

2025 年 9 月 29 日至 10 月 12 日周报

何瑞杰
中山大学, 大湾区大学

1. 项目进展

1.1. 使用神经网络学习生命游戏的演化动力学

1.1.1. 生命游戏规则的变种

本周取 Golly 文档中若干邻域大小为 3 的九种其他规则进行实验，每种规则的具体信息列表如下

规则名称	邻域大小	邻域类型	特性描述
B36/S23	3	Moore	和 Conway 的原版生命游戏相似，但有自我复制结构
B3678/S34678	3	Moore	活细胞群中的死细胞的行为与死细胞群中的活细胞的行为相同
B35678/S5678	3	Moore	有不可预测行为的菱形斑点
B2/S	3	Moore	活细胞每代都会死亡，但该系统常常爆发
B234/S	3	Moore	单个的 2×2 会演化为一个波斯地毯
B345/S5	3	Moore	周期极长的振荡器可以自然地出现
B13/S012V	3	Von Neumann	
B2/S013V	3	Von Neumann	

1.1.2. 数据生成

经过进一步研究发现，Golly 虽然支持大量规则，但无法作为包导入 Python 中使用，只限于其程序之内。一番搜索后我找到了 `pyseagull`，并对所需的关键部分进行了检查。其运行模式十分简单，下面是一段官网给出的模拟代码：

```
import seagull as sg
from seagull.lifeforms import Pulsar

# Initialize board
board = sg.Board(size=(19,60))

# Add three Pulsar lifeforms in various locations
board.add(Pulsar(), loc=(1,1))
board.add(Pulsar(), loc=(1,22))
board.add(Pulsar(), loc=(1,42))

# Simulate board
sim = sg.Simulator(board)
sim.run(sg.rules.conway_classic, iters=1000)
```

相比于先前代码中的逻辑，它要简单得多。即使运行过程中被包装成一个函数，我们依然可以通过 `sg.Simulator` 中的 `get_history()` 方法得到这一次模拟的所有历史数据的 `ndarray`，其形状为 `[iters+1, w, h]`，其中 `iters` 为迭代轮数，`w` 和 `h` 为网格尺寸大小。

另外，`pyseagull` 还支持自定义的简单规则。生命游戏的简单规则可以写为 `B[...]/S[...]` 的字符串格式。最经典的生命游戏的规则为 `B3/S23`，意为死细胞邻居存活数为 3 时，下一时刻复活；活细胞邻居存活数为 2 或 3 时下一时刻继续存活，其他情况下一时刻细胞死亡。自定义函数签名如下

```
seagull.rules.life_rule(X: ndarray, rulestring: str)
```

该函数在 `pyseagull` 的源码实现中通过正则表达式提取 `rulestring` 中的规则信息。由于其灵活性，在需要时我们可以将其拓展为其他更加复杂的规则，例如改变邻域的形状、将邻域的贡献从各向同性改为各向异性。

1.1.3. 对模型和训练的改动

1.1.3.1. 增大卷积核大小

我将 `SimpleCNNTiny` 和 `SimpleCNNSmall` 的卷积核大小增加到 5。

1.1.3.2. 缩小并行模型的参数量

我将 `MultiScale` 网络中并行层中输出的特征图的通道数都降为 2。具体而言，调整后网络的结构如下

```
class MultiScale(nn.Module):
    __version__ = '0.2.0'
    def __init__(self):
        super(MultiScale, self).__init__()
        self.conv_3x3 = nn.Sequential(
            nn.Conv2d(2, 2, kernel_size=3, stride=1,
                     padding=1, padding_mode="circular"),
            nn.BatchNorm2d(2),
            nn.LeakyReLU(0.1)
        )
        self.conv_5x5 = nn.Sequential(
            nn.Conv2d(2, 2, kernel_size=5, stride=1,
                     padding=2, padding_mode="circular"),
            nn.BatchNorm2d(2),
            nn.LeakyReLU(0.1)
        )
        self.conv_3x3_dilated = nn.Sequential(
            nn.Conv2d(2, 2, kernel_size=3, stride=1,
                     padding=2, dilation=2, padding_mode="circular"),
            nn.BatchNorm2d(2),
            nn.LeakyReLU(0.1)
        )
        self.stem = nn.Sequential(
            nn.Conv2d(int(2*3), 4, kernel_size=3, stride=1,
                     padding=1, padding_mode="circular"),
            nn.BatchNorm2d(4),
            nn.ReLU(),
            nn.Conv2d(2, 2, kernel_size=3, stride=1,
                     padding=1, padding_mode="circular")
        )
    ...

