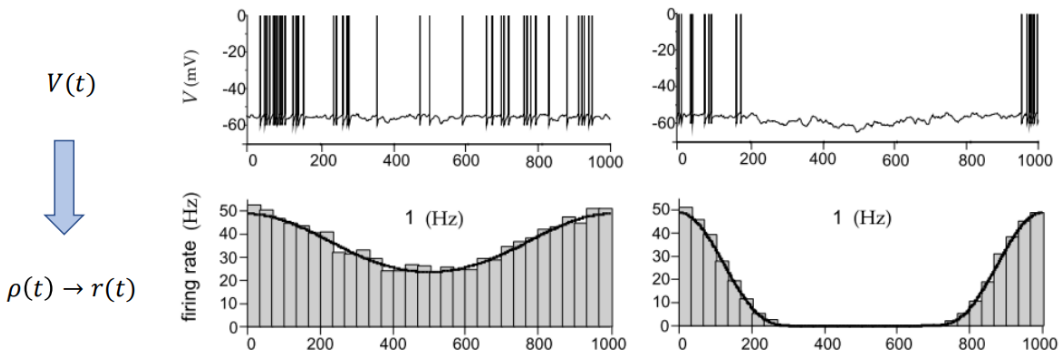
Recurrent Neural Netwroks

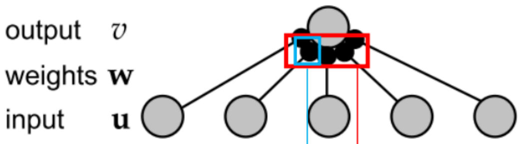
From SNN to rate-based model

- SNN descrete (导数太大)rate-base:发放的概率



  - Adapted from Chance, 2000

- From SNN to rate-based model



$$I_v^i(t) = \sum_{t_i^j<t} K(t-t_i^j) = \int_{-\infty}^{t} d\tau K(t-\tau)\rho_i(\tau)$$

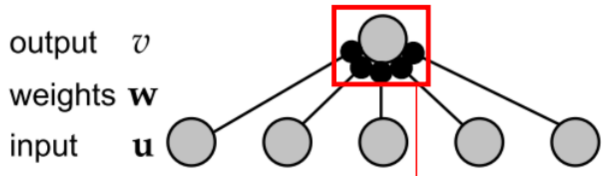
$$K(t) = \frac{1}{\tau_s} e^{-\frac{t}{\tau_s}}$$

$$I_v(t) = \sum_i w_i I_v^i(t) = \sum_i w_i \int_{-\infty}^{t} d\tau K(t-\tau)\rho_i(\tau)$$

$$\approx \sum_i w_i \int_{-\infty}^{t} d\tau K(t-\tau) r_i(\tau)$$

$$\tau_s \frac{dI_v}{dt} = -I_v + \sum_i w_i r_i$$

  - Adapted from Peter Dayan and L.F. Abbott, 2001
  - 突触后电流和突触前的fire rate建立关系
  - 上标j为发放的index



$$\tau_s \frac{dI_v}{dt} = -I_v + \sum_i w_i r_i$$

$$\tau_m \frac{dr_v}{dt} = -r_v + g(I_v)$$

$\tau_s \ll \tau_m \rightarrow \ I_v = \sum_i w_i r_i$ $\qquad$ $\tau_m \ll \tau_s \rightarrow \ r_v = g(I_v)$

$$\tau_m \frac{dr_v}{dt} = -r_v + g(\sum_i w_i r_i) \qquad \tau_m \frac{dI_v}{dt} = -I_v + \sum_i w_i g(I_i)$$

