2025 年 10 月 13 日至 10 月 19 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 $L_{\mathfrak{r}}$，神经网络是 F，我期待它可以满足

$$L_{\mathfrak{r}}[F \circ f] = [F \circ [L_{\mathfrak{r}} f]] \tag{1}$$

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

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

本周学习到了 Poisson 过程，以及一般宽平稳随机过程相关函数的诸性质。

3.3. 随机微分方程

本周没有推进。

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

4.2. Typst 相关

4.2.1. 数学公式自动编号

4.3. Python 相关

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

6. 参考资料