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：

```python
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

1.1.5. 在不同规则的演化系统上的实验结果

1.1.5.1. 调整通常卷积网络参数和权重稀疏化后的结果

1.1.5.1.1. 训练曲线

1.1.5.1.2. 预测结果

1.1.5.2. 以上改动基础上将网络改为群等变 CNN 后的结果

1.1.5.2.1. 训练曲线

1.1.5.2.2. 预测结果

1.1.6. 对模型的可视化的解释

1.1.6.1. 训练完毕的神经网络作为生命游戏模拟器

1.1.6.2. 通过 CNN 权重的直接解释方案

参考资料

1. https://pyseagull.readthedocs.io/
2.

1.2. 微型抗癌机器人在血液中的动力学

1.2.1. 项目目的

微型抗癌机器人是通过癌症细胞散发出的化学吸引物 (chemoattractant) 趋化性驱动 (chemotaxis-driven) 运动，与癌细胞进行配体-受体结合后定向释放药物，达到治疗的目的。本项目研究理想状况下的微型抗癌机器人集群在血液中的动力学。

1.2.2. 建模

目前项目对血液中的化学吸引物、游离的微型机器人和与癌细胞结合的微型机器人分布进行建模。设 t 时刻，位于血液中 $\boldsymbol{x}$ 位置的化学吸引物浓度为 $c(\boldsymbol{x},t)$，化学吸引物正常的消耗或讲解速率为 k，，则有

$$\frac{\partial c}{\partial t} = D_c \nabla^2 c - kc + S_{\Omega_t}(\boldsymbol{x}) \tag{1}$$

其中

- D_c 为化学吸引物在血液中的扩散系数
- k 为化学吸引物正常的消耗或讲解速率
- Ω_t 为癌细胞所在区域，$S_{\Omega_t}(\boldsymbol{x})$ 为癌细胞区域中 $\boldsymbol{x}$ 位置向血液中释放化学吸引物的速度

类似地，设 $\rho(\boldsymbol{x},t)$ 为游离机器人血液中的分布密度，$b(\boldsymbol{x},t)$ 为非游离的机器人的分布密度，有

$$\begin{aligned} \frac{\partial \rho}{\partial t} &= D_\rho \nabla^2 \rho - \nabla \cdot (\chi \rho \nabla c) - k_b \rho \delta_{\Omega_t} + k_u b \\ \frac{\partial b}{\partial t} &= k_b \rho \delta_{\Omega_t} - k_u b \end{aligned} \tag{2}$$

2. 文献阅读

2.1. Denoising Diffusion Probabilistic Models

Jonathan Ho, Ajay Jain and Pieter Abbeel | https://arxiv.org/abs/2006.11239

本周把 DDPM 的剩余部分补完。

2.1.1. 补遗

2.1.2. 实验结果

2.1.3. 总结和讨论

参考资料

1. https://arxiv.org/abs/1907.05600
2. https://arxiv.org/abs/2006.11239
3. https://lilianweng.github.io/posts/2021-07-11-diffusion-models/#nice

2.2. Sliced Score Matching: A Scalable Approach to Density and Score Estimation

- Yang Song, Sahaj Garg, Jiaxin Shi, Stefano Ermon
- https://arxiv.org/abs/1905.07088

2.3. Group Equivariant Convolutional Networks

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

2.3.1. 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

$$\Phi\left(T_{\mathfrak{g}}x\right) = T'_{\mathfrak{g}}\Phi(x) \tag{3}$$

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。

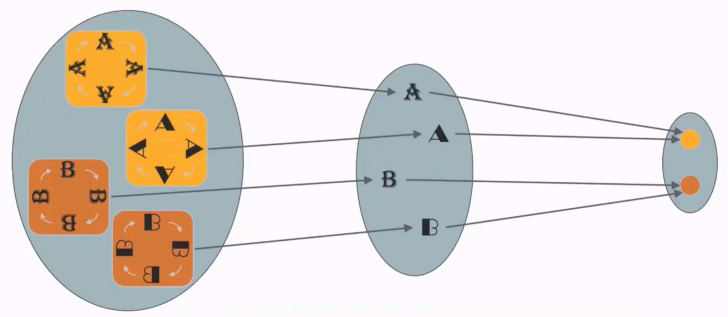

Figure 1: 人眼的旋转等变性——对等变模型的期望

2.3.2. 对称群

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 $\dfrac{\pi}{2}$ 为单位的旋转变换，它有下面的表示：

$$\mathfrak{g}(r,u,v) = \begin{bmatrix} \cos\left(\frac{r\pi}{2}\right) & -\sin\left(\frac{r\pi}{2}\right) & u \\ \sin\left(\frac{r\pi}{2}\right) & \cos\left(\frac{r\pi}{2}\right) & v \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

