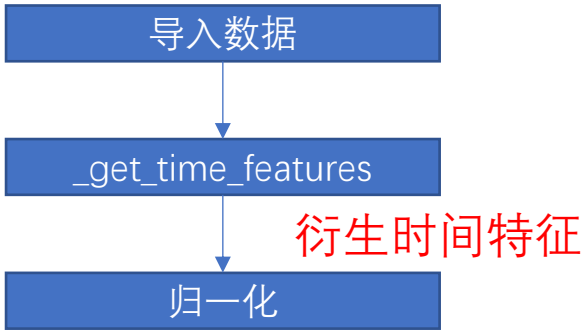
Code

TS2Vec代码



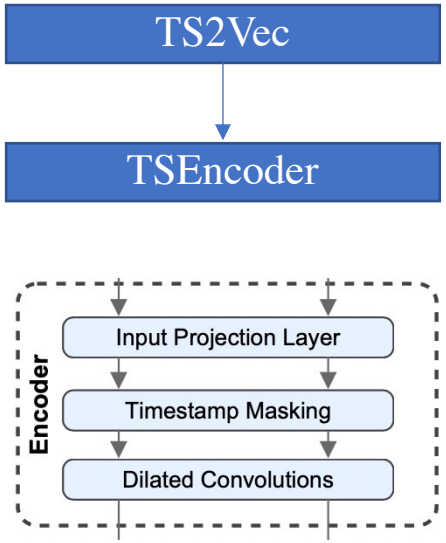
datautils.load_forecast_csv

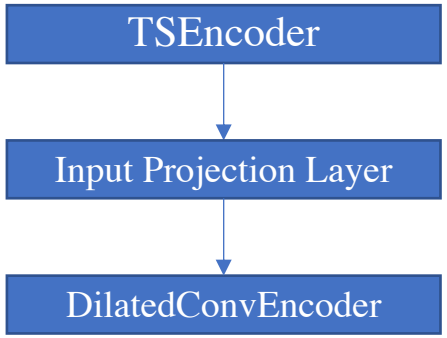
数据预处理模块

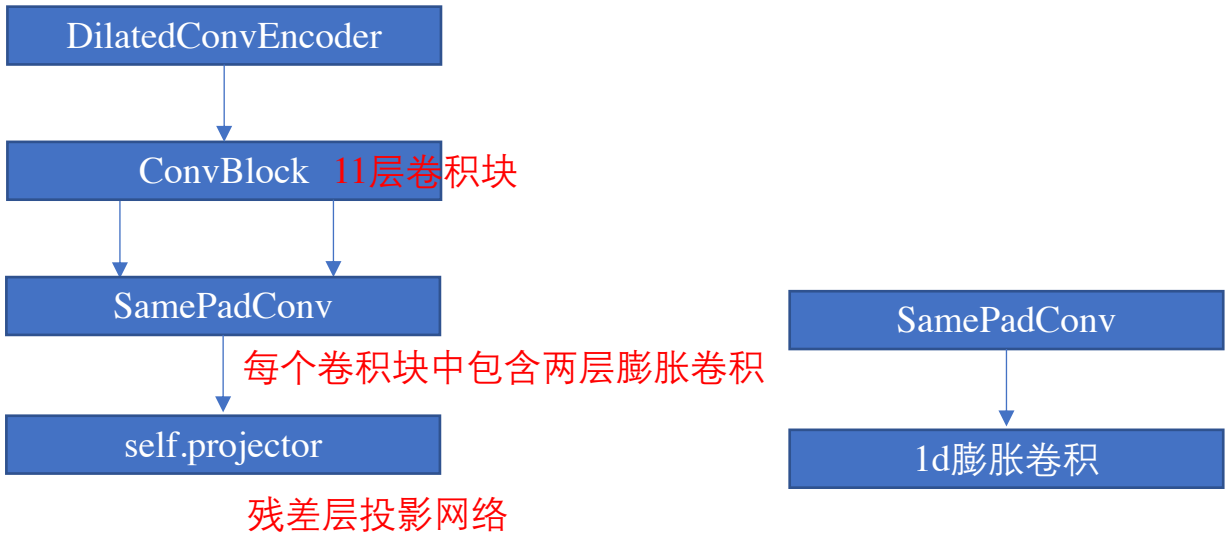


TS2Vec

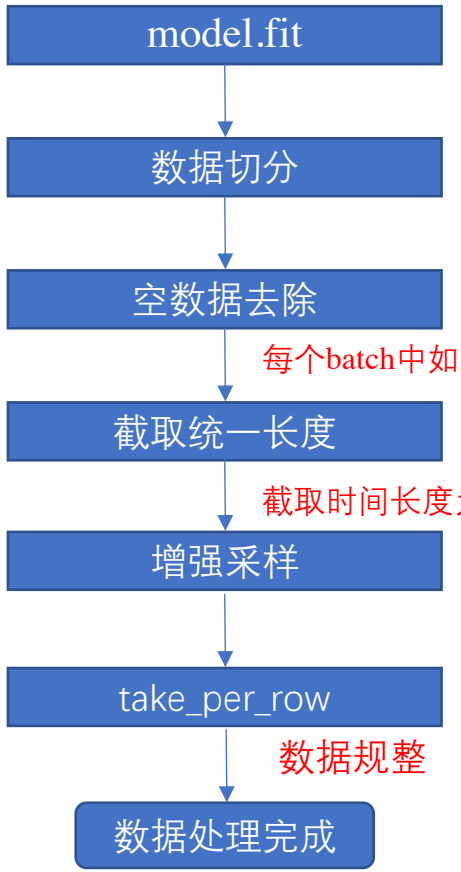
模型结构


TS2Vec
TSEncoder
Encoder
Input Projection Layer
Timestamp Masking
Dilated Convolutions


TSEncoder
Input Projection Layer
DilatedConvEncoder

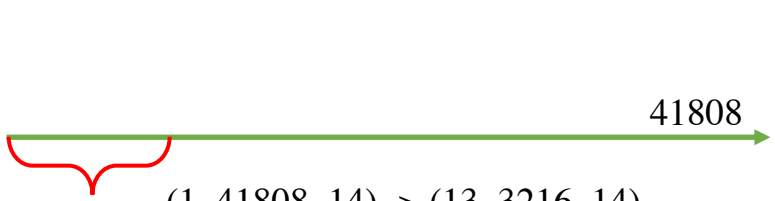

DilatedConvEncoder
ConvBlock 11层卷积块
SamePadConv
每个卷积块中包含两层膨胀卷积
self.projector
残差层投影网络
SamePadConv
1d膨胀卷积

model.fit 模型训练

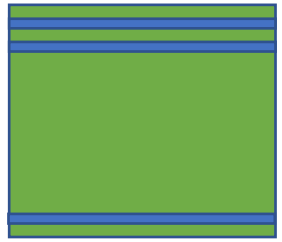


[8, 2695, 14]

8个batch，2695个时间序列，14个特征



13个batch



```
In[10]: indx[:,None]
Out[10]:
array([[192],
       [228],
       [110],
       [212],
       [101],
       [  2],
       [  0],
       [ 58]])
```

+

```
In[11]: np.arange(num_elem)
Out[11]: array([   0,    1,    2, ..., 2692, 2693, 2694])
```

```
In[12]: indx[:,None] + np.arange(num_elem)
Out[12]:
array([[ 192,  193,  194, ..., 2884, 2885, 2886],
       [ 228,  229,  230, ..., 2920, 2921, 2922],
       [ 110,  111,  112, ..., 2802, 2803, 2804],
       ...,
       [   2,    3,    4, ..., 2694, 2695, 2696],
       [   0,    1,    2, ..., 2692, 2693, 2694],
       [  58,   59,   60, ..., 2750, 2751, 2752]])
```

增强采样

时间序列长度

增强采样

```
ts_l = x.size(1)  # 训练时间维度  ts_l: 3000
crop_l = np.random.randint(low=2 ** (self.temporal_unit + 1), high=ts_l+1)  # 随机选择一个crop长度
crop_left = np.random.randint(ts_l - crop_l + 1)  crop_left: 226
crop_right = crop_left + crop_l
crop_eleft = np.random.randint(crop_left + 1)
crop_eright = np.random.randint(low=crop_right, high=ts_l + 1)
crop_offset = np.random.randint(low=-crop_eleft, high=ts_l - crop_eright + 1, size=x.size(0))
```