-
- 下面两个是rate-based model，简化了计算

- From rate-based model to general dynamic system

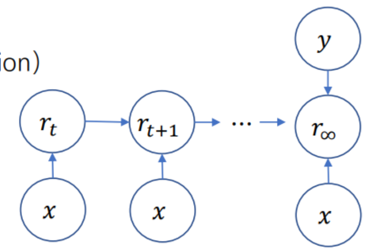
$$\tau_m \frac{dr_v}{dt} = -r_v + g(\sum_i w_i r_i)$$

$$\frac{dr}{dt} = F(r, w, x, y)$$

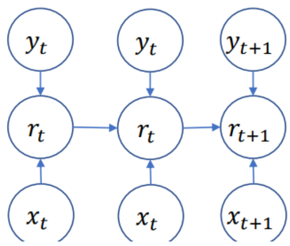
w : parameters
x : input
y : target

$r \in R^n, w \in R^m$

Fixed point representation
- Feedforward model (backpropagation)
- Energy-based model
(Hopfield network, diffusion model)
- Deep Equilibrium Models
- Attractor neural networks



Trajectory representation
- backpropagation through time (BPTT) models (e.g. LSTM)
- Real time recurrent learning (RTRL) models



-
- 用下面那种形式，用更general的形式
- 如何定义输出，t1时刻r作为输出(Trajectory)/r∞达到稳态(fixed point)
- w是连接权重

- Fixed point Representation

$$\frac{dr}{dt} = F(r,w,x,y) \qquad r \in R^n, w \in R^m$$

$$r_\infty = r^* \qquad 0 = F(r^*,w,x,y)$$

Gradient based learning:

$$loss\ function = l(r,y)$$

$$\frac{dl}{dw} = \frac{\partial l}{\partial r}\Big|_{r^*}\frac{dr^*}{dw}$$

$$\frac{d0}{dw} = \frac{dF(r^*,w,x,y)}{dw} = \frac{\partial F}{\partial r}\Big|_{r^*}\frac{dr^*}{dw} + \frac{\partial F}{\partial w}$$

$$J(r^*) = \frac{\partial F}{\partial r}\Big|_{r^*} \qquad \frac{\partial l}{\partial r^*} = \frac{\partial l}{\partial r}\Big|_{r^*}$$

$$\frac{dr^*}{dw} = -J^{-1}\frac{\partial F}{\partial w}$$

$$\frac{dl}{dw} = -\frac{\partial l}{\partial r^*}J^{-1}\frac{\partial F}{\partial w}$$

$$\frac{dv}{dt} = vJ + \frac{\partial l}{\partial r^*}$$

$$v^* = -\frac{\partial l}{\partial r^*}J^{-1}$$

$$\frac{dl}{dw} = v^*\frac{\partial F}{\partial w}$$

$$\frac{dr}{dt} = F(r,w,x,y) + \lambda\left(\frac{\partial l}{\partial r}\right)^T$$

$$\frac{d0}{d\lambda} = \frac{dF(r^*,w,x,y)}{d\lambda} + \left(\frac{\partial l}{\partial r}\right)^T = J\frac{dr^*}{d\lambda} + \left(\frac{\partial l}{\partial r}\right)^T$$

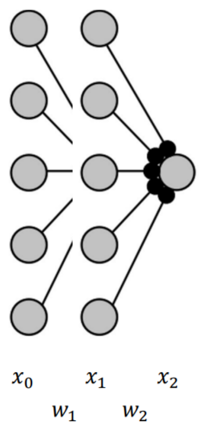
$$\left(\frac{dr^*}{d\lambda}\right)^T = -\frac{\partial l}{\partial r}J^{-T}$$

If $J^{-T} = J^{-1} \Leftrightarrow$ exist E , $s.t.$ F $= \frac{\partial E}{\partial r}$

$$\frac{dl}{dw} = \left(\frac{dr^*}{d\lambda}\right)^T\frac{\partial F}{\partial w}$$