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

$$\mathfrak{g}x \simeq \begin{bmatrix} \cos\left(\frac{r\pi}{2}\right) & -\sin\left(\frac{r\pi}{2}\right) & u \\ \sin\left(\frac{r\pi}{2}\right) & \cos\left(\frac{r\pi}{2}\right) & v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \tag{5}$$

类似地，$p4m$ 也有类似的表示：

$$\mathfrak{g}(r,u,v,m) = \begin{bmatrix} (-1)^m\cos\left(\frac{r\pi}{2}\right) & -(-1)^m\sin\left(\frac{r\pi}{2}\right) & u \\ \sin\left(\frac{r\pi}{2}\right) & \cos\left(\frac{r\pi}{2}\right) & v \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

2.3.3. 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下面的函数：

$$
\begin{aligned}
&F : \mathbb{Z}^2 \to \mathbb{R}^c \\
&(x, y) \mapsto \begin{cases} \mathtt{F}[:, \mathtt{x}, \mathtt{y}], & x \in [0, w-1], y \in [0, h-1] \\ \mathbf{0}, & \text{otherwise} \end{cases}
\end{aligned} \tag{7}
$$

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

$$
L_{\mathfrak{g}} f(x) = f(\mathfrak{g}^{-1} x). \tag{8}
$$

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 $L_{\mathfrak{g}} L_{\mathfrak{h}} = L_{\mathfrak{g}\mathfrak{h}}$。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 $f(x + c)$。

2.3.4. 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 $f : \mathbb{Z}^2 \to \mathbb{R}^{K^{(l)}}$ 和一组中的某个卷积核 $\psi^{(i)} : \mathbb{Z}^2 \to \mathbb{R}^{K^{(l)}}$，有

$$
\begin{aligned}
[f * \psi](\boldsymbol{x}) &= \sum_{\boldsymbol{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\boldsymbol{y}) \psi_k^{(i)}(\boldsymbol{x} - \boldsymbol{y}) \qquad \text{convolution} \\
[f \star \psi](\boldsymbol{x}) &= \sum_{\boldsymbol{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\boldsymbol{y}) \psi_k^{(i)}(\boldsymbol{y} - \boldsymbol{x}) \qquad \text{correlation}
\end{aligned} \tag{9}
$$

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

$$
\begin{aligned}
[(L_t f) \star \psi](\boldsymbol{x}) &= \sum_{\boldsymbol{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} [L_t f_k](\boldsymbol{y}) \psi_k^{(i)}(\boldsymbol{y} - \boldsymbol{x}) \\
&= \sum_{\boldsymbol{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\boldsymbol{y} - \boldsymbol{t}) \psi_k^{(i)}(\boldsymbol{y} - \boldsymbol{x}) \\
&= \sum_{\boldsymbol{z} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\boldsymbol{z}) \psi_k^{(i)}(\boldsymbol{z} - (\boldsymbol{x} - \boldsymbol{t})) \qquad \boldsymbol{z} \leftarrow \boldsymbol{y} - \boldsymbol{t} \\
&= [f \star \psi](\boldsymbol{x} - \boldsymbol{t}) = [L_t [f \star \psi]](\boldsymbol{x}).