增强采样

随机选择$crop_l$

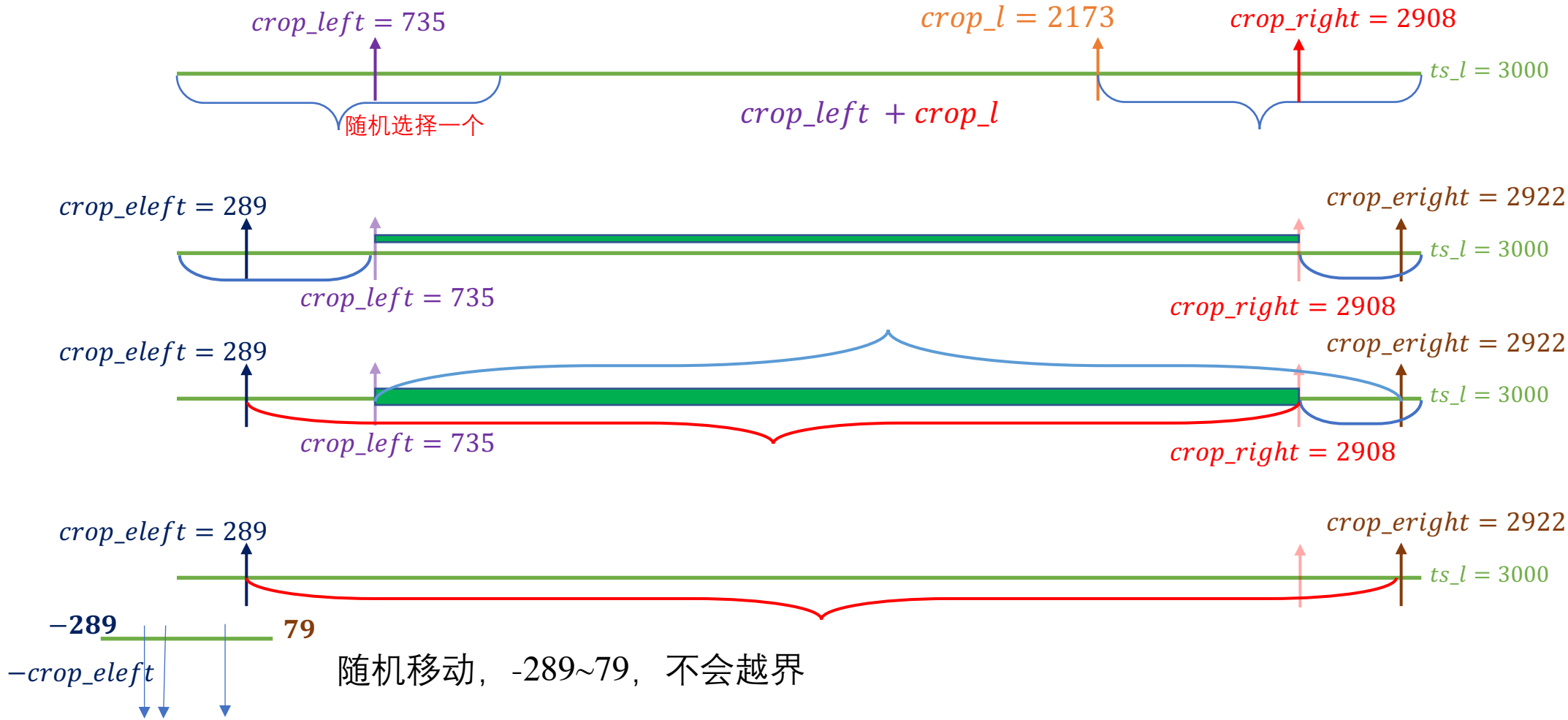
随机选择$crop_left$

计算$crop_right$

随机选择$crop_eleft$

随机选择$crop_eright$

随机选择offset

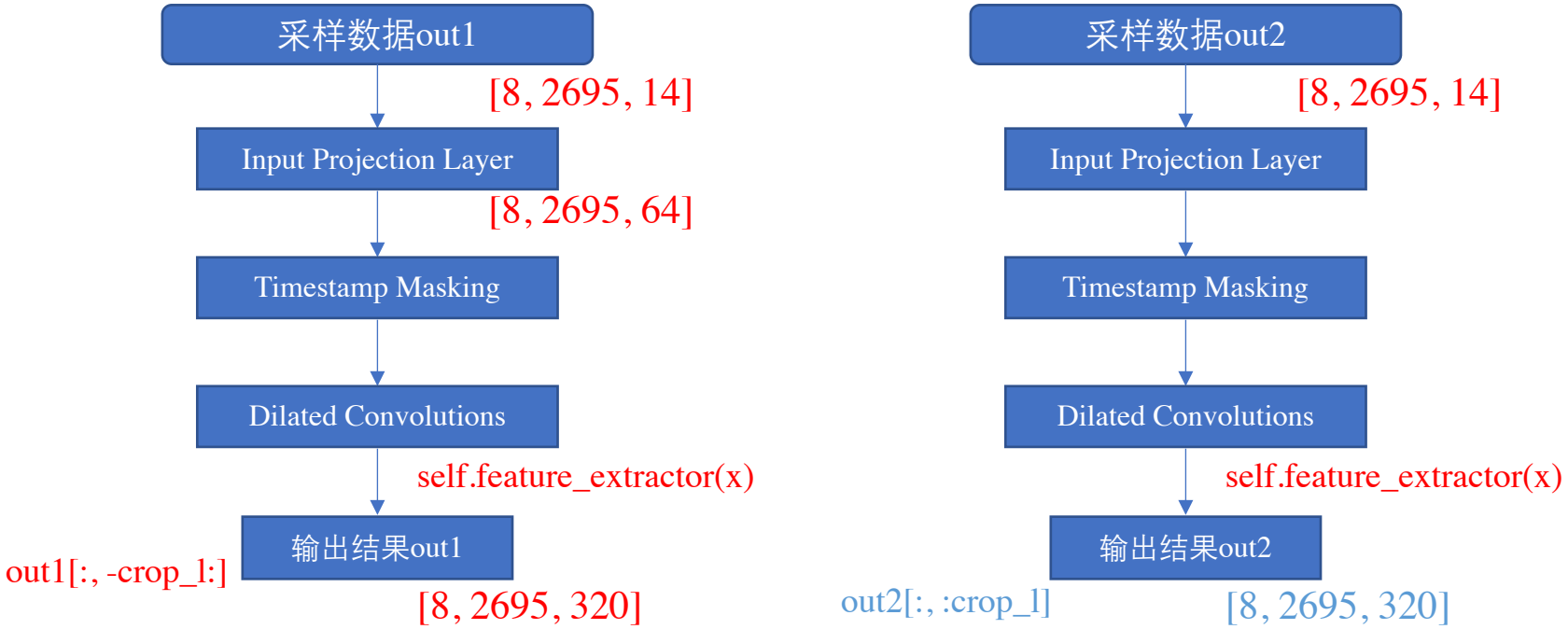


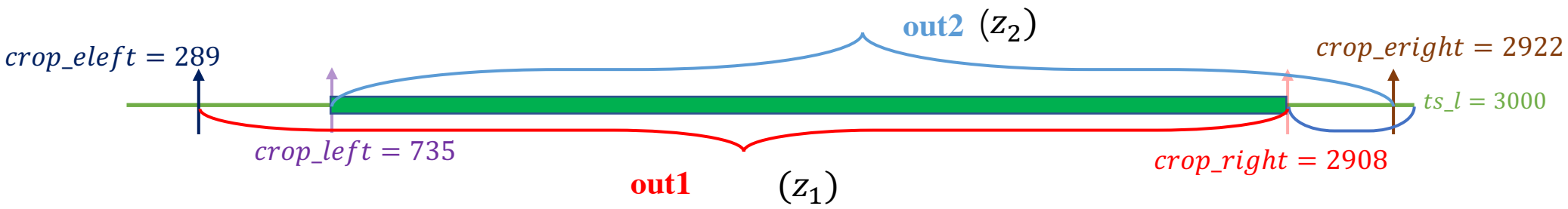
随机移动，-289~79，不会越界

```
In[54]: x[0]
Out[54]:                f1          f2                                  f64
tensor([[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000], t1
        [ 0.1555,  0.4657,  0.5852,  ..., -0.3086, -0.3833, -0.5130], t2
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        ...,
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]], t1847
       grad_fn=<SelectBackward>)
In[55]: x[0].shape
Out[55]: torch.Size([1847, 64])
```

Mask掉某些时刻

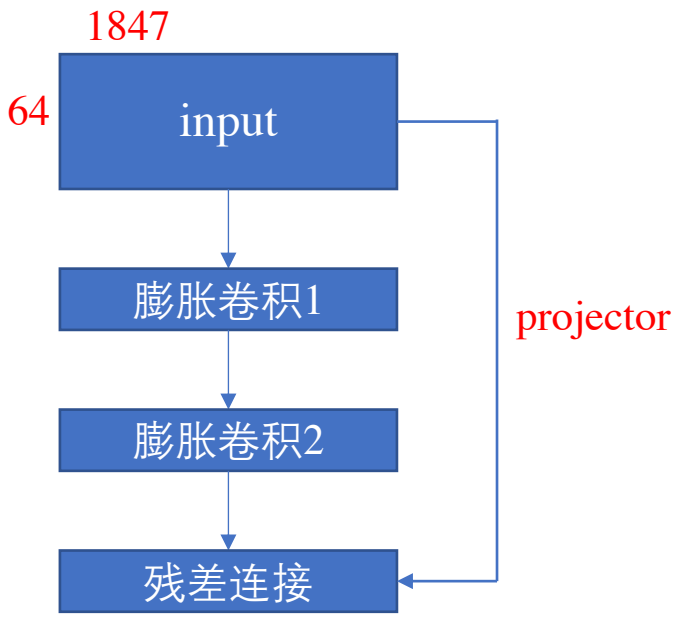
model.fit

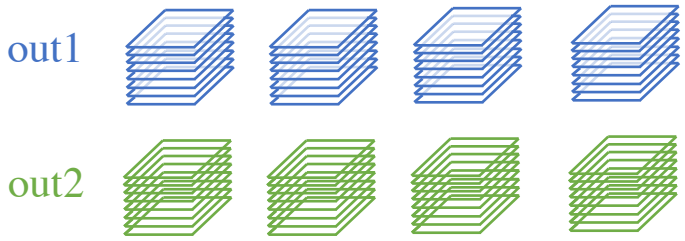

采样数据out1
[8, 2695, 14]
Input Projection Layer
[8, 2695, 64]
Timestamp Masking
Dilated Convolutions
self.feature_extractor(x)
输出结果out1
out1[:, -crop_l:]
[8, 2695, 320]
采样数据out2
[8, 2695, 14]
Input Projection Layer
Timestamp Masking
Dilated Convolutions
self.feature_extractor(x)
输出结果out2
out2[:, :crop_l]
[8, 2695, 320]

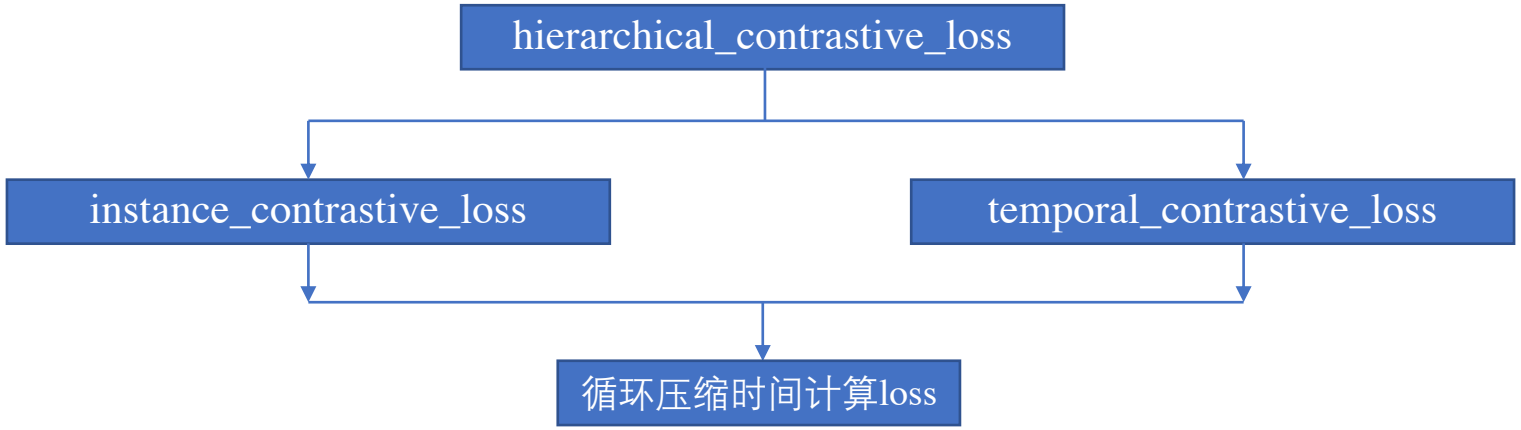

out2 (z_2)
$crop\_eleft = 289$
$crop\_eright = 2922$
$ts\_l = 3000$
$crop\_left = 735$
$crop\_right = 2908$
out1
(z_1)