```

1.1.3.3. 权重稀疏化

在损失函数中添加 L1 损失。具体计算方式如下

```

l1_reg = 0
for name, param in model.named_parameters():
    if 'weight' in name:
        l1_reg = l1_reg + torch.linalg.vector_norm(param, ord=1, dim=None)

```

然后将 $l1_reg$ 假如损失函数中，权重为 10^{-5} 。

此方法来源于 <https://stackoverflow.com/a/58533398>

1.1.4. 群等变 CNN

我发现群等变 CNN 也许可以较好地刻画生命游戏中的平移和旋转不变性——前者通常卷积已经满足，而对于旋转变换，套用群等变 CNN 论文中的记号，假设输入特征图是信号 f ，旋转变换是 L_τ ，神经网络是 F ，我期待它可以满足

$$L_\tau[F \circ f] = [F \circ [L_\tau f]] \quad (1)$$

而这就是对旋转群的等变性质。依照生命游戏的设定，我们需要神经网络对于晶体群 $p4$ 的等变性。由于在 GCNN 这篇论文后有后人提出更普适的框架 e2cnn，而该框架包含我们所需的 P4CNN，因此我采用该框架实现可插拔替换的群等变 CNN 模型。该框架使用方便，仅需仿照通常的 CNN 模型，然后做微小的改动，即可实现插拔可用。我对标 small 大小的 CNN 模型得到了 $p4$ 群等变 CNN：

```

class SimpleP4CNNSmall(GroupEquivariantCNN):
    __version__ = '0.1.0-p4'
    def __init__(self):
        super().__init__()

        # Define transformation group to be p4
        r2_act = gspaces.Rot2dOnR2(N=4)

        # Define input, hidden and output field type
        in_type = enn.FieldType(r2_act, 2 * [r2_act.trivial_repr])
        hid_type = enn.FieldType(r2_act, 8 * [r2_act.regular_repr])
        out_type = enn.FieldType(r2_act, 2 * [r2_act.trivial_repr])
        self.in_type = in_type
        self.out_type = out_type

        # Network blocks
        self.conv1 = enn.R2Conv(in_type, hid_type, kernel_size=3, bias=False,
                               stride=1, padding=1, padding_mode="circular")
        self.bn1 = enn.InnerBatchNorm(hid_type)
        self.act1 = enn.ReLU(hid_type, inplace=True)
        self.conv2 = enn.R2Conv(hid_type, hid_type, kernel_size=3, bias=False,
                               stride=1, padding=1, padding_mode="circular")
        self.bn2 = enn.InnerBatchNorm(hid_type)
        self.act2 = enn.ReLU(hid_type, inplace=True)
        self.conv3 = enn.R2Conv(hid_type, out_type, kernel_size=3, bias=False,
                               stride=1, padding=1, padding_mode="circular")

    def forward(self, x: Float[Array, "batch 2 w h"]) -> Float[Array, "batch 2 w h"]:
        # Turn pytorch tensor to GeometricTensor
        x: enn.GeometricTensor = enn.GeometricTensor(x, self.in_type)
        x = self.act1(self.bn1(self.conv1(x)))
        x = self.act2(self.bn2(self.conv2(x)))
        x = self.conv3(x)

        # Convert GeometricTensor back to pytorch Tensor

```

```

        return x.tensor

def export(self, size) -> nn.Module:
    """
    returns a version of model that doesn't require e2cnn package
    """
    return torch.jit.trace(self, torch.randn(*size))

```

为了与模型训练中的 `torchinfo.summary` 函数兼容，我们将调用方式改为 `summary(model.cpu().export(), ...)` 以保证该函数的正常运行。后者的运行结果为

Layer (type:depth-idx)	Output Shape	Param #
SimpleP4CNNSmall	--	--
R2Conv: 1-1	--	96
└ BlocksBasisExpansion: 2-1	--	--
└ SingleBlockBasisExpansion: 3-1	--	--
InnerBatchNorm: 1-2	--	--
└ BatchNorm3d: 2-2	--	16
ReLU: 1-3	--	--
R2Conv: 1-4	--	1,536
└ BlocksBasisExpansion: 2-3	--	--
└ SingleBlockBasisExpansion: 3-2	--	--
InnerBatchNorm: 1-5	--	--
└ BatchNorm3d: 2-4	--	16
ReLU: 1-6	--	--
R2Conv: 1-7	--	98
└ BlocksBasisExpansion: 2-5	--	--
└ SingleBlockBasisExpansion: 3-3	--	--
Total params: 1,762		
Trainable params: 1,762		
Non-trainable params: 0		
Total mult-adds (M): 0		
Input size (MB): 0.32		
Forward/backward pass size (MB): 0.00		
Params size (MB): 0.00		
Estimated Total Size (MB): 0.32		

