Training Recurrent Neural Networks

董行思

2023-11-25

Outline

1 | A general dynamic system

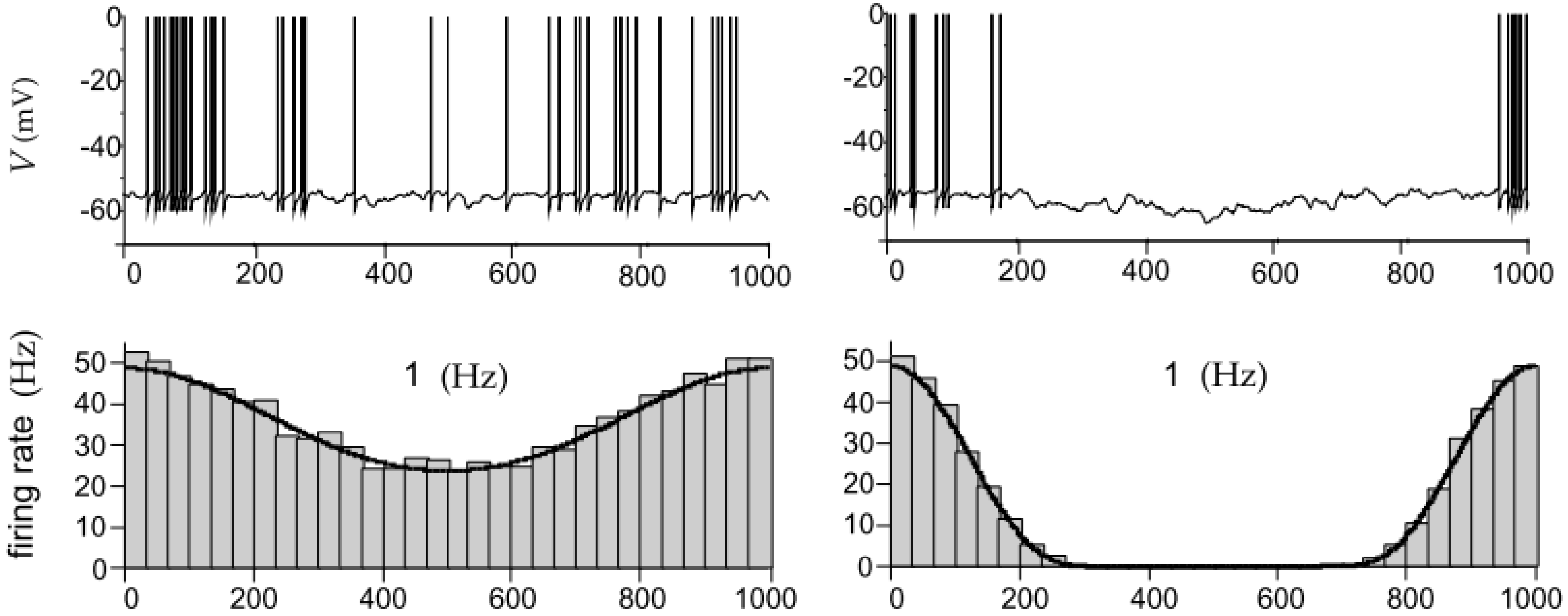
2 | Fixed point representation & learning

3 | Trajectory representation & learning

From SNN to rate-based model

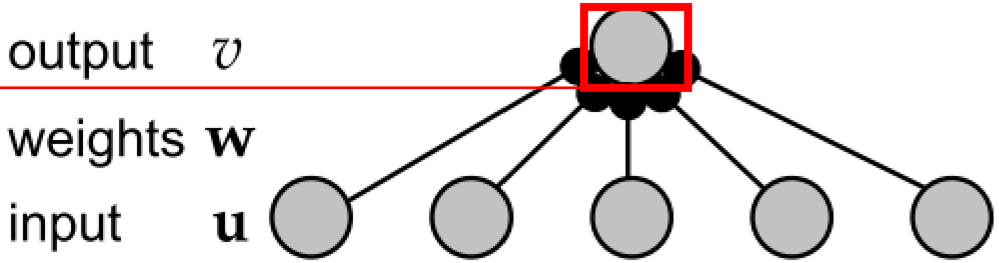
$\rho(t)$

$r(t)$



Adapted from Chance, 2000

From SNN to rate-based model



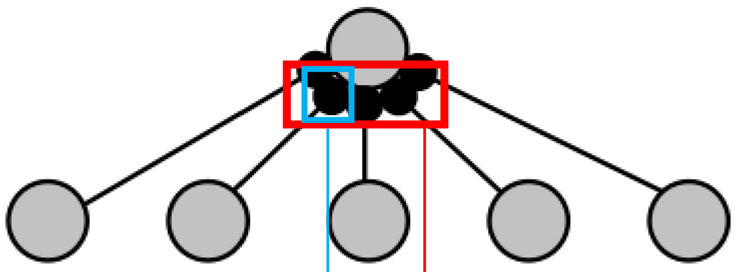
$$\tau_m \frac{du}{dt} = -(u - u_{rest}) + g(I_v)$$

