计算神经与BrainPy

计算神经科学概述

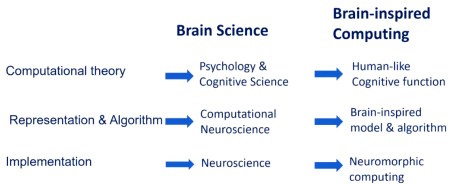
背景

历史

- 1907：LIF 模型
- 1950s：HH模型（重要）
- 1960s：Roll's cable equation（生物模型是为了物理实现，还是必须？）
- 1970s：Amari，Wilson，Cowan et al.
- 1982：Hopfield 模型（Amari-Hopfield 模型）
- 1988：Sejnowski et al. 在Science提出"Computational Neuroscience"

当下正对应物理学的第谷-伽利略时代，大脑工作原理还缺乏清晰的理论。

脑科学的三个层次



神经计算的任务

做出一个模型，可以进行类脑的应用，那么说大脑真的是这样的工作的。

目标与挑战

两大目标

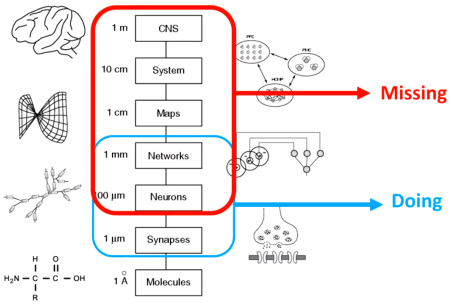
计算神经科学是**脑科学**到**类脑智能**的桥梁：
（1）用计算建模的方法来阐述大脑功能
（2）发展类脑计算

挑战

基于2个目标，其中"**阐述大脑功能**"的重要性不言自明。
因此，本次课程仅讨论"**类脑计算**'

基于人工智能的成功，**吴教授思考新的尝试**：能不能对端到端，基于数据驱动的**大模型**，再对大模型做一些神经生物动力学的**基本约束**，然后进行top down的方式训练网络。

- The missing link: a computational model of higher cognitive function



- 建模工具
  - 要求工具

    当前工具有不足和缺点

    - 效率

      应在并行的计算机设备上高速的仿真运行
    - 整合

      综合建模仿真，训练和分析
    - 灵活

      各尺度的新模型都能很容易地适用
    - 有扩展性

      扩展模型，machine learning
  - BrainPy

    提出解决方案
- 建模举例
  - 图像理解

    （1）图像理解=图像分割+物体识别
    （2）这是一个鸡生蛋问题：如果不做分割，可能识别不出来；如果不能识别，该如何进行分割呢？（AI是非常难的问题）
    （3）对人也是难题。大脑的解决方案是猜测与印证(analysis-by-synthesis)。