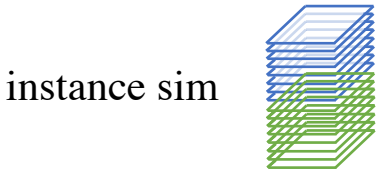
[8, 64, 1847]

```
def forward(self, x):  self:    x: tensor([[[-8.7077e-02, -3.6103e
    residual = x if self.projector is None else self.projector(x)
    x = F.gelu(x)
    x = self.conv1(x)  # 卷积(膨胀)
    x = F.gelu(x)
    x = self.conv2(x)
    return x + residual  # 残差连接
```

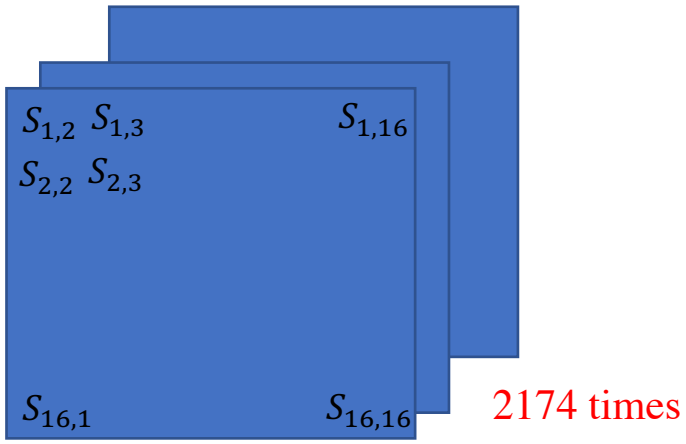




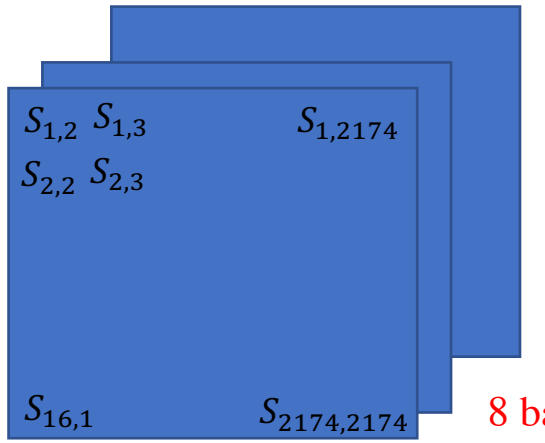




temporal sim



2174 times



8 batch

out1中的第一个batch和out2中的第一个batch是同一组增强数据

$S_{1,9}$表示的是同一段时间序列

Instance-wise Contrastive Loss

[8, 2173, 320]

```
loss += alpha * instance_contrastive_loss(z1, z2)
```

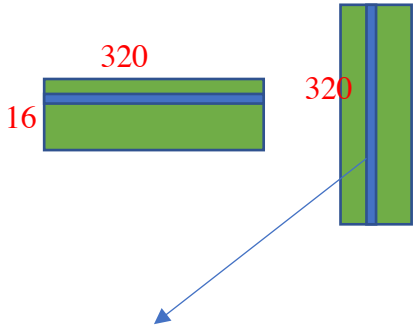
8个batch, 2173个times, 320features

[16, 2173, 320]

```
z = torch.cat([z1, z2], dim=0)  # 2B x T x C  z: tensor([[[-0.1414, -0
z = z.transpose(0, 1)  # T x 2B x C
sim = torch.matmul(z, z.transpose(1, 2))  # T x 2B x 2B
logits = torch.tril(sim, diagonal=-1)[:, :, :-1]    # T x 2B x (2B-1)
logits += torch.triu(sim, diagonal=1)[:, :, 1:]
logits = -F.log_softmax(logits, dim=-1)
```

[2173, 16, 320] **2173个时间，16个batch**

16个batch之间的相似度



$$\ell_{inst}^{(i,t)} = -\log \frac{\exp(r_{i,t} \cdot r'_{i,t})}{\sum_{j=1}^{B} \left(\exp(r_{i,t} \cdot r'_{j,t}) + \mathbb{1}_{[i \neq j]} \exp(r_{i,t} \cdot r_{j,t}) \right)}, \quad (2)$$

这个时间(一共2173时间间隔)，16个batch(16个样本)，每个样本在这个时间320维特征

下三角矩阵

```
In[21]: torch.tril(sim, diagonal=-1)[0,:5,:5]
Out[21]:
tensor([[  0.0000,   0.0000,   0.0000,   0.0000,   0.0000],
        [-12.0627,   0.0000,   0.0000,   0.0000,   0.0000],
        [ 13.7048,  13.3717,   0.0000,   0.0000,   0.0000],
        [  3.8679,   4.2758,   6.9622,   0.0000,   0.0000],
        [ 29.2216,  -8.2743,  19.2786,   6.7525,   0.0000]],
```

上三角矩阵

```
In[22]: torch.triu(sim, diagonal=1)[0,:5,:5]
Out[22]:
tensor([[  0.0000, -12.0627,  13.7048,   3.8679,  29.2216],
        [  0.0000,   0.0000,  13.3717,   4.2758,  -8.2743],
        [  0.0000,   0.0000,   0.0000,   6.9622,  19.2786],
        [  0.0000,   0.0000,   0.0000,   0.0000,   6.7525],
        [  0.0000,   0.0000,   0.0000,   0.0000,   0.0000]],
```

假设矩阵维度为4

out1

```
In[21]: logits = torch.tril(sim, diagonal=-1)[:, :, :-1]
Out[21]:
tensor([[[  0.0000,   0.0000,   0.0000,   0.0000,   0.0000],
         [-12.0627,   0.0000,   0.0000,   0.0000,   0.0000],
         [ 13.7048,  13.3717,   0.0000,   0.0000,   0.0000],
         [  3.8679,   4.2758,   6.9622,   0.0000,   0.0000],
         [ 29.2216,  -8.2743,  19.2786,   6.7525,   0.0000]],
```

out2

```
In[94]: torch.triu(sim, diagonal=1)[:, :, 1:]
Out[22]:
tensor([[[ 0.0000, -12.0627,  13.7048,   3.8679,  29.2216],
         [ 0.0000,   0.0000,  13.3717,   4.2758,  -8.2743],
         [ 0.0000,   0.0000,   0.0000,   6.9622,  19.2786],
         [ 0.0000,   0.0000,   0.0000,   0.0000,   6.7525],
         [  0.0000,   0.0000,   0.0000,   0.0000,   0.0000]],
```

15 16

0

3.4 0

15

16

```
In[24]: logits[0,:4,:3]
Out[24]:
tensor([[-12.0627,  13.7048,   3.8679],
        [-12.0627,  13.3717,   4.2758],
        [ 13.7048,  13.3717,   6.9622],
        [  3.8679,   4.2758,   6.9622]], grad_fn=<SliceBackward>)
```

s(1,2) s(1,3) s(1,4)

s(2,1) s(2,3) s(2,4)

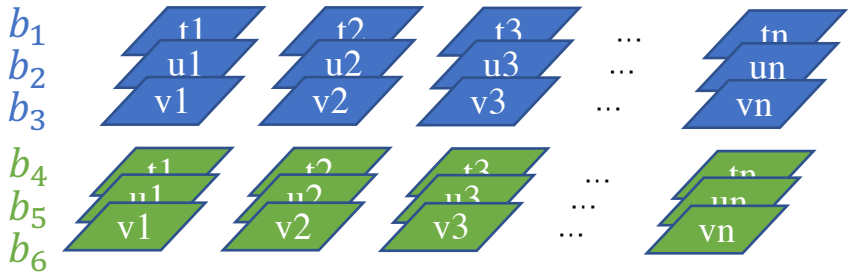
s(3,1) s(3,2) s(3,4)

s(4,1) s(4,2) s(4,3)

假设矩阵维度为6

```
In[24]: logits[0,:4,:3]
Out[24]:
tensor([[-12.0627,  13.7048,   3.8679],
        [-12.0627,  13.3717,   4.2758],
        [ 13.7048,  13.3717,   6.9622],
        [  3.8679,   4.2758,   6.9622]], grad_fn=<SliceBackward>)
```

s(1,2) s(1,3) s(1,4) s(2,1) s(2,3) s(2,4) s(3,1) s(3,2) s(3,4) s(4,1) s(4,2) s(4,3)



$$\ell_{inst}^{(i,t)} = -\log \frac{\exp(r_{i,t}\cdot r'_{i,t})}{\sum_{j=1}^{B}\left(\exp(r_{i,t}\cdot r'_{j,t})+\mathbb{1}_{[i\neq j]}\exp(r_{i,t}\cdot r_{j,t})\right)}, \quad (2)$$

Batch_1={1, 2, 3} Batch_2={4, 5, 6}

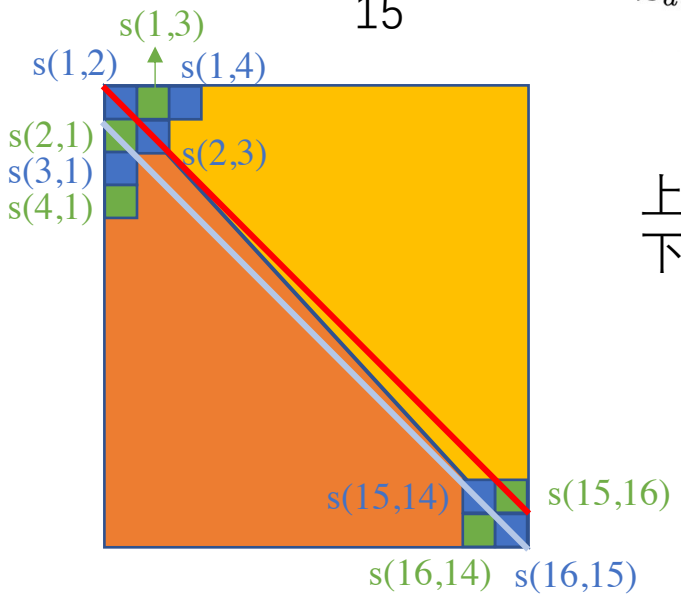
out2

s(1,2)	s(1,3)	s(1,4)	s(1,5)	s(1,6)
s(2,1)	s(2,3)	s(2,4)	s(2,5)	s(2,6)
s(3,1)	s(3,2)	s(3,4)	s(3,5)	s(3,6)
s(4,1)	s(4,2)	s(4,3)	s(4,5)	s(4,6)
s(5,1)	s(5,2)	s(5,3)	s(5,4)	s(5,6)
s(6,1)	s(6,2)	s(6,3)	s(6,4)	s(6,5)

out1

15

$$\mathcal{L}_{dual} = \frac{1}{NT}\sum_{i}\sum_{t}\left(\ell_{temp}^{(i,t)} + \ell_{inst}^{(i,t)}\right). \quad (3)$$