$$when \quad u(t^j) > V_{th}\,, \qquad spike\ at\ time\ t^j$$

$$\rho_i(t) = \sum_{t_i^j < t} \delta(t - t_i^j)$$

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

From SNN to rate-based model



$$\rho_i(t) = \sum_{t_i^j < t} \delta(t - t_i^j)$$

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

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

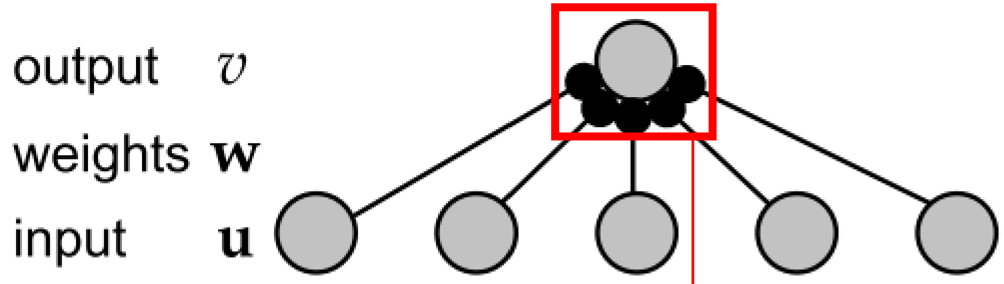
$$I_v(t) = \sum_i w_i I_v^i(t) = \sum_i w_i \int_{-\infty}^{t} d\tau K(t - \tau)\rho_i(\tau)$$

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

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

Adapted from Peter Dayan and L.F. Abbott, 2001

From SNN to rate-based model



$$\begin{cases} \tau_s \frac{dI_v}{dt} = -I_v + \sum_i w_i r_i \\ \tau_m \frac{dr_v}{dt} = -r_v + g(I_v) \end{cases}$$

$\tau_s \ll \tau_m \rightarrow \quad I_v = \sum_i w_i r_i$

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

$\tau_m \ll \tau_s \rightarrow \quad r_v = g(I_v)$

$$\tau_s \frac{dI_v}{dt} = -I_v + \sum_i w_i g(I_i)$$

General dynamic system

$$\begin{cases} \tau_s \dfrac{dI_v}{dt} = -I_v + \sum_i w_i r_i \\ \tau_m \dfrac{dr_v}{dt} = -r_v + g(I_v) \end{cases}$$