- Energy-based model

- back propogation



$$x_1 = f(w_1x_0)$$
$$x_2 = f(w_2x_1)$$

$$\frac{dr}{dt} = F(r,w,x,y)$$

$$\frac{dx_1}{dt} = -x_1 + f(w_1x_0)$$

$$\frac{dx_2}{dt} = -x_2 + f(w_2x_1)$$

$$\frac{dl}{dw} = -\frac{\partial l}{\partial r^*}J^{-1}\frac{\partial F}{\partial w}$$

$$\frac{dv}{dt} = vJ + \frac{\partial l}{\partial r^*}$$

$$v^* = -\frac{\partial l}{\partial r^*}J^{-1}$$

$$\frac{dl}{dw} = v^*\frac{\partial F}{\partial w}$$

$$\frac{dl}{d(w_1,w_2)} = -\frac{\partial l}{\partial(x_1,x_2)}J^{-1}\frac{\partial F}{\partial(w_1,w_2)}$$

$$J = \begin{pmatrix} -1 & 0 \\ w_2f'(w_2x_1) & -1 \end{pmatrix}$$

$$\frac{d(v_1,v_2)}{dt} = (v_1,v_2)\begin{pmatrix} -1 & 0 \\ w_2f'(w_2x_1) & -1 \end{pmatrix} + \frac{\partial l}{\partial(x_1,x_2)}$$

$$v_2^* = \frac{\partial l}{\partial x_2}, v_1^* = \frac{\partial l}{\partial x_2}w_2f'(w_2x_2)$$

$$\frac{dl}{d(w_1,w_2)} = \begin{pmatrix} \frac{\partial l}{\partial x_2}w_2f'(w_2x_2) \\ \frac{\partial l}{\partial x_2} \end{pmatrix}\begin{pmatrix} \frac{\partial f(w_1x_0)}{\partial w_1} & 0 \\ 0 & \frac{\partial f(w_2x_1)}{\partial w_2} \end{pmatrix}$$

- Trajectory represemtation

Trajectory representation

PEKING UNIVERSITY

$$\frac{dr}{dt} = F(r,w,x,y) \qquad r \in R^n, w \in R^m$$

$$loss\ function\!: l = \int \alpha_t l_t(r_t,y_t)dt$$

$$\frac{dl_t(r_t,y_t)}{dw} = \frac{\partial l_t}{\partial r_t}\frac{dr_t}{dw}$$

$$\frac{dr_t}{dw} = \frac{d}{dw}\int_0^t dr_\tau = \frac{d}{dw}\int_0^t\frac{dr_\tau}{d\tau}d\tau$$
$$= \frac{d}{dw}\int_0^t F(r,w,x,y)d\tau$$
$$= \frac{d}{dw}\int_0^t F(r,w,x,y)d\tau$$
$$= \int_0^t\frac{dF(r,w,x,y)}{dw}d\tau$$

$$p_t = \frac{dr_t}{dw}$$

$$\frac{dp_t}{dt} = \frac{dF(r,w,x,y)}{dw} = J(r_t)p_t + \frac{\partial F}{\partial w}$$

Real time recurrent learnir
Time: $O(n^2m * T)$
Space: $O(mn + n^2)$

$$\frac{dp_t}{dt} = J(r_t)p_t + \frac{\partial F}{\partial w}$$

$$p_t = [J(r_{t-1})\Delta t + 1]p_{t-1} + \frac{\partial F(r_{t-1})}{\partial w}\Delta t$$

$$p_t = \frac{\partial r_t}{\partial r_{t-1}}p_{t-1} + \frac{\partial F(r_{t-1})}{\partial w}\Delta t$$

$$p_t = \frac{\partial r_t}{\partial r_{t-1}}\frac{\partial r_{t-1}}{\partial r_{t-2}}p_{t-1} + \frac{\partial r_t}{\partial r_{t-1}}\frac{\partial F(r_{t-1})}{\partial w}\Delta t + \frac{\partial F(r_{t-1})}{\partial w}\Delta t$$

$$\frac{dr_t}{dw} = p_t = \int_0^t\frac{\partial r_t}{\partial r_\tau}\frac{\partial F(r_\tau,w,x,y)}{\partial w}d\tau$$

$$\frac{dl_t(r_t,y_t)}{dw} = \int_0^t\frac{\partial l_t}{\partial r_t}\frac{\partial r_t}{\partial r_\tau}\frac{\partial F(r_\tau,w,x,y)}{\partial w}d\tau$$