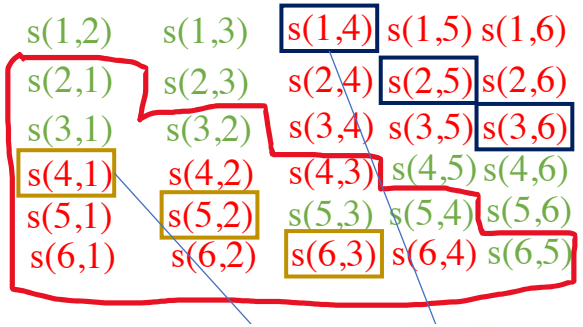
上三角：行和列+1比较（包括对角线）
下三角：行和列比较

out2

s(1,2)	s(1,3)	**s(1,4)**	s(1,5)	s(1,6)
s(2,1)	s(2,3)	s(2,4)	**s(2,5)**	s(2,6)
s(3,1)	s(3,2)	s(3,4)	s(3,5)	**s(3,6)**
s(4,1)	s(4,2)	s(4,3)	s(4,5)	s(4,6)
s(5,1)	**s(5,2)**	s(5,3)	s(5,4)	s(5,6)
s(6,1)	s(6,2)	**s(6,3)**	s(6,4)	s(6,5)

out1

红色为正样本，其余为负样本



B=3

i = [0, 1, 2]

B + i – 1 = [2, 3, 4]

B + i = [3, 4, 5]

只有batch1和batch4才是同一个时间的Out1和out2

假设B=3 i=[0, 1, 2]

```
i = torch.arange(B, device=z1.device)
loss = (logits[:, i, B + i - 1].mean() + logits[:, B + i, i].mean()) / 2
```

[0, 1, 2]
[2, 3, 4]

[3, 4, 5]
[0, 1, 2]

同batch

不同的batch之间相互相关性

$$\ell_{inst}^{(i,t)} = -\log \frac{\exp(r_{i,t} \cdot r'_{i,t})}{\sum_{j=1}^{B} \left(\exp(r_{i,t} \cdot r'_{j,t}) + \mathbb{1}_{[i \neq j]} \exp(r_{i,t} \cdot r_{j,t}) \right)}, \quad (2)$$

batch

不同bathc

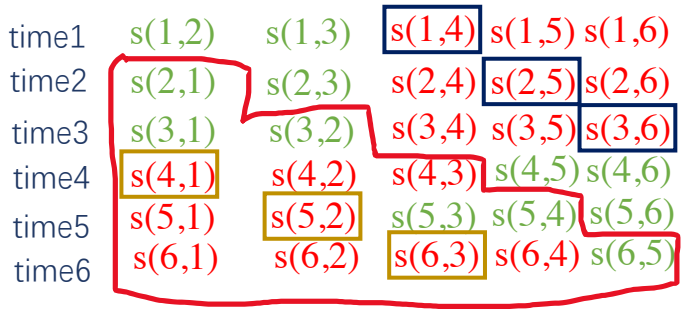
$$\mathcal{L}_{dual} = \frac{1}{NT} \sum_{i} \sum_{t} \left(\ell_{temp}^{(i,t)} + \ell_{inst}^{(i,t)} \right). \quad (3)$$

所有batch和时间求和

Temporal Contrastive Loss

$$\ell_{temp}^{(i,t)} = -\log \frac{\exp(r_{i,t} \cdot r'_{i,t})}{\sum_{t' \in \Omega} \left(\exp(r_{i,t} \cdot r'_{i,t'}) + \mathbb{1}_{[t \neq t']} \exp(r_{i,t} \cdot r_{i,t'}) \right)}, \quad (1)$$

所有的时间




```
def temporal_contrastive_loss(z1, z2):  z1: tensor([[[-1.4141e-01, -1.6659e-0
    B, T = z1.size(0), z1.size(1)  B: 8  T: 2173
    if T == 1:
        return z1.new_tensor(0.)
    z = torch.cat([z1, z2], dim=1)  # B x 2T x C; z:[8, 4346, 320]; z1:[8, 21
    sim = torch.matmul(z, z.transpose(1, 2))  # B x 2T x 2T; channel*channel;
    logits = torch.tril(sim, diagonal=-1)[:, :, :-1]    # B x 2T x (2T-1)  log
    logits += torch.triu(sim, diagonal=1)[:, :, 1:]
    logits = -F.log_softmax(logits, dim=-1)

    t = torch.arange(T, device=z1.device)  t: tensor([   0,    1,    2,  ...,
    loss = (logits[:, t, T + t - 1].mean() + logits[:, T + t, t].mean()) / 2
    return loss
```

循环压缩时间计算loss

```
z1 = F.max_pool1d(z1.transpose(1, 2), kernel_size=2).transpose(1, 2)  # max_pooling
z2 = F.max_pool1d(z2.transpose(1, 2), kernel_size=2).transpose(1, 2)
```

[8, 2173, 320] -> [8, 1086, 320]

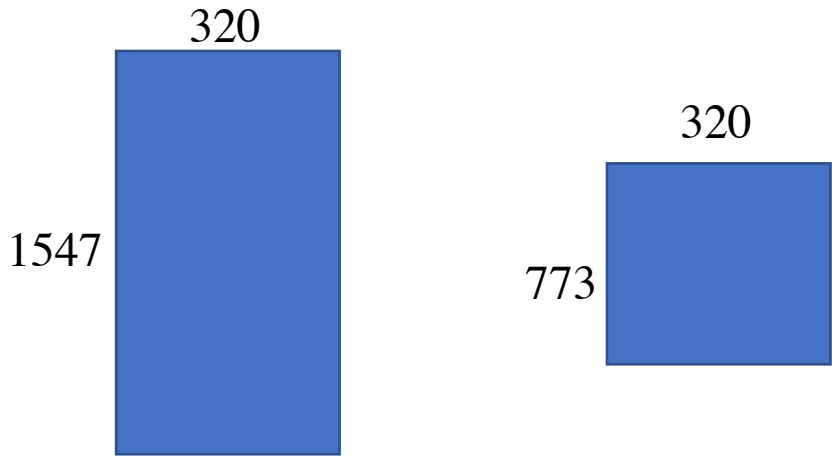
对时间维度进行pooling

Algorithm 1: Calculating the hierarchical contrastive loss

procedure HIERLOSS(r, r')
  $\mathcal{L}_{hier} \leftarrow \mathcal{L}_{dual}(r, r')$;
  $d \leftarrow 1$;
  while time_length(r) > 1 **do**
    // The maxpool1d operates along the time axis.
    $r \leftarrow \text{maxpool1d}(r, \text{kernel\_size} = 2)$;
    $r' \leftarrow \text{maxpool1d}(r', \text{kernel\_size} = 2)$;
    $\mathcal{L}_{hier} \leftarrow \mathcal{L}_{hier} + \mathcal{L}_{dual}(r, r')$;
    $d \leftarrow d + 1$;
  end while
  $\mathcal{L}_{hier} \leftarrow \mathcal{L}_{hier}/d$;
  return $\mathcal{L}_{hier}$
end procedure

$$\mathcal{L}_{dual} = \frac{1}{NT} \sum_i \sum_t \left(\ell_{temp}^{(i,t)} + \ell_{inst}^{(i,t)} \right). \quad (3)$$

对相邻时间进行压缩



[8, 1547, 320] -> [8, 773, 320]

tasks.eval_forecasting

预测代码

时间点t向前推200个时间点

tasks.eval_forecasting

[1, 69680, 14]: batch为1，
69680个时间序列，14个特征

model.encode

测试数据数据预处理

累积到一个batch的数据

self._eval_with_pooling

预测这个batch的时间序列embedding
[256, 1, 320]

torch.split(out, n_samples)

进行时间切分

输出完整数据

[1, 5000, 320]
5000个时间点(每个向前推200个时刻)，每个320维特征

根据特征进行预测

测试数据数据预处理

torch_pad_nan

前后padding

累积batch_size

进来的数据是一个个时间点，
累积到batch数量后，再执行
(如果每个时间点执行也可以)

self._eval_with_pooling

[256, 201, 14]

TS2Vec

[256, 201, 320]

out[:, slicing]

201个时间步，选择最后一个为输出
[256, 1, 320]

根据特征进行预测

使用一个时刻t_0，去预测t_1~t_7时刻的值

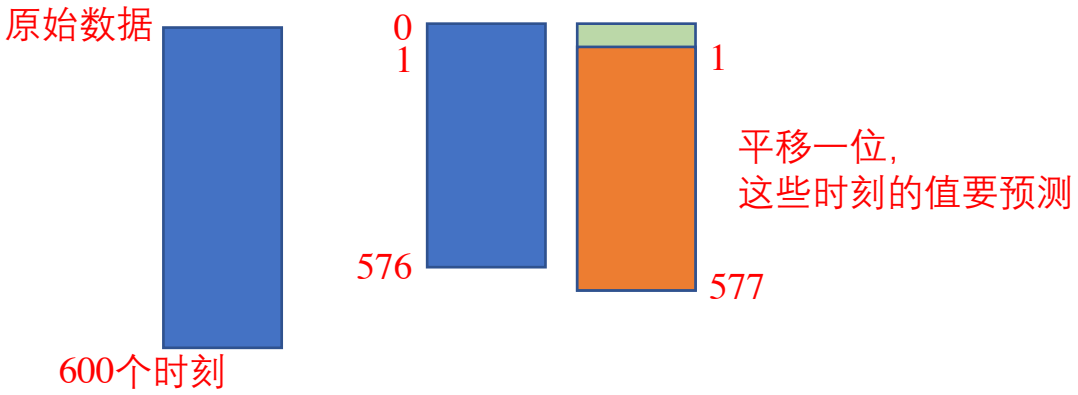
$(1, 576, 24, 7)$

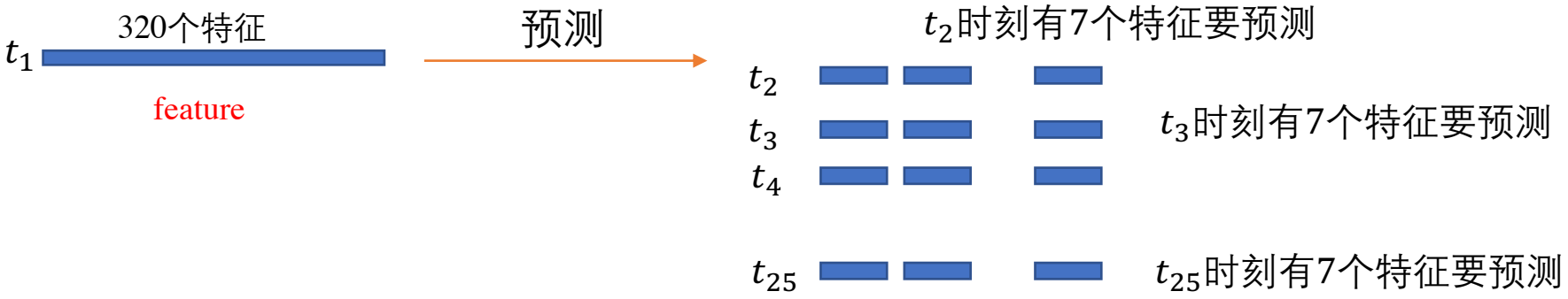
再向下平移一位，表示下一个时刻

```
labels = np.stack([ data[:, i:1+n+i-pred_len] for i in range(pred_len)], axis=2)[:, 1:]
```

```
labels.reshape(-1, labels.shape[2]*labels.shape[3]
```