    Reverse Hierarchy Theory：视觉2个同路，快速和慢速。快速通路进行快速猜测后，把猜测结果反馈给慢速通路，帮助细节局部识别。

    神经计算需要回答的核心问题：
    如，什么是局部和整体特征？如何快速提取整体特征？如何产生的整体猜测？从整体到局部的过程是如何实现的？等等

    （下一级3个是吴教授课题组开展的一些工作）

    - 如何提取整体特征

- 如何从表示空间中形成“整体”假设?
- 如何feedback从整体到局部

BrainPy简介

BrainPy模块

BrainPy Architecture

结构图 包装

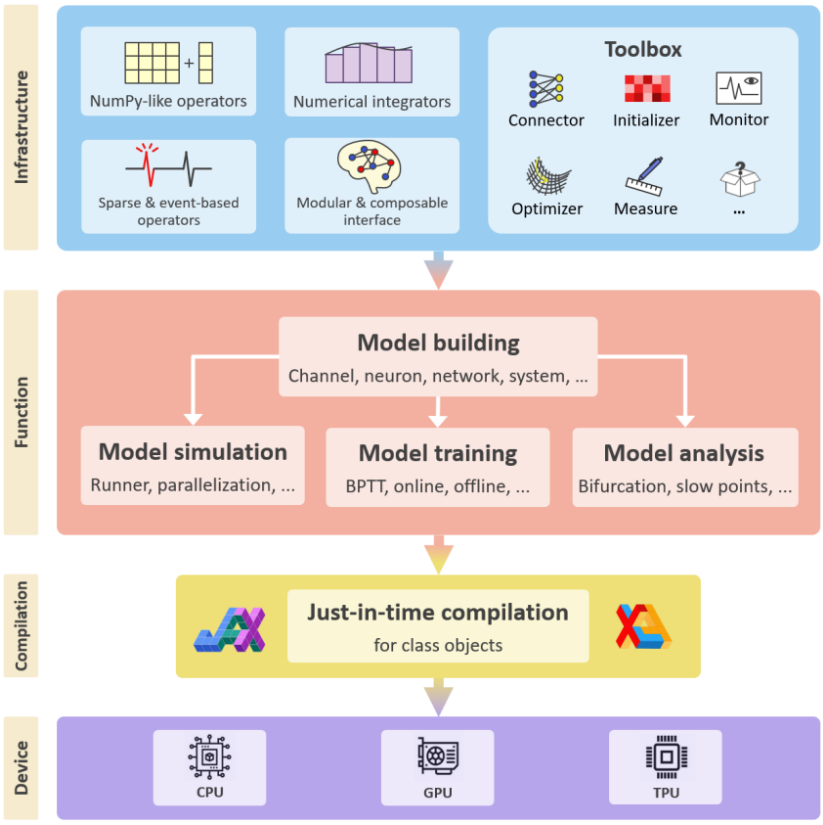
注：这种包装模式非常值得学习！

基础架构Infrastructure
算子，工具包等

函数功能Functions
建模(从微观到宏观)，模型仿真(生物动力学)，模型训练(调参)，模型分析(产生行为的原因)

即时编译Just-in-time compilation
底层提速

硬件设备Devices



- 主要特点
  - 稠密运算算子

    dense operator

    - **NumPy 数组运算**

```
import numpy as np
# 创建数组
>>> a =  np.zeros((2, 2))
>>> a
array([[0., 0.],
       [0., 0.]])
# 数组原位操作
>>> a[0] += 5.
# 数组内置函数
>>> a.max()
5.
# 线性代数函数
>>> np.dot(a, np.ones((2, 1)))
array([[10.],
       [ 0.]])
```

    BrainPy 数组运算

```
import brainpy.math as bm
# 创建数组
>>> a = bm.zeros((2, 2))
>>> a
JaxArray(DeviceArray([[0., 0.],
                      [0., 0.]]))
# 数组原位操作
>>> a[0] += 5.
# 数组内置函数
>>> a.max()
DeviceArray(5., dtype=float32)
# 线性代数函数
>>> bm.dot(a, bm.ones((2, 1)))
JaxArray(DeviceArray([[10.],
                      [ 0.]]))
```

    - **NumPy 随机数运算**

```
# 正态分布
>>> np.random.normal(size=(2, 1))
array([[-0.81543646],
       [ 1.20518382]])
# 均匀分布
>>> np.random.uniform(size=(2, 1))
array([[0.79619099],
       [0.48682065]])
```

    BrainPy 随机数运算

```
# 正态分布
>>> bm.random.normal(size=(2, 1))
JaxArray(DeviceArray([[-0.2434824 ],
                      [-0.16923107]]))
# 均匀分布
>>> bm.random.uniform(size=(2, 1))
JaxArray(DeviceArray([[0.37081087],
                      [0.998075  ]]))
```

  - 专用的算子

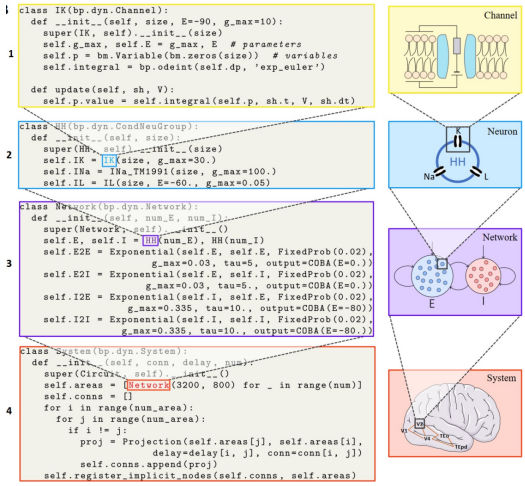
    Dedicated operators
    应用脑动力学稀疏连接属性与事件驱动的计算特征
    将大脑动力学模拟的复杂性降低几个数量级
    减少冗余计算

  - 数值积分器

    Numerical Integrators
    常微分方程(Ordinary differential equations): brainpy.odeint
    随机微分方程(Stochastic differential equations): brainpy.sdeint
    分数阶微分方程(Fractional differential equations): brainpy.fdeint
    时滞微分方程Delayed differential equations 对大脑而言，信号的传递需要时间

  - 模块化和可组合性

    Modular and composable



- 面向对象的即时编译

  Object-oriented JIT compilation
  BrainPy提供面向对象的转换(transformation)：
  brainpy.math.jit
  brainpy.math.grad
  brainpy.math.for_loop
  brainpy.math.ifelse

编程基础

即时编译Just-in-Time compilation

即时编译可以提高运行速度。
(1)类对象必须继承自brainpy.BrainPyObject，它是BrainPy的基类，它的方法会被自动JIT编译
(2)**所有与时间相关的变量必须定义为brainpy.math.Variable**

用户不需要知道JIT的细节也可用。但有一个缺点，不容易debug，这需要关掉turn off JIT

```
class LogisticRegression(bp.BrainPyObject):
  def __init__(self, dimension):
      super(LogisticRegression, self).__init__()

      # parameters
      self.dimension = dimension

      # variables
      self.w = bm.Variable(2.0 * bm.ones(dimension) - 1.3)

  def __call__(self, X, Y):
      u = bm.dot(((1.0 / (1.0 + bm.exp(-Y * bm.dot(X, self.w))) - 1.0) * Y), X)
      self.w.value = self.w - u # in-place update
```

