Hodgkin-Huxley神经元模型（Programming）

Dynamic Programming Basics

- integrators
  - $\tau \frac{dw}{dt} = v + a - bw$
  - $\frac{dv}{dt} = v - \frac{v^3}{3} - w + I_{ext}$
  - ```
    dw = lambda w,t,V,a,b,w: V+a-b*w
    dV = lambda V,t,w,Iext: V-V*V*V/3-w+Iext
    joint_eq = bp.JointEq(dw,dV)
    integral = bp.odeint(joint_eq,method = 'rk2')
    runner = bp.integrators.IntegratorRunner(
        integral,
        monitors=['V'],
        inits=dict(V=0., w=0.),
        args=dict(a=a, b=b, tau=tau, Iext=Iext),
        dt=0.01
    )
    ```
- Dynamical System
  - 模型更新的规则
  - 含有state的model
    - dynamic system描述的是state的变化规则
    - $S(t+dt) = F(S(t), input, t, dt)$
  - 不含有state的model(input依赖)
    - $y = F(x, t)$
  - ForLoop运行Dynamical System
    - ```
      class YourDynamicalSystem(bp.DynamicalSystem)
          def update(self,x)
              pass
      # 获取bp.share内t的值
      bp.share.loda("t")
      # 修改bp.share内t和dt的值
      bp.share.save(t=100,dt=0.1)

      inputs = bp.inputs.section_input([0., 6.0, 0.], [100., 200., 100.])
      indices = np.array(inputs.size)

      def run(i,x):
          neu.step_run(i,x)
          return neu.V.value
      ```

```
vs = bm.for_loop(run,(indices,inputs),progress_bar = True)
```

- DSRunner运行Dynamical System

```
# target: 目标网络
instance_of_dynamical_system = net

# inputs: 输入
# static_inputs: 不随时间变化的inputs
static_inputs = [('E.input',20.),('I.input',20.)]
# iterable_inputs: 随时间发生变化的inputs
# 0-100ms: inputs = 0
# 100-1100ms: inputs = 20
# 1100-1200ms: inputs = 0
# length: 总时间
I,length = bp.inputs.section_input(values=[0,20.,0],
                                   duration=[100,1000,100],
                                   return_length=True,
                                   dt=0.1)
iterable_inputs = [('E.input',I,'iter'),('I.input',I,'iter')]

# monitors: 程序运行过程中监控的变量值
# 监控1号、2号和3号E神经元的spike以及所有E神经元的V
interested_variables_to_monitor = [('E.spike',[1,2,3]),'E.V']

runner = DSRunner(target = instance_of_dynamical_system,
                  inputs = inputs_for_target_DynamicalSystem,
                  monitors = interested_variables_to_monitor,
                  dyn_vars = dynamical_changed_variables,
                  jit = enable_jit_or_not,
                  progress_bar=report_the_running_progress,
                  numpy_mon_after_run = transform_into_numpy_ndarray
                 )
# 运行
runner.run(duration = simulation_time_length,
           inputs = input_data,
           reset_state = whether_reset_the_model_states,
           shared_args = shared_arguments_across_different_layers,
           progress_bar = report_the_running_progress,
           eval_time = evaluate_the_running_time
          )
# 查看监控变量（时间和E群体的spike）
bp.visualize.raster_plot(runner.mon.ts,runner.mon['E.spike'],show=True)
```

Run a built-in HH model

```
import brainpy as bp
import brainpy.math as bm
```

```
# 多个神经元具有不同的输入
current, length = bp.inputs.section_input(values=[0.,bm.asarray([1., 2., 4., 8.,
10., 15.]), 0.],
                                          duration=[10, 2, 25],
                                          return_length=True)
# 实例化 定义size
hh_neurons = bp.neurons.HH(current.shape[1])

runner = bp.DSRunner(hh_neurons,
                     monitors=['V','m','h','n'],
                     inputs=('input',current,'iter')
                    )
runner.run(length)
bp.visualize.line_plot(runner.mon.ts,runner.mon.V,plot_idx=[0,1,2],show=True)
```

Build a HH model from scratch