未来24个时间点，每个时间点7个特征需预测，共168个

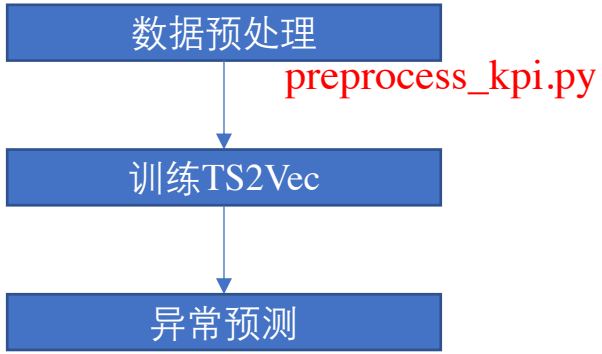


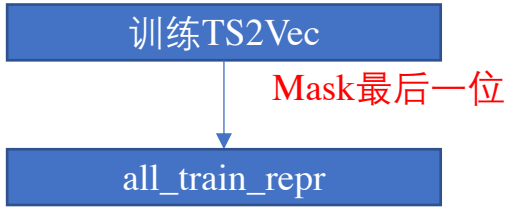


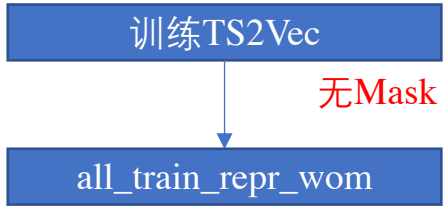
t_1时刻节点特征

预测(回归)未来一段时间内的所有特征

异常检测方法


数据预处理
preprocess_kpi.py
训练TS2Vec
异常预测


训练TS2Vec
Mask最后一位
all_train_repr


训练TS2Vec
无Mask
all_train_repr_wom

```
train_err = np.abs(all_train_repr_wom[k] - all_train_repr[k]).sum(axis=1)  # 非mask 和 mask_last; 训练误差
test_err = np.abs(all_test_repr_wom[k] - all_test_repr[k]).sum(axis=1)  # 测试误差  test_err: [ 71.36469
```

$$\alpha_t = \|r_t^u - r_t^m\|_1 . \qquad (4)$$

(4391, 320): 按行求和

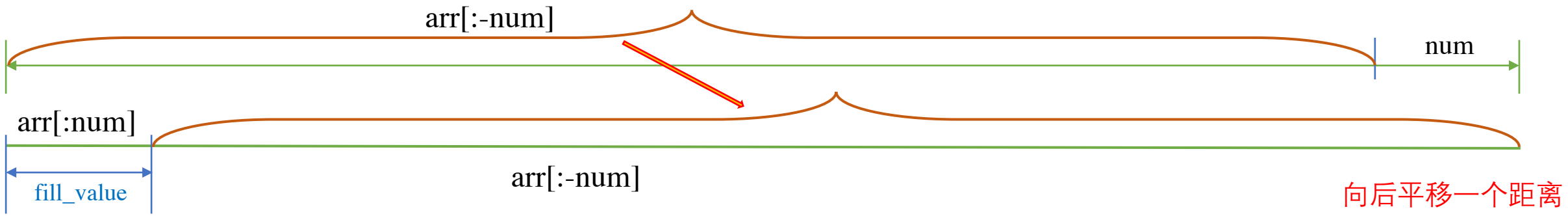
```
ma = np_shift(bn.move_mean(np.concatenate([train_err, test_err]), 21), 1)  # 合并训练测试误差; 计算移动平均
```

```
>>> a = np.array([1.0, 2.0, 3.0, np.nan, 5.0])
>>> bn.move_mean(a, window=2)  移动平均
array([ nan,  1.5,  2.5,  nan,  nan])
>>> bn.move_mean(a, window=2, min_count=1)
array([ 1. ,  1.5,  2.5,  3. ,  5. ])
```

窗口为21，合并误差，得出均值

this transferred version of our model as TS2Vec†. We set $\beta = 4$ empirically and $Z = 21$ following (Ren et al. 2019)

```
def np_shift(arr, num, fill_value=np.nan):
    result = np.empty_like(arr)  # 极小的数，填充空值
    if num > 0:  #                向后平移一个距离
        result[:num] = fill_value
        result[num:] = arr[:-num]
    elif num < 0:
        result[num:] = fill_value
        result[:num] = arr[-num:]
    else:
        result[:] = arr
    return result
```



```
train_err = np.abs(all_train_repr_wom[k] - all_train_repr[k]).sum(axis=1)  # 非mask 和 mask_last; 训练误差
test_err = np.abs(all_test_repr_wom[k] - all_test_repr[k]).sum(axis=1)  # 测试误差  test_err: [ 71.36469
```

```
ma = np_shift(bn.move_mean(np.concatenate([train_err, test_err]), 21), 1)  # $\overline{\alpha}_t = \frac{1}{Z}\sum_{i=t-Z}^{t-1}\alpha_i$ 均
```

```
train_err_adj = (train_err - ma[:len(train_err)]) / ma[:len(train_err)]  # 原数据和平均移动的误差
test_err_adj = (test_err - ma[len(train_err):]) / ma[len(train_err):]
train_err_adj = train_err_adj[22:]

thr = np.mean(train_err_adj) + 4 * np.std(train_err_adj)
test_res = (test_err_adj > thr) * 1
```

$$\alpha_t^{adj} = \frac{\alpha_t - \overline{\alpha}_t}{\overline{\alpha}_t}$$

$$\overline{\alpha}_t = \frac{1}{Z}\sum_{i=t-Z}^{t-1}\alpha_i$$

前Z位的移动平均

$$\alpha_t^{adj} = \frac{\alpha_t - \overline{\alpha}_t}{\overline{\alpha}_t}$$

$$\alpha_t^{adj} > \mu + \beta\sigma$$

```
In[29]: (62.9+66.4+92.0)/3
Out[29]: 73.76666666666667
```

[62.9, 66.4, 92.0, 68.4, 116.5, 116.7, 86.3, 64.6, 93.5, 70.3]

[nan, nan, 73.8, 75.6, 92.3, 100.6, 106.5, 89.2, 81.5, 76.2]

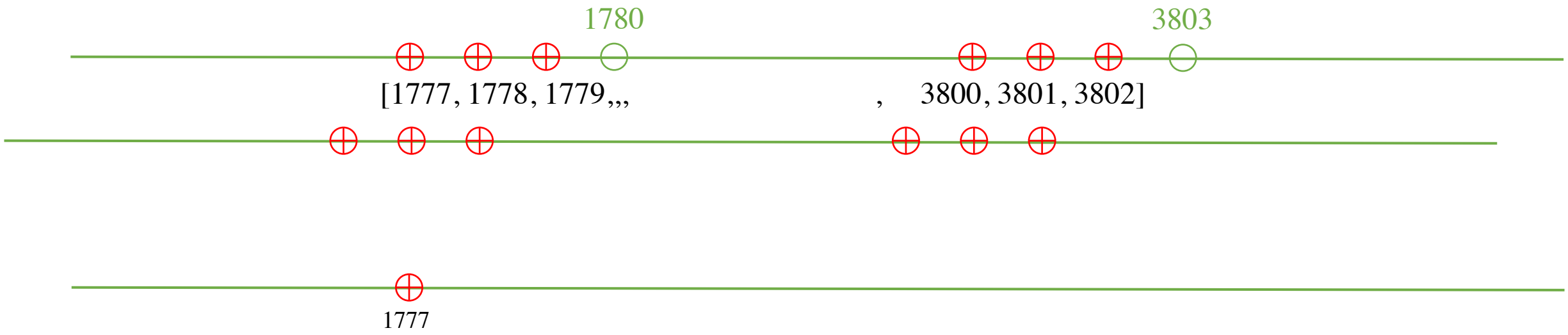
[nan, nan, nan, 73.8, 75.6, 92.3, 100.6, 106.5, 89.2, 81.5]

np_shift: 平移num位

```
def get_range_proba(predict, label, delay=7):  predict: [0 0 0 0 0 0 0 0 0 0 0 0 0 (
    splits = np.where(label[1:] != label[:-1])[0] + 1  # 第一个出现label不同的index
    is_anomaly = label[0] == 1  # 第一个值是否异常  is_anomaly: False
    new_predict = np.array(predict)  new_predict: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (
    pos = 0  pos: 0
```