\end{aligned} \tag{10}
$$

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：

$$
\begin{aligned}
[(L_r f) \star \psi](\boldsymbol{x}) &= \sum_{\boldsymbol{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} [L_r f_k](\boldsymbol{y}) \psi_k^{(i)}(\boldsymbol{y} - \boldsymbol{x}) \\
&= \sum_{\boldsymbol{y} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(r^{-1} \boldsymbol{y}) \psi_k^{(i)}(\boldsymbol{y} - \boldsymbol{x}) \\
&= \sum_{\boldsymbol{z} \in \mathbb{Z}^2} \sum_{k=1}^{K^{(l)}} f_k(\boldsymbol{z}) \psi_k^{(i)}(r\boldsymbol{z} - \boldsymbol{x}) \qquad \boldsymbol{z} \leftarrow r^{-1} \boldsymbol{y}
\end{aligned} \tag{11}
$$

$$
\begin{aligned}
&= \sum_{\boldsymbol{z}\in\mathbb{Z}^2}\sum_{k=1}^{K^{(l)}} f_k(\boldsymbol{z})\psi_k^{(i)}(r(\boldsymbol{z}-r^{-1}\boldsymbol{x})) \\
&= \sum_{\boldsymbol{z}\in\mathbb{Z}^2}\sum_{k=1}^{K^{(l)}} f_k(\boldsymbol{z})\left[L_{r^{-1}}\psi_k^{(i)}\right](\boldsymbol{z}-r^{-1}\boldsymbol{x}) \\
&= [f\star L_{r^{-1}}\psi](r^{-1}\boldsymbol{x}) = L_r[f\star L_{r^{-1}}\psi](\boldsymbol{x})
\end{aligned}
\tag{12}
$$

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。

2.3.5. 群等变模块

2.3.5.1. 群等变相关

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

$$
\left[f\star\psi^{(i)}\right](\boldsymbol{x}) = \sum_{\boldsymbol{y}\in\mathbb{Z}^2}\sum_{k=1}^{K^{(l)}} f_k(\boldsymbol{y})\psi_k^{(i)}(-\boldsymbol{x}+\boldsymbol{y}) \tag{13}
$$

注意 $\boldsymbol{x}$ 对应着 $\mathbb{Z}^2$ 上所有平移操作构成的群中的一个群元素 $\mathfrak{g}$，而 $-\boldsymbol{x}$ 对应着它的逆元 $\mathfrak{g}^{-1}$。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：

$$
f^{(1)}(\mathfrak{g}) = \left[f\star\psi^{(i)}\right](\mathfrak{g}) = \sum_{\boldsymbol{y}\in\mathbb{Z}^2}\sum_{k=1}^{K^{(l)}} f_k^{(0)}(\boldsymbol{y})\psi_k^{(0,i)}(\mathfrak{g}^{-1}\boldsymbol{y}) \tag{14}
$$

其中 $\mathfrak{g}\in\mathfrak{G}$，$f$ 和 $\psi^{(0,i)}$ 都是 $\mathbb{Z}^2$ 到 $\mathbb{R}$ 的函数，但得到的新信号 $\left[f\star\psi^{(i)}\right]$ 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：

$$
f^{(l+1)}(\mathfrak{g}) = \left[f\star\psi^{(i)}\right](\mathfrak{g}) = \sum_{\mathfrak{h}\in\mathfrak{G}}\sum_{k=1}^{K^{(l)}} f_k^{(l)}(\mathfrak{h})\psi_k^{(l,i)}(\mathfrak{g}^{-1}\mathfrak{h}) \tag{15}
$$

其中 $l\geqslant 1$，$\mathfrak{g},\mathfrak{h}\in\mathfrak{G}$，$f_k^{(l)}$ 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：