1.1.5. 在不同规则的演化系统上的实验结果

实验正在运行，目前的结果是群等变 CNN (small) 可以很好的学习上述的所有规则（测试集正确率接近 100%）。而使用并行多尺度 CNN 的训练结果在区间不连续，例如 B3678/S34678 这样的规则下学习较为困难。另外我注意到对于不同的规则，不能使用一套超参数生成数据，由于规则不同，使用一套参数在某些规则下会产生大量的重复数据，这会影响网络的训练。

具体的结果分析会在下周的周报中呈现，其中包括所有情况的训练曲线，以及对神经网络作为演化模拟器和训练权重的分析。可遇见的困难是，对于群等变 CNN，我应该如何提取并解释它的权重。

参考资料

1. <https://pyseagull.readthedocs.io/>
2. <https://arxiv.org/abs/1602.07576>
3. <https://github.com/QUVA-Lab/e2cnn>

2. 文献阅读

2.1. Group Equivariant Convolutional Networks

- Taco S. Cohen and Max Welling
- <https://arxiv.org/abs/1602.07576>

2.1.1. 何为等变，等变何为

考虑线性空间 V 和上面的一个变换群 \mathfrak{G} ，我们称之为 \mathfrak{G} -空间。对于线性空间上的一个函数 Φ ，如果它满足

$$\Phi(T_{\mathfrak{g}}x) = T'_{\mathfrak{g}}\Phi(x) \quad (2)$$

其中 $T_{\mathfrak{g}}$ 是指对 V 中的向量做对应于群元 \mathfrak{g} 的变换。 T 和 T' 不必相同，但必须要是 \mathfrak{G} 中元素的线性表示，即满足对任意 $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$ ，有 $T(\mathfrak{gh}) = T(\mathfrak{g})T(\mathfrak{h})$ 。

为什么我们需要等变性？众所周知，CNN 的卷积模块对输入使用了共用参数的一个卷积核，因此具有平移等变性（直觉就能看出，稍后证明），但对旋转或是更复杂的变换没有等变性。相比之下，人眼可以识别出一个在我们所居住的三维空间中以任意可能姿态出现的同一物体：不管它是出现在什么位置、什么姿态、还是镜中或水中的倒影。在图像中，我们也可以轻易看出被平移、旋转或是镜像后的图像包含的还是原来的那个物体，而我们希望将神经网络也加入某种“几何先验”，让神经网络也具备这样的能力。

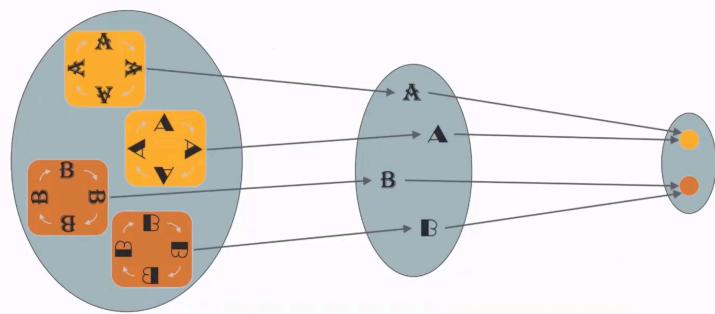


Figure 1: 人眼的旋转等变性——对等变模型的期望

2.1.2. 对称群

本文中对除了 $p4$ 、 $p4m$ 这样的晶体群以外的群和群元素使用 `mathfrak` 字体，如 $\mathfrak{g}, \mathfrak{h}, \mathfrak{G}, \mathfrak{H}, \dots$ 以与通常的函数 f, g, h, \dots 作区分。两个对于图像而言典型的对称群分别为 $p4$ 和 $p4m$ 。前者是包含了 \mathbb{Z}^2 上的所有平移变换和以 $\frac{\pi}{2}$ 为单位的旋转变换，它有下面的表示：

$$\mathfrak{g}(r, u, v) = \begin{bmatrix} \cos(\frac{r\pi}{2}) & -\sin(\frac{r\pi}{2}) & u \\ \sin(\frac{r\pi}{2}) & \cos(\frac{r\pi}{2}) & v \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