BPTT
Time: $O(n^2T + nmT)$
Space: $O(mn + n^2)$

- 实时同步计算rt
- offline 可以减少时间复杂度

- Example

$$\frac{dr}{dt} = -r + wr + b + x$$

For a input sequence $x_{1:T} \in R$, get the target output $y_{1:T} \in R$

Real time recurrent learning

1. Initial $r_0 = y_0$, $p_0 = 0$
2. For a given sequence $x_{1:T}$, compute $r_{1:T}, p_{1:T}$ according to $\frac{dr}{dt} = -r + wr + b + x$, $\frac{dp_w}{dt} = (-I + w)p_w + r$, $\frac{dp_b}{dt} = (-I + w)p_b + 1$
3. Set $l_t = \frac{1}{2T}(r_t - y_t)^2$, leading to $\Delta w = -\frac{\eta}{T}\sum(r_t - y_t)p_t$, $\Delta b = -\frac{\eta}{T}\sum(r_t - y_t)p_b$

Pseudo code of BPTT

1. Initial $r_0 = y_0$, $p_0 = 0$
2. For a given sequence $x_{1:T}$, compute $r_{1:T}$ according to $\frac{dr}{dt} = -r + wr + b + x$
3. Set $l_t = \frac{1}{2T}(r_t - y_t)^2$, leading to $\Delta w = -\eta \sum_t \sum_\tau \frac{1}{T}(r_t - y_t)\frac{\partial r_t}{\partial r_\tau} r$, $\Delta b = -\eta \sum_t \sum_\tau \frac{1}{T}(r_t - y_t)\frac{\partial r_t}{\partial r_\tau}$

-
- Homework

Homework

$$\frac{dr}{dt} = -r + wr + b + x$$

For a input sequence $x_{1:T} \in R$, get the target output $y_{1:T} \in R$

Real time recurrent learning

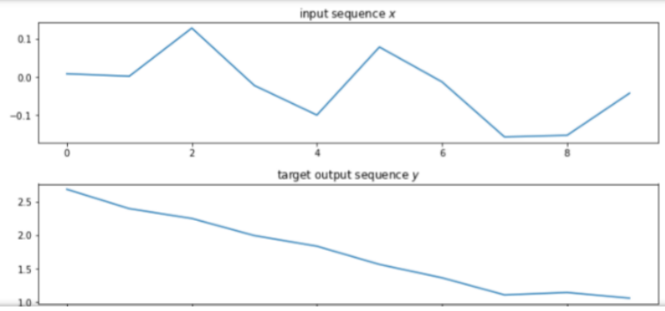
1. Initial $r_0 = y_0$, $p_0 = 0$
2. For a given sequence $x_{1:T}$, compute $r_{1:T}, p_{1:T}$ according to $\frac{dr}{dt} = -r + wr + b + x$, $\frac{dp_w}{dt} = (-I + w)p_w + r$, $\frac{dp_b}{dt} = (-I + w)p_b + 1$
3. Set $l_t = \frac{1}{2T}(r_t - y_t)^2$, leading to $\Delta w = -\frac{\eta}{T}\sum(r_t - y_t)p_t$, $\Delta b = -\frac{\eta}{T}\sum(r_t - y_t)p_b$

1. Train an RNN to generate a sequence

Real time recurrent learning: see Example for models detail

In [2]:
```
###构造数据，x为给定输入序列，y为想要rnn给出的输出序列，序列长度为T
T = 10
x = bm.random.normal(0,0.1,size=(T,))
y = bm.exp(bm.linspace(1,0,T)) + bm.random.normal(0,0.1,size=(T,))
y0 = y[0]

###可视化输入与输出
plt.figure(figsize=(T,5))
plt.subplot(2,1,1)
plt.plot(bm.arange(T),x)
plt.title('input sequence $x$')
plt.subplot(2,1,2)
plt.plot(bm.arange(T),y)
plt.title('target output sequence $y$')
plt.tight_layout()
plt.show()
```