数据算子

array静态数组，随时间不变
variable动态变量，随时间改变

未标记为**动态变量**的数组将被JIT编译为**静态数组**，对静态数组的修改在JIT编译环境中无效。

- array

  静态数组

  BrainPy array转换成JAX array
  .value
  即BrainPy array中存的值就是JAX array

- variable

  动态变量

  转换成动态变量
  bm.Variable()

- in-place updating

  原地更新

  索引/切片Indexing and slicing
  增量赋值Augmented assignment
  对value赋值：.value（推荐，常用）
  用update赋值（不推荐）

控制流

条件语句

if-else

若判断的值不是基于变量的值，正常
若if判断的值是基于variable的，报错

解决方案下一级

- brainpy.math.where

  ```
  a = 1.
  bm.where(a < 0, 0., 1.)
  ```

  ()中参数：

  1. if判断的部分，如果a<0
  2. True执行的部分，0.
  3. False执行的部分，1.

- brainpy.math.ifelse

```
def ifelse(condition, branches, operands):
  true_fun, false_fun = branches
  if condition:
    return true_fun(operands)
  else:
    return false_fun(operands)
```

循环语句

for loop

```
def for_loop_function(body_fun, xs):
  ys = []
  for x in xs:
    results = body_fun(x)
    ys.append(results)
  return ys
```

- while loop

```
while cond_fun(x):
    x = body_fun(x)
```

- Python和基础模块
  - Python
    - 基本元素

      1.变量variable
      2.关键词keyword（变量不能取成关键词）
      3.算子operator（+-*/，**，Bool运算，与and或or非not）
      4.模块module
      5.控制流Control statements
      6.函数
    - 数据类型
      - List

        可修改的，中括号[]

        获得对应元素：myList[0]
        切片：myList[0:3]
        列表运算：myList*2，myList+myList2
        元素替换：myList[0]=“hello”
        copy一个列表：myList2=myList[:]
      - Tuple

        不可修改，小括号()
      - Dictionary

        {}
    - Class类

      继承父类
      - 属性：attribute
      - 方法：method

- NumPy

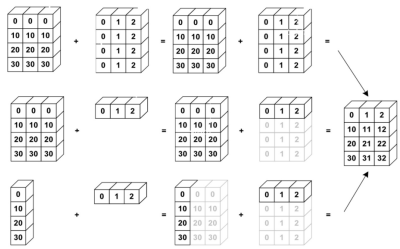
  处理高维数组和矩阵运算
  是非常基础的模块，后续很多模块是基于Numpy开发的

  - Array

    高维数组array
    零矩阵np.zeros((2,2))
    1矩阵np.ones(3,3)
    单位阵np.eye(3)，迹np.trace(a)

    例子：a=np.arange(10).reshape(2,5)
    a.ndim查看维数
    a.shape查看array结构
    a.size查看元素个数
    a.T转置
    a.dtype查看数据类型
    a.reshape注意应该原元素个数=reshape后的元素个数，即总维度不变。

    - array broadcasting：

    - 规约操作

      .sum(axis=0)对array的第1个维度进行压缩，若axis=1则是第2维度压缩
      .cumsum
      .max
      .min

    - Slicing

      a[0:3,:]不同维度间的切片用“，”分开就可以了。

    - 矩阵操作

      np.row_stack([a,a])
      np.column_stack([a,a])

  - Linear algebra

- 线性代数

```
import numpy.linalg

qr              ## Computes the QR decomposition
cholesky        ## Computes the Cholesky decomposition
inv(A)          ## Inverse
solve(A,b)      ##  Solves Ax = b for A full rank
lstsq(A,b)      ##  Solves arg minx ||Ax - b||2
eig(A)          ##  Eigenvalue decomposition
eigvals(A)      ##  Computes eigenvalues
svd(A, full)    ## Singular value decomposition
pinv(A)         ##  Computes pseudo-inverse of A
```

- 傅里叶变换
  -

```
import numpy.fft
fft  ## 1-dimensional DFT
fft2 ## 2-dimensional DFT
fftn ## N-dimensional DFT
ifft ## 1-dimensional inverse DFT (etc.)
rfft ## Real DFT (1-dim)
```

- 随机和分布

  import numpy.random
  beta
  binomial
  chisquare
  exponential
  dirichlet
  gamma
  laplace
  lognormal

  -

```
import numpy.random

rand(d0,d1,...,dn)       ## Random values in a given shape
randn(d0, d1, ...,dn)    ## Random standard normal
randint(lo, hi, size)    ## Random integers [lo, hi)
choice(a, size, repl, p) ## Sample from a
shuffle(a)               ## Permutation (in-place)
permutation(a)           ## Permutation (new array)
```

- SciPy

  SciPy是基于NumPy Array的一个用于的算法和数学工具的模块。

  scipy.linalg 线性代数（linear algebra）
  scipy.stats 统计（statistics）
  scipy.optimize 最优化（optimization）
  scipy.sparse 稀疏矩阵（sparse matrices）
  scipy.signal 信号处理（signal processing）