其中 $r \in \{0, 1, 2, 3\}$ ， $(u, v) \in \mathbb{Z}^2$ 。群元作用在向量上的结果就可以写为

$$\mathfrak{g}x \simeq \begin{bmatrix} \cos(\frac{r\pi}{2}) & -\sin(\frac{r\pi}{2}) & u \\ \sin(\frac{r\pi}{2}) & \cos(\frac{r\pi}{2}) & v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \quad (4)$$

类似地， $p4m$ 也有类似的表示：

$$\mathfrak{g}(r, u, v, m) = \begin{bmatrix} (-1)^m \cos(\frac{r\pi}{2}) & -(-1)^m \sin(\frac{r\pi}{2}) & u \\ \sin(\frac{r\pi}{2}) & \cos(\frac{r\pi}{2}) & v \\ 0 & 0 & 1 \end{bmatrix} \quad (5)$$

2.1.3. 特征图的信号视角

对于形状为 $[c, w, h]$ 的特征图 F , 为推到方便起见, 我们可以将其看作是有一个有界支撑集的函数:

$$F : \mathbb{Z}^2 \rightarrow \mathbb{R}^c$$

$$(x, y) \mapsto \begin{cases} F[:, x, y], & x \in [0, w-1], y \in [0, h-1] \\ \mathbf{0}, & \text{otherwise} \end{cases} \quad (6)$$

这样就可以建立和 \mathbb{R} 上函数之间卷积操作类似的表达式。对于一个信号 F , 我们定义群元素 \mathfrak{g} 作用在其上的结果为

$$L_{\mathfrak{g}} f(x) = f(\mathfrak{g}^{-1}x). \quad (7)$$

其中 $L_{\mathfrak{g}}$ 是对应于群元 \mathfrak{g} 之变换 $T_{\mathfrak{g}}$ 的一个实例化, 并满足 $L_{\mathfrak{g}} L_{\mathfrak{h}} = L_{\mathfrak{gh}}$ 。它的直观理解是, 假如 $L_{\mathfrak{g}}$ 是一个向左的平移变换, 则 $L_{\mathfrak{g}} f(x)$ 就将信号 (例如图片) 向左平移 c , 则平移后图片中给定位置的像素值就等于原图片中向右平移相同距离的像素值, 也即 $f(x + c)$ 。

2.1.4. 通常卷积模块的等变性

首先回忆 CNN 中的卷积操作 $*$ 和相关操作 \star , 它们在 CNN 的前向传播和反向传播中成对出现。考虑特征图 $f : \mathbb{Z}^2 \rightarrow \mathbb{R}^{K^{(l)}}$ 和一组中的某个卷积核 $\psi^{(i)} : \mathbb{Z}^2 \rightarrow \mathbb{R}^{K^{(l)}}$, 有

$$[f * \psi](\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\mathbf{y}) \psi_k^{(i)}(\mathbf{x} - \mathbf{y}) \quad \text{convolution} \quad (8)$$

$$[f \star \psi](\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\mathbf{y}) \psi_k^{(i)}(\mathbf{y} - \mathbf{x}) \quad \text{correlation}$$

现在我们验证相关操作的平移等变性。考虑 \mathbb{Z}^2 上的平移群元 t 对应的变换 L_t , 有

$$[(L_t f) \star \psi](\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} [L_t f_k](\mathbf{y}) \psi_k^{(i)}(\mathbf{y} - \mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\mathbf{y} - t) \psi_k^{(i)}(\mathbf{y} - \mathbf{x})$$

$$= \sum_{\mathbf{z} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\mathbf{z}) \psi_k^{(i)}(\mathbf{z} - (\mathbf{x} - t)) = [f \star \psi](\mathbf{x} - t) = [L_t[f \star \psi]](\mathbf{x}). \quad \mathbf{z} \leftarrow \mathbf{y} - t \quad (9)$$

但是相关变换对旋转没有等变性, 对于 p4 中的群元 r 对应的旋转变换 L_r , 有:

$$[(L_r f) \star \psi](\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} [L_r f_k](\mathbf{y}) \psi_k^{(i)}(\mathbf{y} - \mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(r^{-1}\mathbf{y}) \psi_k^{(i)}(\mathbf{y} - \mathbf{x})$$

$$= \sum_{\mathbf{z} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\mathbf{z}) \psi_k^{(i)}(r\mathbf{z} - \mathbf{x}) = \sum_{\mathbf{z} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\mathbf{z}) \psi_k^{(i)}(r(\mathbf{z} - r^{-1}\mathbf{x})) \quad \mathbf{z} \leftarrow r^{-1}\mathbf{y} \quad (10)$$

$$= \sum_{\mathbf{z} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\mathbf{z}) \left[L_{r^{-1}} \psi_k^{(i)} \right] (\mathbf{z} - r^{-1}\mathbf{x})$$

$$= [f \star L_{r^{-1}} \psi](r^{-1}\mathbf{x}) = L_r[f \star L_{r^{-1}} \psi](\mathbf{x})$$