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

$$\tau_s \frac{dI_v}{dt} = -I_v + \sum_i w_i g(I_i)$$

$$\frac{dr}{dt} = F_w(r, x)$$

$w : parameters$
$x : input$
$y : target$

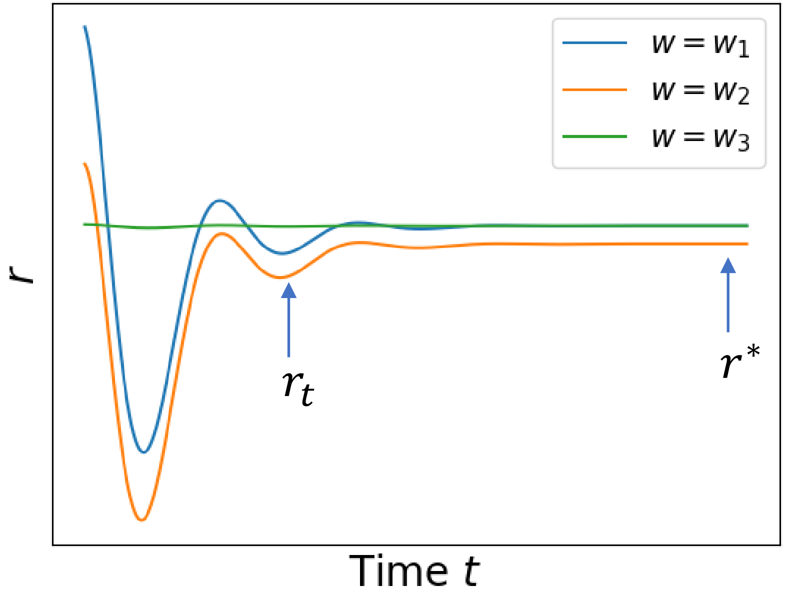
$F_w(r, x) \in C^1$

General dynamic system

$$\frac{dr}{dt} = F_w(r, x)$$

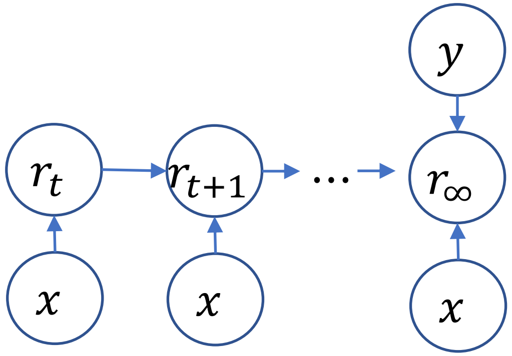
w : *parameters*
x : *input*
y : *target*

$$loss\ function = \begin{cases} l(r^*, y) & 0 = F_w(r^*, x) \\ l(r_t, y) & \end{cases}$$



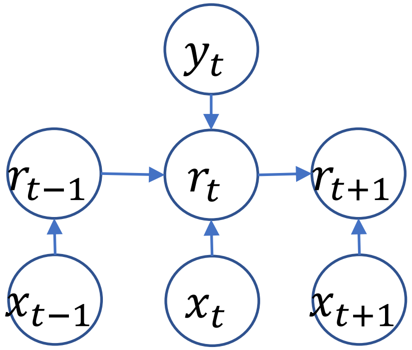
Fixed point representation

- 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



Fixed point representation

$$\frac{dr}{dt} = F_w(r,x) \qquad 0 = F_w(r^*,x)$$

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

Gradient based learning: $\frac{dl}{dw} = \frac{\partial l}{\partial r^*}\frac{dr^*}{dw}$

How to calculate: $\frac{dr^*}{dw}$

$$0 = F_w(r^*,x)$$

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

$$\frac{dr^*}{dw} = -J^{-1}\frac{\partial F}{\partial w} \qquad J = \frac{\partial F}{\partial r}\Big|_{r^*}$$

$$\frac{dl}{dw} = -\frac{\partial l}{\partial r^*}J^{-1}\frac{\partial F}{\partial w} \qquad r \in R^n, w \in R^m$$