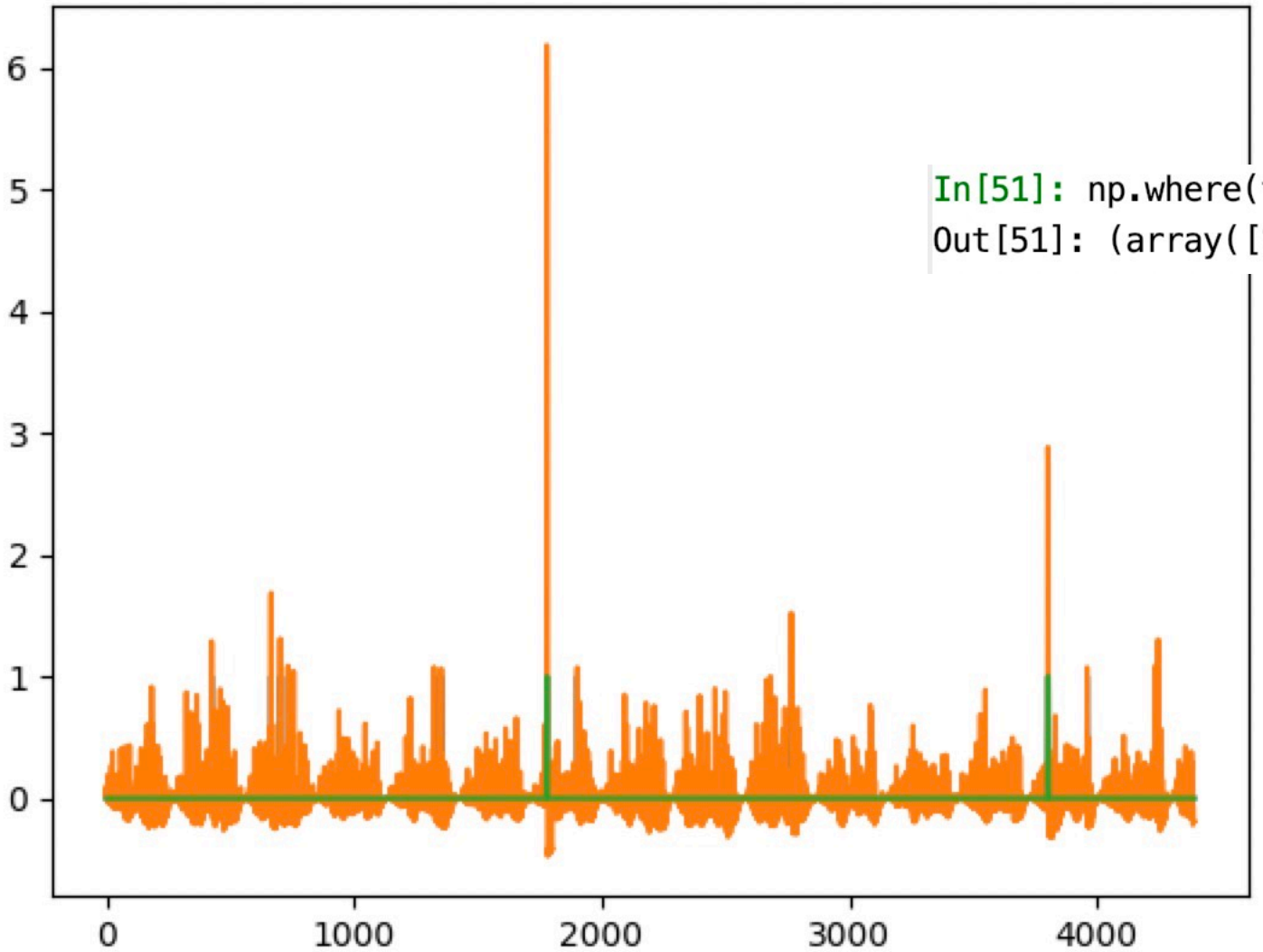
```
In[54]: splits
Out[54]: array([1777, 1780, 3800, 3803])
```

1777开始时，进行label类别转变(之前位正，现在转为负样本)
1780又开始转换，负转成正样本



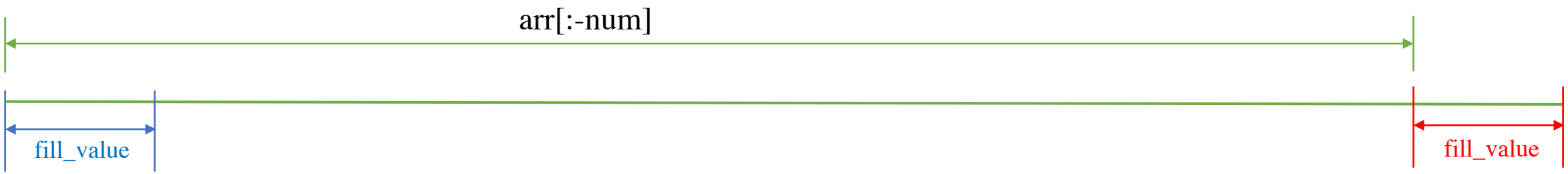

```
for sp in splits:  sp: 1777
    if is_anomaly:
        if 1 in predict[pos:min(pos + delay + 1, sp)]:
            new_predict[pos: sp] = 1
        else:
            new_predict[pos: sp] = 0
    is_anomaly = not is_anomaly
    pos = sp
```

预测结果




```
In[51]: np.where(test_labels==1)
Out[51]: (array([1777, 1778, 1779, 3800, 3801, 3802]),)
```

```python
def np_shift(arr, num, fill_value=np.nan):
    result = np.empty_like(arr)  # 极小的数，填充空值
    if num > 0:  #
        result[:num] = fill_value
        result[num:] = arr[:-num]
    elif num < 0:
        result[num:] = fill_value
        result[:num] = arr[-num:]
    else:
        result[:] = arr
    return result
```



FlowScope: Spotting Money Laundering Based on Graphs

ML(洗钱)有以下几个特征：

1. 转入转出之间形成高密度子图（大量的资金需要在账户上快速转入转出）
2. 中间账户清零（中间账户作为桥梁，增加被发现风险，所以需要将中间账户清零）

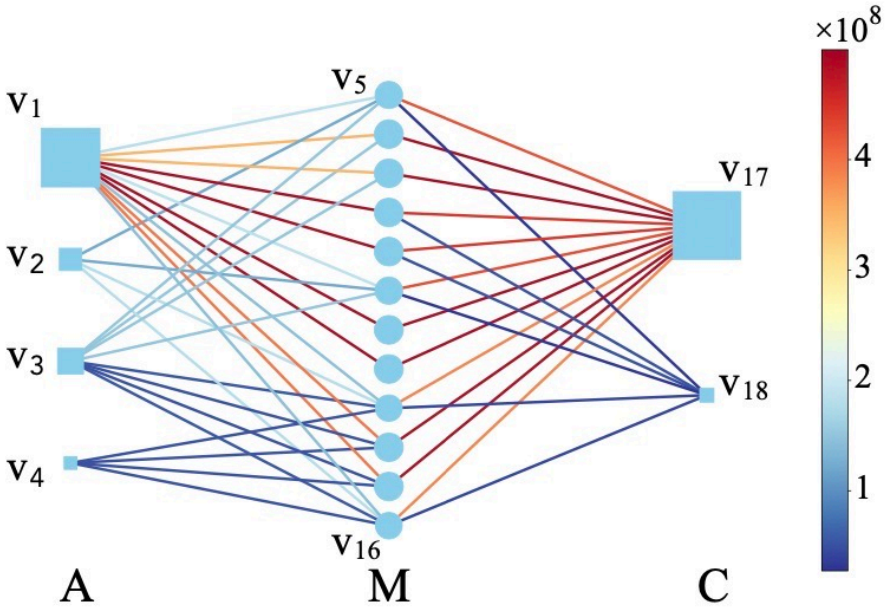
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ $\mathcal{V}$表示账户，$\mathcal{E}$表示边

$\mathcal{V} = \mathcal{X} \cup \mathcal{W} \cup \mathcal{Y}$

x – 起始账户

y – 终止账户

w – 中间账户



典型的洗钱过程

Symbol	Interpretation
$\mathcal{W}$	Inner accounts of the bank
$\mathcal{X}, \mathcal{Y}$	Outer accounts mainly transferring money into or receiving money transfers out of the bank
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Tripartite graph of transfers in the bank
$\mathcal{V}$	Nodes of graph $\mathcal{G}$, i.e. $\mathcal{X} \cup \mathcal{W} \cup \mathcal{Y}$
$\mathcal{S}$	Node subset of graph $\mathcal{G}$, i.e. $\mathcal{A} \cup \mathcal{M}_1 \cup \cdots \cup \mathcal{M}_{k-2} \cup \mathcal{C}$
e_{ij}	Total amount of money on edge (i, j)
d_i, d_i^+, d_i^-	Degree (weight), out-degree, in-degree of node
$f_i(\mathcal{S}), q_i(\mathcal{S})$	Minimum and maximum of node's out-degree and in-degree, $\forall v_i \in \mathcal{M}_l$
w_i	Weight assigned to a node in priority tree
$\mathcal{S}^*$	Optimal subset of nodes maximizing metric
$\hat{\mathcal{S}}$	Subset returned by FlowScope
$\mathcal{S}'$	Subset just before removing node $v* \in \mathcal{S}^*$ in the next step by FlowScope
$g(\mathcal{S})$	Metric of ML anomalousness

$$\mathcal{V}^k = \mathcal{X} \cup \underbrace{\mathcal{W} \cup \cdots \cup \mathcal{W}}_{k-2} \cup \mathcal{Y}$$

将中间账户都简化为W

$$\mathcal{W}_0 = \mathcal{X}, \mathcal{W}_{k-1} = \mathcal{Y}$$

e_{ij} – v_i到v_j之间的转账总和

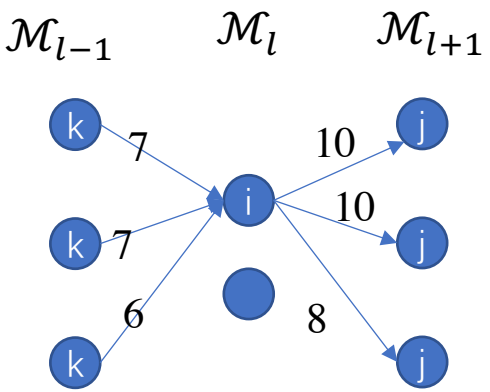
$$v_i \in \mathcal{M}_l, l \in \{1, 2, \cdot, k-2\}$$

$$d_i^+(S) = \sum_{v_j \in \mathcal{M}_{l+1} \wedge (i,j) \in \mathcal{E}} e_{ij}$$

out-degree

$$d_i^-(S) = \sum_{v_k \in \mathcal{M}_{l-1} \wedge (k,i) \in \mathcal{E}} e_{ki}$$

in-degree



$d_i^+(S) = 28$

$d_i^-(S) = 20$

$f_i(S) = \min(28, 20) = 20$

$q_i(S) = max(28, 20) = 28$

$$f_i(\mathcal{S}) = \min\{d_i^+(\mathcal{S}), d_i^-(\mathcal{S})\}, \forall v_i \in \mathcal{M}_l \quad (1)$$

$$q_i(\mathcal{S}) = \max\{d_i^+(\mathcal{S}), d_i^-(\mathcal{S})\}, \forall v_i \in \mathcal{M}_l \quad (2)$$

中间账户层

$$g^k(\mathcal{S}) = \frac{1}{|\mathcal{S}|}\sum_{l=1}^{k-2}\sum_{v_i\in\mathcal{M}_l} f_i(\mathcal{S}) - \lambda(q_i(\mathcal{S}) - f_i(\mathcal{S})) \quad (3)$$