可见通常的相关操作对旋转没有等变性。对于卷积，注意只需令 $\varphi(x) = \psi(-x)$ ，相关操作就变成了卷积操作。因此卷积的等变性和相关操作相同。

2.1.5. 群等变模块

2.1.5.1. 群等变相关操作

为了让卷积操作对旋转、以至更加一般的操作具有等变性，作者提出了群相关操作。注意上文中的相关操作

$$[f \star \psi^{(i)}](\mathbf{x}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\mathbf{y}) \psi_k^{(i)}(-\mathbf{x} + \mathbf{y}) \quad (11)$$

注意 \mathbf{x} 对应着 \mathbb{Z}^2 上所有平移操作构成的群中的一个群元素 \mathbf{g} ，而 $-\mathbf{x}$ 对应着它的逆元 \mathbf{g}^{-1} 。我们可以自然地将原来相关操作写成包含群元素的形式，这就得到了第一层的群相关：

$$f^{(1)}(\mathbf{g}) = [f \star \psi^{(i)}](\mathbf{g}) = \sum_{\mathbf{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k^{(0)}(\mathbf{y}) \psi_k^{(0,i)}(\mathbf{g}^{-1} \mathbf{y}) \quad (12)$$

其中 $\mathbf{g} \in \mathfrak{G}$, f 和 $\psi^{(0,i)}$ 都是 \mathbb{Z}^2 到 \mathbb{R} 的函数，但得到的新信号 $[f \star \psi^{(i)}]$ 的定义域为 \mathfrak{G} 。因此接下来的各层中群相关操作的表达式需要做一些微调：

$$f^{(l+1)}(\mathbf{g}) = [f \star \psi^{(i)}](\mathbf{g}) = \sum_{\mathbf{h} \in \mathfrak{G}} \sum_{k=1}^{K^{(l)}} f_k^{(l)}(\mathbf{h}) \psi_k^{(l,i)}(\mathbf{g}^{-1} \mathbf{h}) \quad (13)$$

其中 $l \geq 1$, $\mathbf{g}, \mathbf{h} \in \mathfrak{G}$, $f_k^{(l)}$ 和 $\psi_k^{(l,i)}$ 的定义域都是 \mathfrak{G} (这里假设每层的变换群相同)。接下来证明它是关于群 \mathfrak{G} 的元素等变的：

$$\begin{aligned} [[L_u f] \star \psi](\mathbf{g}) &= \sum_{\mathbf{h} \in X} \sum_{k=1}^K [L_u f_k](\mathbf{h}) \psi_k^{(i)}(\mathbf{g}^{-1} \mathbf{h}) = \sum_{\mathbf{h} \in X} \sum_{k=1}^K f_k(u^{-1} \mathbf{h}) \psi_k^{(i)}(\mathbf{g}^{-1} \mathbf{h}) \\ &= \sum_{\mathbf{p} \in X} \sum_{k=1}^K f_k(\mathbf{p}) \psi_k^{(i)}(\mathbf{g}^{-1} u \mathbf{p}) = \sum_{\mathbf{p} \in X} \sum_{k=1}^K f_k(\mathbf{p}) \psi_k^{(i)}((u^{-1} \mathbf{g})^{-1} \mathbf{p}) \quad \mathbf{p} \leftarrow u^{-1} \mathbf{h} \quad (14) \\ &= [f \star \psi](u^{-1} \mathbf{g}) = [L_u [f \star \psi]](\mathbf{g}) \end{aligned}$$

注意当 \mathfrak{G} 不是交换群时，群卷积和群相关操作也不交换，但是有 $f \star \psi = [\psi \star f]^*$ ，其中 \square^* 是内卷积操作，即 $f^*(g) = f(g^{-1})$ ：

$$\begin{aligned} [f \star \psi](\mathbf{g}) &= \sum_{\mathbf{h} \in X} \sum_{k=1}^K f_k(\mathbf{h}) \psi_k(\mathbf{g}^{-1} \mathbf{h}) = \sum_{\mathbf{h} \in X} \sum_{k=1}^K f_k(\mathbf{h}) \psi_k(\mathbf{g}^{-1} \mathbf{h}) \quad \mathbf{p} \leftarrow \mathbf{g}^{-1} \mathbf{h} \quad (15) \\ &= \sum_{\mathbf{p} \in X} \sum_{k=1}^K \psi_k(\mathbf{p}) f_k((\mathbf{g}^{-1})^{-1} \mathbf{p}) = [\psi \star f](\mathbf{g}^{-1}) = [\psi \star f]^*(\mathbf{g}). \end{aligned}$$