$$J \in R^{n*n} \qquad \frac{\partial F}{\partial w} \in R^{n*m}$$

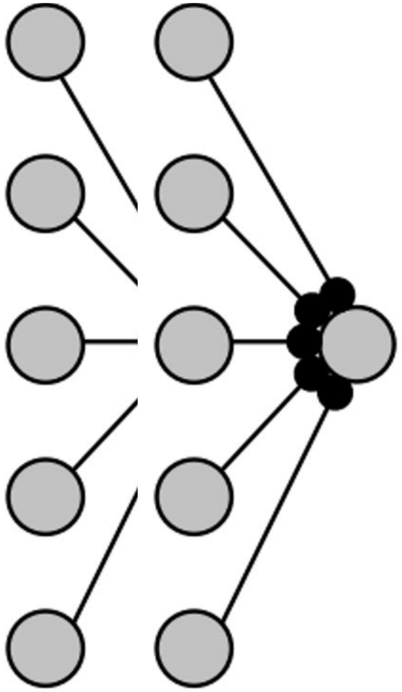
$$O(n^3 + n*m)$$

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

$$v^* = -\frac{\partial l}{\partial r^*}J^{-1} \qquad O(n^2T + n*m)$$

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

Feedforward model



$$r_1 = f_1(w_1 r_0)$$
$$r_2 = f_2(w_2 r_1)$$

$\Longleftrightarrow$

$$\frac{dr_1}{dt} = -r_1 + f_1(w_1 r_0)$$

$$\frac{dr_2}{dt} = -r_2 + f_2(w_2 r_1)$$

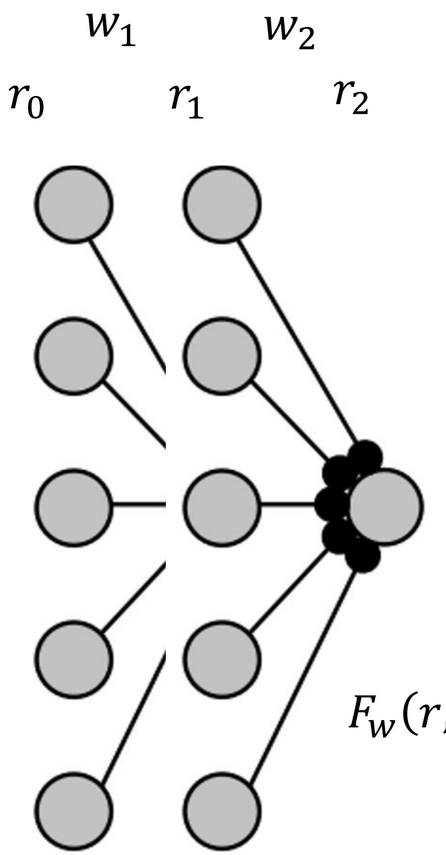
$$\frac{dr}{dt} = F_w(r, x)$$

$$r = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} \qquad F_w(r, x) = \begin{pmatrix} -r_1 + f_1(w_1 r_0) \\ -r_2 + f_2(w_2 r_1) \end{pmatrix}$$

$$\frac{dl}{dw_2} = \frac{\partial l}{\partial r_2}\frac{\partial f_2}{\partial w_2} = \frac{\partial l}{\partial r_2} f_2' r_1$$

$$\frac{dl}{dw_1} = \frac{\partial l}{\partial r_2} f_2' w_2 \frac{\partial f_1}{\partial w_1} = \frac{\partial l}{\partial r_2} f_2' w_2 f_1' r_0$$

Feedforward model



$$r_1 = f_1(w_1 r_0)$$
$$r_2 = f_2(w_2 r_1)$$

$$\frac{dr}{dt} = F_w(r, x)$$

$$r = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$$

$$F_w(r, x) = \begin{pmatrix} -r_1 + f_1(w_1 r_0) \\ -r_2 + f_2(w_2 r_1) \end{pmatrix}$$

$$\frac{dl}{dw_2} = \frac{\partial l}{\partial r_2} \frac{\partial f_2}{\partial w_2}$$

$$\frac{dl}{dw_1} = \frac{\partial l}{\partial r_2} f_2' w_2 \frac{\partial f_1}{\partial w_1}$$

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

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

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

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

$$\frac{dl}{d(w_1, w_2)} = -\frac{\partial l}{\partial (r_1, r_2)} J^{-1} \frac{\partial F}{\partial (w_1, w_2)}$$

$$J = \begin{pmatrix} -1 & 0 \\ w_2 f_2' & -1 \end{pmatrix}$$

$$\frac{d(v_1, v_2)}{dt} = (v_1, v_2) \begin{pmatrix} -1 & 0 \\ w_2 f_2' & -1 \end{pmatrix} + \frac{\partial l}{\partial (r_1, r_2)}$$

$$v_2^* = \frac{\partial l}{\partial r_2}, v_1^* = \frac{\partial l}{\partial r_2} f_2' w_2$$

$$\frac{dl}{d(w_1, w_2)} = \begin{pmatrix} \frac{\partial l}{\partial r_2} f_2' w_2 \\ \frac{\partial l}{\partial r_2} \end{pmatrix} \begin{pmatrix} \frac{\partial f_1(w_1 r_0)}{\partial w_1} & 0 \\ 0 & \frac{\partial f_2(w_2 r_1)}{\partial w_2} \end{pmatrix}$$

Energy-based model

$$\frac{dr}{dt} = F_w(r,x) \qquad 0 = F_w(r^*,x)$$

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

Gradient based learning: $\frac{dl}{dw} = \frac{\partial l}{\partial r^*}\frac{dr^*}{dw}$

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

$$F_w(r,x) = \begin{pmatrix} -r_1 + f_1(w_1 r_0) \\ -r_2 + f_2(w_2 r_1) \end{pmatrix}$$

$$F_w(r,x) = \frac{\partial}{\partial r}\left[\left(-r_1 + f_1(w_1 r_0)\right)^2 + \left(-r_2 + f_2(w_2 r_1)\right)^2\right]$$

$$= \begin{pmatrix} -r_1 + f_1(w_1 r_0) + \frac{\partial f_1}{\partial r_1}\left(-r_2 + f_2(w_2 r_1)\right) \\ -r_2 + f_2(w_2 r_1) \end{pmatrix}$$

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

$$0 = F_w(r_\lambda^*,x) + \lambda\left(\frac{\partial l}{\partial r}\right)^T\Big|_{r=r_\lambda^*} \qquad r^* = r_{\lambda=0}^*$$

$$\frac{d0}{d\lambda} = \frac{dF_w(r_\lambda^*,x)}{d\lambda} + \left(\frac{\partial l}{\partial r}\right)^T = J\frac{dr_\lambda^*}{d\lambda} + \left(\frac{\partial l}{\partial r}\right)^T$$

$$\left(\frac{dr_{\lambda=0}^*}{d\lambda}\right)^T = -\frac{\partial l}{\partial r}J^{-T}$$