```
import barinpy as bp
import brainpy.math as bm

class HH(bp.dyn.NeuDyn)
    def __init__(self,size,
                 ENa=50., gNa=120.,
                 EK=-77., gK=36.,
                 EL=-54.387, gL=0.03,
                 V_th=0., C=1.0, T=6.3):
        ## 调用HH的父类的初始化操作
        super(HH, self).__init__(size=size)
        # 定义神经元参数
        self.ENa = ENa
        self.EK = EK
        self.EL = EL
        self.gNa = gNa
        self.gK = gK
        self.gL = gL
        self.C = C
        self.V_th = V_th
        self.T_base = 6.3
        self.phi = 3.0**((T-self.T_base)/10.0)
        # 定义神经元变量
        self.V = bm.Variable(-70.68 * bm.ones(self.num))
        self.m = bm.Variable(0.0266 * bm.ones(self.num))
        self.h = bm.Variable(0.772 * bm.ones(self.num))
        self.n = bm.Variable(0.235 * bm.ones(self.num))
        self.input = bm.Variable(bm.zeros(self.num))
        self.spike = bm.Variable(bm.zeros(self.num, dtype = bool))
        self.t_last_spike = bm.Variable(bm.ones(self.num) * -1e7)
        # 定义积分函数
        self.integral = bp.odeint(f=self.derivative, method = 'exp_auto')

    @property
```

```
    def derivative(self):
        return bp.JointEq(self.dV, self.dm, self.dh, self.dn)
    # Iext为input
    def dV(self, V, t, m, h, n, Iext):
        I_Na = (self.gNa * m ** 3.0 * h) * (V - self.ENa)
        I_K = (self.gK * n ** 4.0) * (V - self.EK)
        I_leak = self.gL * (V - self.EL)
        dVdt = (-I_Na - I_K - I_leak + Iext) / self.C
        return dVdt

    def dm(self, m, t, V):
        alpha = 0.1 * (V + 40) / (1-bm.exp(-(V + 40) / 10))
        beta = 4.0 * bm.exp(-(V + 65) / 18)
        dmdt = alpha * (1 - m) - beta * m
        return self.phi * dmdt

    def dh(self, h, t, V):
        alpha = 0.07 * bm.exp(-(V + 65) / 20.)
        beta = 1 / (1 + bm.exp(-(V + 35) / 10.))
        dhdt = alpha * (1 - h) - beta * h
        return self.phi * dhdt

    def dn(self, n, t, V):
        alpha = 0.01 * (V + 55) / (1 - bm.exp(-(V + 55) / 10))
        beta = 0.125 * bm.exp(-(V + 65) / 80)
        dndt = alpha * (1 - n) + beta * n
        return self.phi * dndt

    def update(self, x=None):
        t = bp.share.load('t')
        dt = bp.share.load('dt')
        # 计算更新后的值
        V, m, h, n = self.integral(self.V, self.m, self.h, self.n, t, self.input,
dt=dt)
        # 判断是否发生动作电位（t时刻V值小于阈值，t+dt时刻V值大于阈值）
        self.spike.value = bm.logical_and(self.V < self.V_th, V > = self.V_th)
        self.t_last_spike.value = bm.where(self.spike, t, self.t_last_spike)
        # 更新变量的值
        self.V.value = V
        self.m.value = m
        self.h.value = h
        self.n.value = n
        # 重置输入
        self.input[:] = 0.

current, length = bp.inputs.section_input(values = [0., bm.asarray([1., 2., 4., 8.,
10. ,15.]), 0.],
                                          duration = [10, 2, 25],
                                          return_length = True)
hh_neurons = HH(current.shape[1])
runner = bp.DSRunner(hh_neurons, monitors = ['V','m','h','n'], inputs =
('input',current,'iter'))
```

```
runner.run(length)
```

Customize a conductance-based model

- 构建离子通道的类-构建神经元的类（从底层到高层）