2.1.5.2. 群等变信号的线性组合和单点非线性函数的等变性

假设有两个关于群 \mathfrak{G} 等变的信号 $f(\cdot)$ 和 $g(\cdot)$ ，显然其线性组合也是关于群 \mathfrak{G} 等变的：

$$[L_g[af + bg]](\mathbf{u}) = [af + bg](\mathbf{g}^{-1} \mathbf{u}) = [aL_g f + bL_g g](\mathbf{u}) \quad (16)$$

其中 $\mathbf{g} \in \mathfrak{G}$ 。对于单点非线性函数 $\sigma: \mathbb{R} \rightarrow \mathbb{R}$ ，其作用在信号上的方式是简单的函数复合，即 $\sigma \circ f$ 容易证明它对群 \mathfrak{G} 中的任一元素 \mathbf{g} 的等变性：

$$L_g[\sigma \circ f](\mathbf{u}) = [\sigma \circ f](\mathbf{g}^{-1} \mathbf{u}) = \sigma[f(\mathbf{g}^{-1} \mathbf{u})] = [\sigma \circ L_g](\mathbf{u}). \quad (17)$$

2.1.5.3. 子群池化和陪集池化

回忆通常卷积网络的（最大值）池化操作，对于单通道的信号 $f : \mathbb{Z}^2 \rightarrow \mathbb{R}$ ，它其实可以写为

$$[\text{MaxPool}_1 f](x) = \max_{u \in \mathcal{N}(x)} f(u) \quad (18)$$

其中 $\mathcal{N}(x)$ 是 x 的一个邻域，实践中常常为以 x 为中心的一个正方形邻域，如九宫格。再在 \mathbb{Z}^2 的某个子集（例如 $(3\mathbb{Z})^2$ ）上采样，最后再映射回 \mathbb{Z}^2 。这就是池化中的两步操作：

1. 计算给定位置 x 的邻域最大值
2. 选取一部分邻域最大值作为输出

我们可以依据这两步构造包含群的版本。首先对于第一步，我们可以考虑一个在单位变换 e 的一个包含于群 \mathfrak{G} 的“邻域” $\mathfrak{U} \subset \mathfrak{G}$ ，然后考虑 $g\mathfrak{U} := \{gu : u \in \mathfrak{U}\}$ ，这就得到了邻域最大值的群元素版本：

$$Pf(g) = \max_{t \in g\mathfrak{U}} f(t) \quad (19)$$

这个操作是群等变的：

$$\begin{aligned} [L_u P]f(g) &= L_u[Pf](g) = [Pf](u^{-1}g) = \max_{t \in u^{-1}g\mathfrak{U}} f(t) \\ &= \max_{h \in g\mathfrak{U}} f(u^{-1}h) = \max_{h \in g\mathfrak{U}} [L_u f](h) \quad h \leftarrow uh \\ &= [PL_u](g). \end{aligned} \quad (20)$$

简而言之，有 $L_u P = PL_u$ ，因为算子复合有结合律。在此之后，回忆一般池化的采样为网格 \mathbb{Z}^2 的一个子集，也是一个网格。以群的视角来看，就对应着 \mathfrak{G} 的子群 \mathfrak{H} 。因此群池化的第二步就是用子群 \mathfrak{H} 的元素对 Pf 做下采样。

接着我们看一个十分有用的特例。在上文中我们并未要求 \mathfrak{U} 的代数结构，此时我们若要求 \mathfrak{U} 为 \mathfrak{G} 的一个子群 \mathfrak{H} ，那么对于 $g \in \mathfrak{G}$ ， $g\mathfrak{H}$ 被称为是一个陪集（coset），而陪集可以将原来的群 \mathfrak{G} 做划分，并得到等价关系 $g \sim h \Leftrightarrow g\mathfrak{H} = h\mathfrak{H}$ 。对于处于同一等价类中的元素 g 和 h ，容易验证 $Pf(g) = Pf(h)$ ，因此对于每个不同的陪集，我们只需选出一个代表元来计算 Pf 的函数值即可，这就是陪集池化。我们也可以将陪集池化中的 Pf 视作在商集 $\mathfrak{G}/\mathfrak{H}$ 上的函数。