潜在的利益　　层内所有节点　　中间账户剩余(赤字)的钱

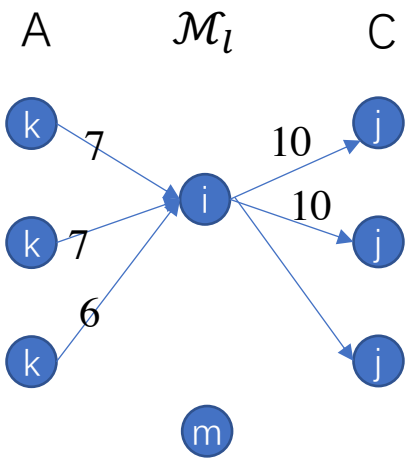
1　　k-2

$$\mathcal{V}^k = \mathcal{X} \cup \underbrace{\mathcal{W} \cup \cdots \cup \mathcal{W}}_{k-2} \cup \mathcal{Y}$$

$\lambda = 1$

$$f_i(\mathcal{S}) - \lambda(q_i(\mathcal{S}) - f_i(\mathcal{S})) = 20 - \lambda(28 - 20) = 16$$

转入转出钱和剩余钱的差值，用来表示中间节点转账的风险度，
如果向外转出的钱多，并且剩余的钱少，则高风险。
（因为欺诈者会倾向于将中间账户的钱清零）



$d_i^+(S) = 28$

$d_i^-(S) = 20$

$f_i(S) = \min(28, 20) = 20$

$q_i(S) = max(28, 20) = 28$

如果中间账户存留钱比较多，可以理解为潜在的利润空间会缩小

14 -> 8

λ越大，表示欺诈者需要承担的伪装代价越多

Algorithm 1: FlowScope

Input: Graph $G = (\mathcal{V}, \mathcal{E})$

Output: Node set of dense tripartite flow: $\hat{\mathcal{S}}$

$\mathcal{A} \leftarrow \mathcal{X}, \mathcal{M}_1 \leftarrow \mathcal{W}, \cdots, \mathcal{M}_{k-2} \leftarrow \mathcal{W}, \mathcal{C} \leftarrow \mathcal{Y}$
// generate k-partite node subsets from $\mathcal{G}$

$\mathcal{S} \leftarrow \mathcal{A} \cup \mathcal{M}_1 \cup \cdots \cup \mathcal{M}_{k-2} \cup \mathcal{C}$

$w_i \leftarrow$ calculate node weight as Eq. (5)

$\mathcal{T} \leftarrow$ build priority tree for $\mathcal{S}$ with $w_i(\mathcal{S})$

while $\mathcal{A}, \mathcal{M}_1, \cdots, \mathcal{M}_{k-2}$ *and* $\mathcal{C}$ *is not empty* **do**

  $v \leftarrow$ find the minimum weighted node in $\mathcal{T}$

  $\mathcal{S} \leftarrow \mathcal{S} \setminus \{v\}$ 找到最小的$w_i(S)$，将其去掉

  update priorities in $\mathcal{T}$ for all neighbors of v

  $g(\mathcal{S}) \leftarrow$ calculate as Eq. (3)

end 目标是最大化g(s)

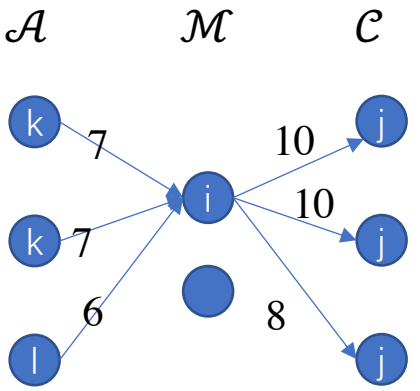
return $\hat{\mathcal{S}}$ that maximizes $g(\mathcal{S})$ seen during the loop.

$$g^k(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{l=1}^{k-2} \sum_{v_i \in \mathcal{M}_l} f_i(\mathcal{S}) - \lambda(q_i(\mathcal{S}) - f_i(\mathcal{S})) \quad (3)$$

$$= \frac{1}{|\mathcal{S}|} \sum_{l=1}^{k-2} \sum_{v_i \in \mathcal{M}_l} (1+\lambda) f_i(\mathcal{S}) - \lambda q_i(\mathcal{S}),\ k \geq 3$$

$$w_i(\mathcal{S}) = \begin{cases} f_i(\mathcal{S}) - \frac{\lambda}{1+\lambda} q_i(\mathcal{S}), & \text{if } v_i \in \mathcal{M}_l \\ d_i(\mathcal{S}), & \text{if } v_i \in \mathcal{A} \cup \mathcal{C} \end{cases} \quad (5)$$

转账增益



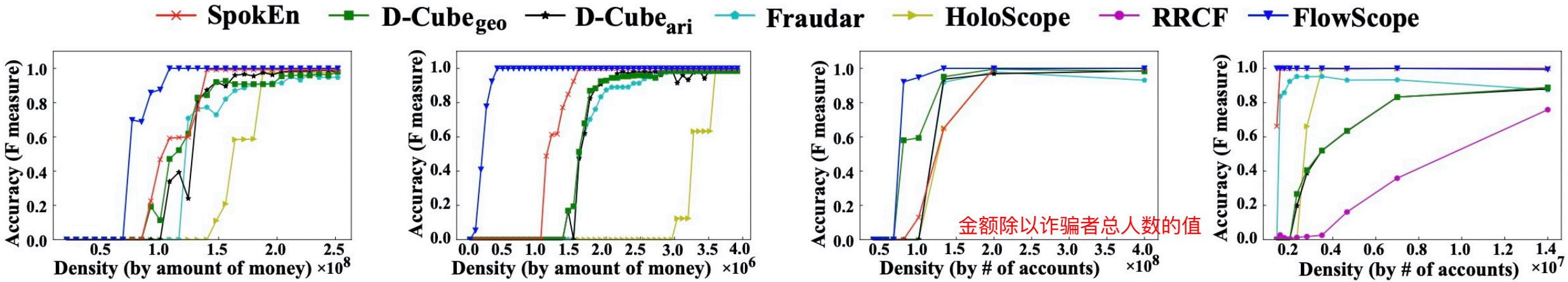
$d_i^+(S) = 28$

$d_i^-(S) = 20$

$f_i(S) = \min(28, 20) = 20$

$q_i(S) = max(28, 20) = 28$

作为中间节点($\mathcal{M}$)	作为起始节点($\mathcal{A}$)	作为终止节点($\mathcal{C}$)
$w_i(S) = 20 - \frac{1}{2} * 28 = 6$	$w_i(S) = 28$	$w_i(S) = 20$
$w_l(S) = 0 - \frac{1}{2} * 6 = -3$	$w_l(S) = 6$	$w_l(S) = 0$



(a) CBank: descending money

(b) CFD: descending money

(c) CBank: ascending # of accounts

(d) CFD: ascending # of accounts

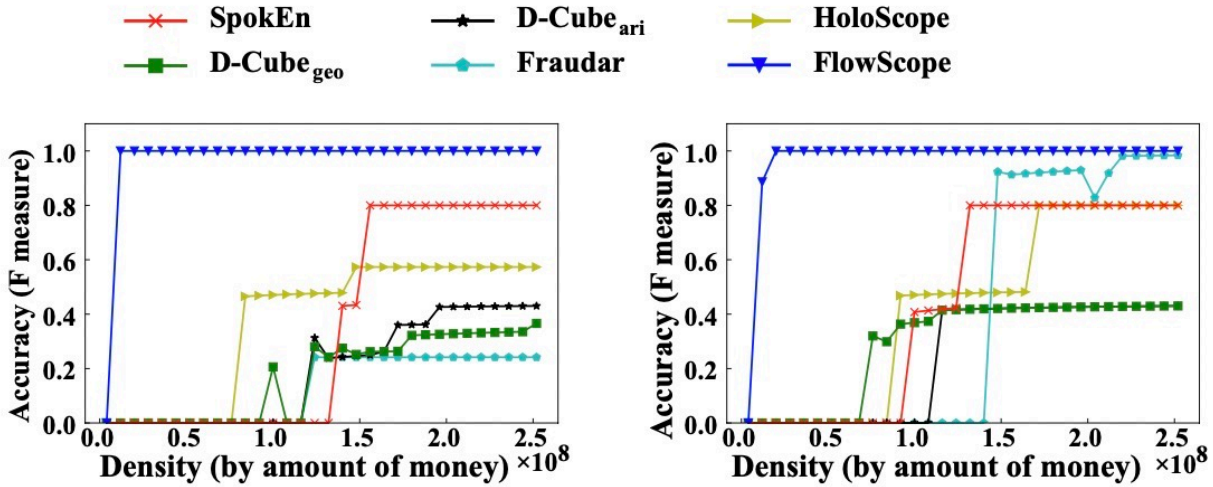
在较少的洗钱情况下，FlowScope就可以侦测

增加用户数量来分赃账款

起始, 中间, 终止账户的比例

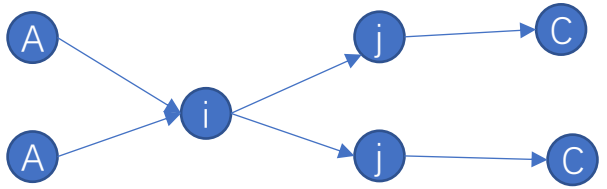
不同数量资金和账户数量

Dataset	metrics*	A:M:C	D-Cube$_{ari}$	D-Cube$_{geo}$	Fraudar	HoloScope	SpokEn	RRCF	FlowScope
CBank	FAUC	5:9:1	0.417 / 0.600	0.591 / 0.810	0.347 / 0.634	0.276 / 0.466	0.610 / 0.753	- / -	**0.633 / 0.800**
		5:5:5	0.502 / 0.658	0.501 / 0.709	0.467 / 0.683	0.379 / 0.655	0.598 / 0.708	- / -	**0.757 / 0.843**
		7:5:3	0.533 / 0.727	0.522 / 0.779	0.529 / 0.704	0.377 / 0.547	0.633 / 0.708	- / -	**0.761 / 0.843**
	F1 $\geq$ 0.9 (million \$ / node size)	5:9:1	190 / 30	- / 45	- / 30	210 / 15	154 / 30	- / -	**132 / 75**
		5:5:5	150 / 45	- / 45	- / 42	- / 30	116 / 45	- / -	**84.0 / 90**
		7:5:3	175 / 30	166 / 54	180 / 33	- / 15	122 / 30	- / -	**76.0 / 90**
CFD	FAUC	5:9:1	0.498 / 0.577	0.528 / 0.577	0.409 / 0.770	0.125 / 0.773	0.716 / **0.894**	0.253 / 0.538	**0.939** / 0.877
		5:5:5	0.565 / 0.633	0.592 / 0.736	0.549 / 0.867	0.143 / 0.810	0.716 / 0.897	0.236 / 0.364	**0.962 / 0.900**
		7:5:3	0.580 / 0.734	0.520 / 0.725	0.593 / 0.826	0.0356 / 0.818	0.728 / 0.898	0.213 / 0.434	**0.970 / 0.900**
	F1 $\geq$ 0.9 (million \$ / node size)	5:9:1	2.21 / 15	2.19 / 15	- / 60	3.52 / 60	1.71 / 120	- / 15	**0.400 / 150**
		5:5:5	1.87 / 30	2.05 / 30	- / 150	- / 75	1.23 / **150**	- / 15	**0.240 / 150**
		7:5:3	- / 30	- / 30	- / 120	- / 60	1.46 / 135	- / 15	**0.240 / 150**