If $J^{-T} = J^{-1} \Leftrightarrow exist\ E \quad s.t.\quad F_w(r,x) = \frac{\partial E}{\partial r}$

$$\left(\frac{dr_{\lambda=0}^*}{d\lambda}\right)^T = -\frac{\partial l}{\partial r^*}J^{-1}$$

$$\frac{dl}{dw} = \left(\frac{dr_{\lambda=0}^*}{d\lambda}\right)^T\frac{\partial F}{\partial w} \qquad \frac{dr_{\lambda=0}^*}{d\lambda} \approx \frac{r_\lambda^* - r^*}{\lambda}$$

Trajectory representation

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

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

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

$$\frac{dr_t}{dw} = \frac{d}{dw}\int_0^t dr_\tau = \frac{d}{dw}\int_0^t \frac{dr_\tau}{d\tau} d\tau$$
$$= \frac{d}{dw}\int_0^t F_w(r,x) d\tau$$
$$= \frac{d}{dw}\int_0^t F_w(r,x) d\tau$$
$$= \int_0^t \frac{dF_w(r,x)}{dw} d\tau$$

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

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

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

SNN

北京大學 PEKING UNIVERSITY

$$\frac{du}{dt} = -V_{th}s + ws + x$$

$$s_t = H(u_t - V_{th})$$

$$\frac{dl_t}{dw} = \frac{\partial l_t}{\partial s_t}\frac{\partial s_t}{\partial u_t}\frac{du_t}{dw}$$

$$p_t = \frac{du_t}{dw} = \int_0^t \frac{d}{dw}(-V_{th}s + ws + x)d\tau$$

$$\frac{dp_t}{dt} = \frac{d}{dw}(-V_{th}s + ws + x) = (w - V_{th})\frac{\partial s_t}{\partial u_t}p_t + s$$

Homework

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

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

Pseudo code of RTRL

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

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

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

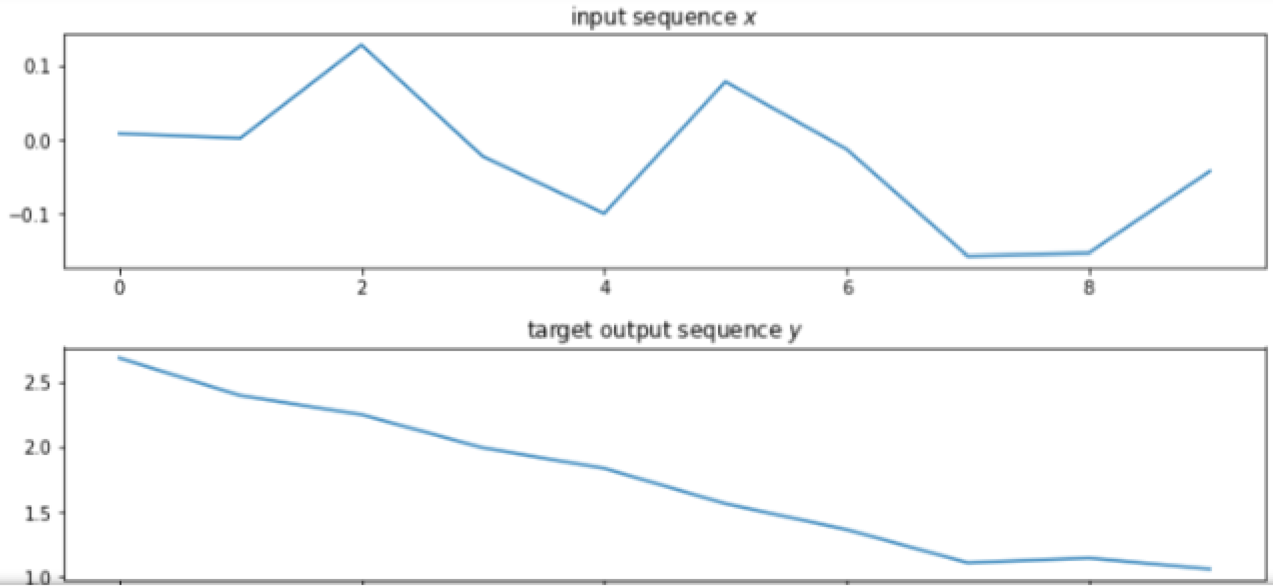
$\Delta w = -\frac{\eta}{T}\sum(r_t - y_t)p_t$, $\Delta b = -\frac{\eta}{T}\sum(r_t - y_t)p_b$

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()
```

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

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

```
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()
```

Q&A