最后给出一个陪集池化的例子。考虑旋转群 $p4$ ，如果我们对所有位置上的所有旋转变换做池化，得到的信号的定义域是 $p4/\mathfrak{R}$ ，它和 \mathbb{Z}^2 同构；如果我们在 \mathbb{Z} 中对子群 $n\mathbb{Z}$ 的陪集做池化，这给出了在 $\mathbb{Z}/n\mathbb{Z}$ 上的特征图信号。

2.1.6. 群相关操作的高效实现

对于平面的变换群 \mathfrak{G} ，如果每个群元 $g \in \mathfrak{G}$ 都可以写成一个平移变换 $t \in \mathbb{Z}^2$ 和一个稳定子，即保零点的旋转变换 s 的复合，我们称它是可分的。例如群 $p4$ ，它的元素 g 总是可以写成 $g = ts$ ，其中 t 是平移， s 是以原点为旋转中心的旋转，根据前文中提到的群元对应的变换的同态性质，有

$$[f * \psi^{(i)}](g) = [f * \psi^{(i)}](ts) = \sum_{h \in \mathfrak{X}} \sum_k f_k(h) L_t \left[L_s \psi_k^{(i)}(h) \right] \quad (21)$$

注意平移变换 L_t 对应的就是通常卷积中的平移变换。所以可以先算出 $L_s \psi_k$ ，然后直接套用通常的卷积即可。其次，注意这里的 ψ_k 是群 \mathfrak{G} 到 \mathbb{R} 的映射，因此该映射可以直接使用一个数组 F 来表示——我们不需要群元素的表示，而是只需要将它们编号，然后构造一个字典/数组来查找即可。

考虑 \mathbb{Z}^2 上可分的变换群 \mathfrak{G} , 由于它的群元可以写成平移和稳定子的复合。假设后者的数量为 S , 水平和垂直方向平移的格数为 n , 那么该群一共有 $S \times n \times n$ 个群元。假设上一层的通道数为 K , 下一层通道数为 K' , 那么对应于这一层卷积核的数组 F 的形状为

$$\overbrace{K' \times K \times \underbrace{S \times n \times n}_{\substack{\text{群元素编号, 对应于 } \psi_k^{(i)} \\ \text{考虑上一层的所有通道, 对应于 } \psi^{(i)}}}}^{\text{考虑下一层的所有通道, 对应于完整的 } \psi}. \quad (22)$$

接着我们需要将 ψ 和 L_s 配对, 由于 s 是任选的稳定子, 配对后相当于一个新的函数 $g(\mathfrak{s}', \mathfrak{s}, \mathfrak{t}) = h(\mathfrak{s}'^{-1}\mathfrak{s}\mathfrak{t})$, 所以配对后的结果是 $K' \times S \times K \times \mathcal{S} \times \mathcal{n} \times \mathcal{n}$, 我们将配对后的数组记为 G 。为了计算 $\mathfrak{s}'^{-1}\mathfrak{s}\mathfrak{t}$ 我们需要用到一个双射 $g(s, u, v)$, 它接受 $S \times n \times n$ 中的一个三元组, 这对应于群 \mathfrak{G} 中一个元素的指标, 输出该群元素对应的(可逆的)矩阵表示。因此 $\mathfrak{s}\mathfrak{t}$ 对应的指标是 (s, t_x, t_y) , 而 \mathfrak{s}' 由于是稳定子, 没有复合平移变换, 因此对应的指标是 $(s', 0, 0)$, 因此可以得到 $\mathfrak{s}'^{-1}\mathfrak{s}\mathfrak{t}$ 对应的指标是

$$(\bar{s}, \bar{t}_x, \bar{t}_y) = g^{-1}[(g(s', 0, 0))^{-1}g(s, t_x, t_y)] \quad (23)$$

因此就有对应 $G[i, s', j, s, t_x, t_y] = F[i, j, \bar{s}, \bar{t}_x, \bar{t}_y]$ 。这样得到的 G 就是对应于 $L_s\psi$ 的映射字典。

接下来要执行的是通常的卷积操作, 只需将 G 的形状 $K' \times S' \times K \times S \times n \times n$ 变形为 $(K'S) \times (KS) \times n \times n$, 即输入通道数为 KS 输出通道数为 $K'S$ 的卷积; 特征图 f 也可以做类似的操作, 其原本的形状为 $K \times S \times n \times n$, 将前两维合并, 得到的形状是 $(KS) \times n \times n$, 和改变形状后的卷积核对应。