(a) CBank: Transfer chains of (5,5,5,5)

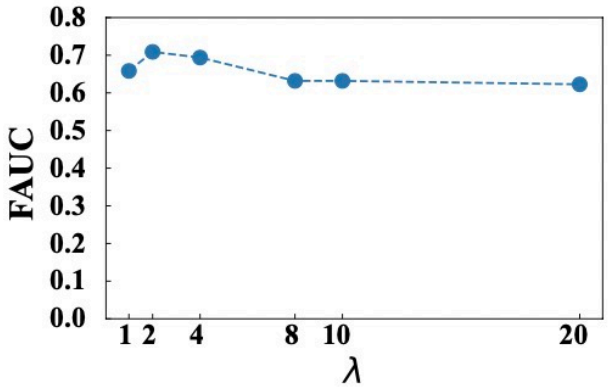
(b) CBank: Transfer chains of (2,8,8,2)

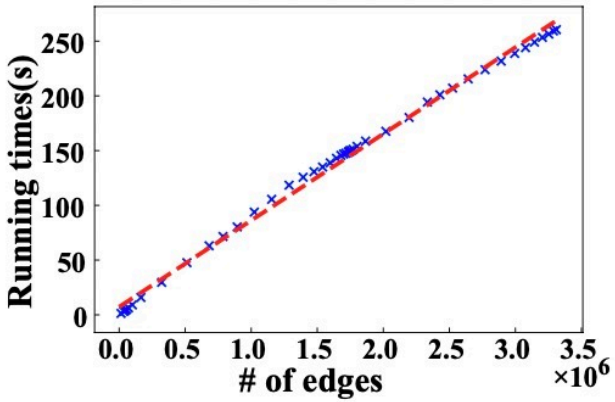


两个中间账户

起始 中间 终止

可以捕获长链路的转账洗钱方式





(a) FlowScope is stable for a reasonable range of λ.

(b) FlowScope scales near linearly

超参数稳定性

拓展性

Code

```
# 定义Wi, di+, di-
bestAveScore = float('-inf')  bestAveScore: -inf
rowDeltas = np.squeeze(M1.sum(axis=1).A)  # 行求和，起始节点wi  r
midDeltas1 = np.squeeze(M1.sum(axis=0).A)  # 列求和，M1节点收款;
midDeltas2 = np.squeeze(M2.sum(axis=1).A)  # M2节点向外转账; di+
colDeltas = np.squeeze(M2.sum(axis=0).A)  # 列求和，终止节点wi  c
```

```
for (m1, m2) in zip(midDeltas1, midDeltas2):
    temp = min(m1, m2)  # 计算fi(S)
    temp2 = max(m1, m2)  # 计算qi(S)
    mid_min.append(temp)  # 每个节点的fi
    mid_max.append(temp2)  # 每个节点的qi
```

$$f_i(\mathcal{S}) = \min\{d_i^+(\mathcal{S}), d_i^-(\mathcal{S})\},\ \forall\, v_i \in \mathcal{M}_l \qquad (1)$$

$$q_i(\mathcal{S}) = \max\{d_i^+(\mathcal{S}), d_i^-(\mathcal{S})\},\ \forall\, v_i \in \mathcal{M}_l \qquad (2)$$

```
new_mid_priority = (1+alpha) * mid_min - alpha * mid_max  # 计算中间节点, Wi(S)
```

$f_i(S)$ $q_i(S)$

$$w_i(\mathcal{S}) = \begin{cases} f_i(\mathcal{S}) - \dfrac{\lambda}{1+\lambda} q_i(\mathcal{S}), & \text{if } v_i \in \mathcal{M}_l \\ d_i(\mathcal{S}), & \text{if } v_i \in \mathcal{A} \cup \mathcal{C} \end{cases} \qquad (5)$$

```python
class MinTree:
    def __init__(self, degrees):  self:    degrees:
        self.height = int(math.ceil(math.log(len(degrees), 2)))  # log(len)
        self.numLeaves = 2 ** self.height  #  the operater form of pow
        self.numBranches = self.numLeaves - 1
        self.n = self.numBranches + self.numLeaves
        self.nodes = [float('inf')] * self.n
        for i in range(len(degrees)):
            self.nodes[self.numBranches + i] = degrees[i]
        for i in reversed(range(self.numBranches)):
            self.nodes[i] = min(self.nodes[2 * i + 1], self.nodes[2 * i + 2])
```

$self.height = 3$

$self.numLeaves = 8$

$self.numBranches = 7$

$self.n = 15$

$degrees = [0.12, 0.2, 0.1, 0.15, 0.2, 0.3, 0.4]$

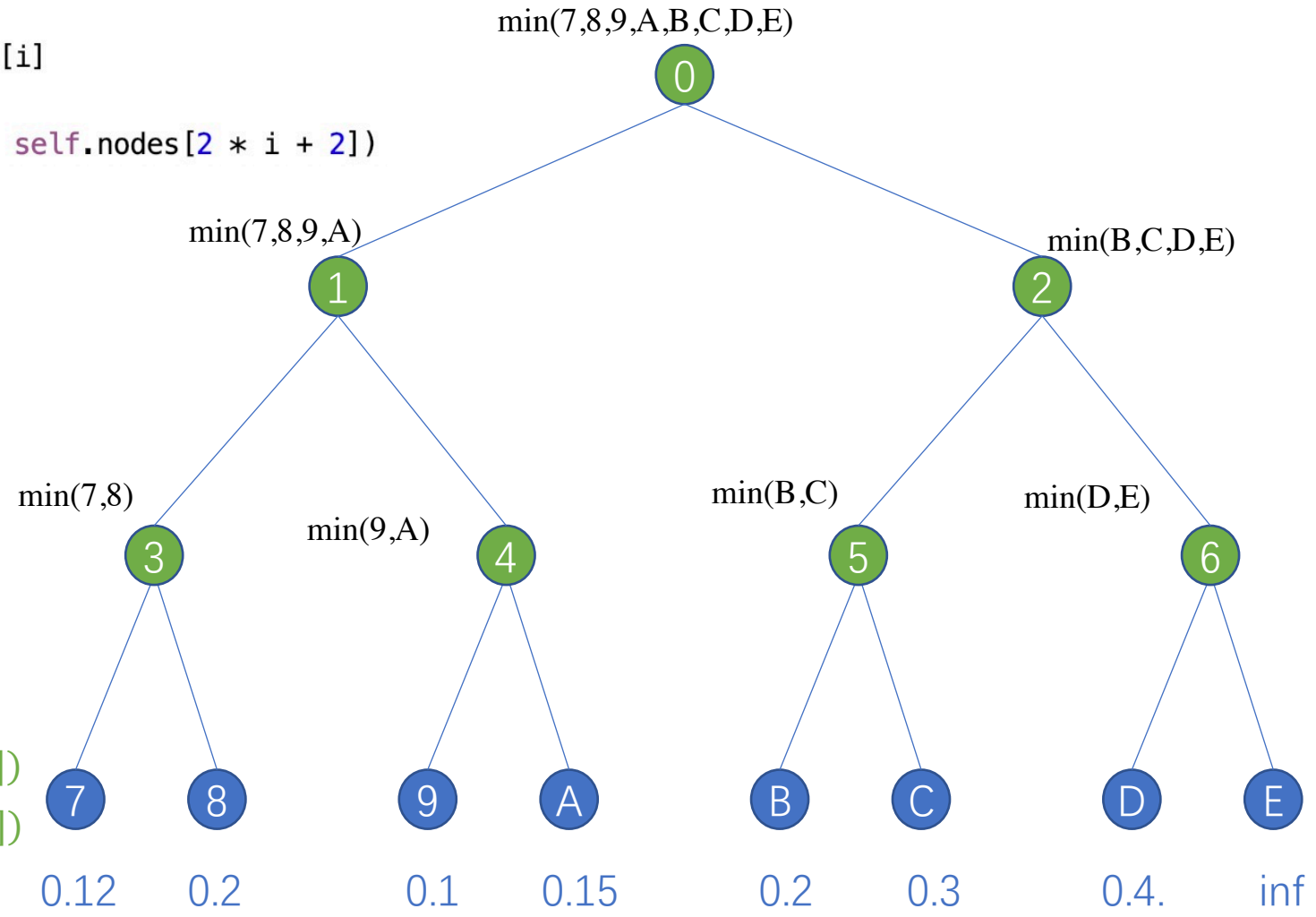
$self.nodes[7] = 0.12$

$self.nodes[8] = 0.2$ 叶子节点赋值

$self.nodes[6] = \min(slef.nodes[13], self.nodes[14])$

$self.nodes[5] = \min(slef.nodes[11], self.nodes[12])$

分支节点赋值



```
curScore1 = sum(mid_min)  # fi的总和
curScore2 = sum(abs(midDeltas1 - midDeltas2))
curAveScore1 = curScore1 / (len(rowSet) + len(midSet) + len(colSet))
curAveScore2 = curScore2 / (len(rowSet) + len(midSet) + len(colSet))
```

M1节点收款　M2节点向外转账　流入流出差

$$g^k(\mathcal{S}) = \frac{1}{|\mathcal{S}|}\sum_{l=1}^{k-2}\sum_{v_i \in \mathcal{M}_l} f_i(\mathcal{S}) - \lambda(q_i(\mathcal{S}) - f_i(\mathcal{S})) \qquad (3)$$

节点i转账的最大值 – 转账最小值

$$f_i(\mathcal{S}) = \min\{d_i^+(\mathcal{S}), d_i^-(\mathcal{S})\},\ \forall\, v_i \in \mathcal{M}_l \qquad (1)$$

$$q_i(\mathcal{S}) = \max\{d_i^+(\mathcal{S}), d_i^-(\mathcal{S})\},\ \forall\, v_i \in \mathcal{M}_l \qquad (2)$$

```
curAveScore1 = curScore1 / (len(rowSet) + len(midSet) + len(colSet))  # 重新计算分值
```

课程总结

1. 群组异常检测：Fraudar, SliceNDice, DeepFD
2. 离群点异常检测：ONE, DAGMM
3. 时间序列异常检测：MSCRED, TadGAN, TS2Vec
4. 资金关系异常检测：FlowScope