$$
\begin{aligned}
[[L_{\mathfrak{u}}f]\star\psi](\mathfrak{g}) &= \sum_{\mathfrak{h}\in X}\sum_{k=1}^{K}[L_{\mathfrak{u}}f_k](\mathfrak{h})\psi_k^{(i)}(\mathfrak{g}^{-1}\mathfrak{h}) \\
&= \sum_{\mathfrak{h}\in X}\sum_{k=1}^{K} f_k(\mathfrak{u}^{-1}\mathfrak{h})\psi_k^{(i)}(\mathfrak{g}^{-1}\mathfrak{h}) \\
&= \sum_{\mathfrak{p}\in X}\sum_{k=1}^{K} f_k(\mathfrak{p})\psi_k^{(i)}(\mathfrak{g}^{-1}\mathfrak{u}\mathfrak{p}) \qquad\qquad \mathfrak{p}\leftarrow\mathfrak{u}^{-1}\mathfrak{h} \\
&= \sum_{\mathfrak{p}\in X}\sum_{k=1}^{K} f_k(\mathfrak{p})\psi_k^{(i)}\left((\mathfrak{u}^{-1}\mathfrak{g})^{-1}\mathfrak{p}\right) \\
&= [f\star\psi](\mathfrak{u}^{-1}\mathfrak{g}) = [L_{\mathfrak{u}}[f\star\psi]](g)
\end{aligned}
\tag{16}
$$

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 $f^*(g)=f(g^{-1})$：

$$
\begin{aligned}
[f \star \psi](\mathfrak{g}) &= \sum_{\mathfrak{h} \in X} \sum_{k=1}^{K} f_k(\mathfrak{h}) \psi_k(\mathfrak{g}^{-1}\mathfrak{h}) \\
&= \sum_{\mathfrak{h} \in X} \sum_{k=1}^{K} f_k(\mathfrak{h}) \psi_k(\mathfrak{g}^{-1}\mathfrak{h}) \qquad \mathfrak{p} \leftarrow \mathfrak{g}^{-1}\mathfrak{h} \\
&= \sum_{\mathfrak{p} \in X} \sum_{k=1}^{K} \psi_k(\mathfrak{p}) f_k\Big(\big(\mathfrak{g}^{-1}\big)^{-1}\mathfrak{p}\Big) \\
&= [\psi \star f](\mathfrak{g}^{-1}) = [\psi \star f]^*(\mathfrak{g}).
\end{aligned}
\tag{17}
$$

2.3.5.2. *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：

$$
\big[L_{\mathfrak{g}}[af + bg]\big](u) = [af + bg](\mathfrak{g}^{-1}u) = aL_{\mathfrak{g}}f + bL_{\mathfrak{g}}g \tag{18}
$$

其中 $\mathfrak{g} \in \mathfrak{G}$。对于单点非线性函数 $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$，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 $\mathfrak{g}$ 的等变性：

$$
L_{\mathfrak{g}} \tag{19}
$$

3. 学习进度

3.1. 机器学习理论

3.1.1. Markov Chain Monte Carlo (MCMC)

3.2. 随机过程

本周

3.3. 随机微分方程

本周

4. 问题解决记录

4.1. uv 相关

uv 是基于 Rust 的新一代 Python 包管理器，具有可迁移性强、快速、简单的特点。

4.1.1. Pytorch CUDA 版本的配置

若不特意配置，在通过 uv 在环境中添加或下载 Pytorch 时自动下载的是 CPU 版本，无法享受硬件加速。假设系统中 CUDA 的版本是 11.8，可以在 `pyproject.toml` 中配置

4.2. Typst 相关

4.2.1. 数学公式自动编号

4.3. Python 相关

4.3.1. 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。

4.4. 深度学习相关

5. 下周计划

论文阅读

1. 生成模型
   -
2. 动力学
   -

项目进度

1. 使用神经网络学习生命游戏的演化动力学
   -
2. 微型抗癌机器人在血液中的动力学
   -

理论学习

1. 随机过程课程
   -
2. 随机微分方程
   -