2.1.7. 实验结果和讨论

实验中用作参照的模型是全卷积网络和 ResNet。本文提出的模型在做了随机旋转的 RotatedMNIST 和 CIFAR 系列数据集上都有相比于其他模型更好的效果; 在群卷积的范畴之中(通常卷积可以视作 \mathbb{Z}^2 上平移变换群卷积这一特殊情形) 拥有最多群元素 $p4m$ 在 CIFAR 系列数据集上的效果最好。

Table 1: 本文模型 (P4CNN) 与各 Baseline
模型在 Rotated MNIST 上的训练误差

Network	Test Error (%)
Larochelle et al. (2007)	10.38 ± 0.27
Sohn & Lee (2012)	4.2
Schmidt & Roth (2012)	3.98
Z2CNN	5.03 ± 0.0020
P4CNNT Rotation Pooling	3.21 ± 0.0012
P4CNN	2.28 ± 0.0004

Table 2: 通常卷积网络 (\mathbb{Z}^2)、 $p4$ 和 $p4m$ 在 CIFAR10 和 CIFAR10+ 上的训练误差和模型参数量

Network	G	CIFAR10	CIFAR10+	Param.
All-CNN	\mathbb{Z}^2	9.44	8.86	1.37M
	$p4$	8.84	7.67	1.37M
	$p4m$	7.59	7.04	1.22M
	\mathbb{Z}^2	9.45	5.61	2.64M
ResNet44	$p4m$	6.46	4.94	2.62M

从上面的结果可以看到, 群等变模块可以替换通常卷积模型中的卷积、非线性层和池化层, 并得到更好的效果。

群等变 CNN 的进一步的拓展改进包括将六边形网格中的变换群、 \mathbb{R}^3 中的变换群、连续(局部紧)群以及更大的有限群。不过对于后两个实现难度和计算成本较大。

3. 学习进度

3.1. 随机过程

本周学习到了 Poisson 过程的定义和若干简单性质。

3.2. 随机微分方程

本周没有推进。

4. 问题解决记录

4.1. Typst 相关

4.1.1. 数学公式自动编号

4.2. Python 相关

4.2.1. 猴子补丁

在使用 `pyseagull` 这个包时，我对它的源码进行了一些改动。这导致这些改动在其他机器上搭建环境时不可迁移。解决此问题的方法是使用 Python 的 [猴子补丁](#)。它利用 Python 的灵活性，直接覆盖包中的某些函数或类参数。以 `pyseagull` 为例，假如我需要修改它的 `life_rule` 函数，而 `life_rule` 又要用到 `_parse_rulestring` 和 `_count_neighbors`，就可以像下面这样写

```
from seagull.rules import life_rule

def life_rule_monkey_patch(X: np.ndarray, rulestring: str) -> np.ndarray:
    """
    Monkey Patch for function `life_rule`.
    Add support for Von Neumann Neighborhood.
    """

    ...

def _parse_rulestring_monkey_patch(r: str) -> Tuple[List[int], List[int]]:
    """
    Add support for Von Neumann Neighborhood.
    """

    ...

def _count_neighbors_monkey_patch(X: np.ndarray, von_neumann: bool) -> np.ndarray:
    """
    Add support for Von Neumann Neighborhood.
    """

    ...

# Apply monkey patch
life_rule = life_rule_monkey_patch
```

将这段代码放在前面，执行时就会对 `seagull` 的 `life_rule` 函数做直接的替换，以实现自定义的新功能。对于类变量，也是同理。

5. 下周计划

论文阅读

1. 生成模型
 - DDPM 收尾
 - Sliced Score Matching: A Scalable Approach to Density and Score Estimation
2. 动力学
 - 暂无
3. 其他
 - General $E(2)$ - Equivariant Steerable CNNs

项目进度

1. 使用神经网络学习生命游戏的演化动力学
 - 收集实验数据，若有必要，实现更小的 $p4$ -等变 CNN 模型并测试
 - 尝试对模型权重进行解释，并用模型作为系统的演化模拟器，统计验证所学到的规则是否正确
2. 微型抗癌机器人在血液中的动力学
 - 开始学习 PDE 的数值解方法

理论学习

1. 随机过程课程
 - 完成 Poisson 过程的学习
 - 预习 Markov 过程
2. 随机微分方程
 - 完成第四章 随机积分
 - 第五章 随机微分方程 开头