-

Homework

$$\frac{dr}{dt} = -r + wr + b + x$$

For a input sequence $x_{1:T} \in R$, get the target output $y_{1:T} \in R$

Real time recurrent learning

1. Initial $r_0 = y_0$, $p_0 = 0$
2. For a given sequence $x_{1:T}$, compute $r_{1:T}, p_{1:T}$ according to $\frac{dr}{dt} = -r + wr + b + x$, $\frac{dp_w}{dt} = (-I + w)p_w + r$, $\frac{dp_b}{dt} = (-I + w)p_b + 1$
3. Set $l_t = \frac{1}{2T}(r_t - y_t)^2$, leading to $\Delta w = -\frac{\eta}{T}\sum(r_t - y_t)p_t$, $\Delta b = -\frac{\eta}{T}\sum(r_t - y_t)p_b$

```
class RNN(bp.DynamicalSystemNS):
    def __init__(self,dt=bm.dt):
        super(RNN, self).__init__(name=None)

        self.r = bm.Variable(bm.zeros(1))
        self.pw = bm.Variable(bm.zeros(1))
        self.pb = bm.Variable(bm.zeros(1))

        self.w = bm.Variable(bm.ones(1))
        self.b = bm.Variable(bm.ones(1))
        self.dt = dt

    def reset_neuron(self,y0):
        self.r = bm.Variable(bm.ones(1)*y0)
        self.pw[0].value = 0
        self.pb[0].value = 0

    def update(self,x):
        dr = ((self.w-1)*self.r + self.b + x)*self.dt
        self.r.value = self.r + dr

        # 这两行需要写出p_w的计算细节
        #

        dpb = ((self.w-1)*self.pb + 1)*self.dt
        self.pb.value = self.pb + dpb

    def train(self,r_seq,pw_seq,pb_seq,y):
        eta = 0.1

        #写出dw的更新法则
        self.w.value = self.w + dw

        #写出db的更新法则
        self.b.value = self.b + db
        return bm.mean(bm.square((r_seq-y)))/2,dw,db


rnn = RNN()
rnn.reset_neuron(y0)
runner = bp.DSRunner(rnn, monitors=['r'])
runner.run(inputs = x)
plt.plot(bm.arange(T), bm.squeeze(runner.mon.r),label = 'RNN_output')
plt.plot(bm.arange(T),y,label = 'target_output')
plt.legend()
plt.show()
```

-

$$\frac{dr}{dt} = -r + wr + b + x$$

For a input sequence $x_{1:T} \in R$, get the target output $y_{1:T} \in R$

Real time recurrent learning

1. Initial $r_0 = y_0$, $p_0 = 0$
2. For a given sequence $x_{1:T}$, compute $r_{1:T}, p_{1:T}$ according to $\frac{dr}{dt} = -r + wr + b + x$, $\frac{dp_w}{dt} = (-I + w)p_w + r$, $\frac{dp_b}{dt} = (-I + w)p_b + 1$
3. Set $l_t = \frac{1}{2T}(r_t - y_t)^2$, leading to

$\Delta w = -\frac{\eta}{T}\sum(r_t - y_t)p_t$, $\Delta b = -\frac{\eta}{T}\sum(r_t - y_t)p_b$

```
for epoch in range(10):
    rnn.reset_neuron(y0)
    runner = bp.DSRunner(rnn, monitors=['r','pw','pb'])
    runner.run(inputs = x)
    loss,dw,db = rnn.train(bm.squeeze(runner.mon.r),bm.squeeze(runner.mon.pw),bm.squeeze(runner.mon.pb),y)
    print('epoch',epoch,':loss=',loss,dw,db,'w=',rnn.w,)

rnn.reset_neuron(y0)
runner = bp.DSRunner(rnn, monitors=['r'])
runner.run(inputs = x)
plt.plot(bm.arange(T), bm.squeeze(runner.mon.r),label = 'RNN_output')
plt.plot(bm.arange(T),y,label = 'target_output')
plt.legend()
plt.show()
```

From rated-based model to general dynamic system

- Fixed point
- Trajectory