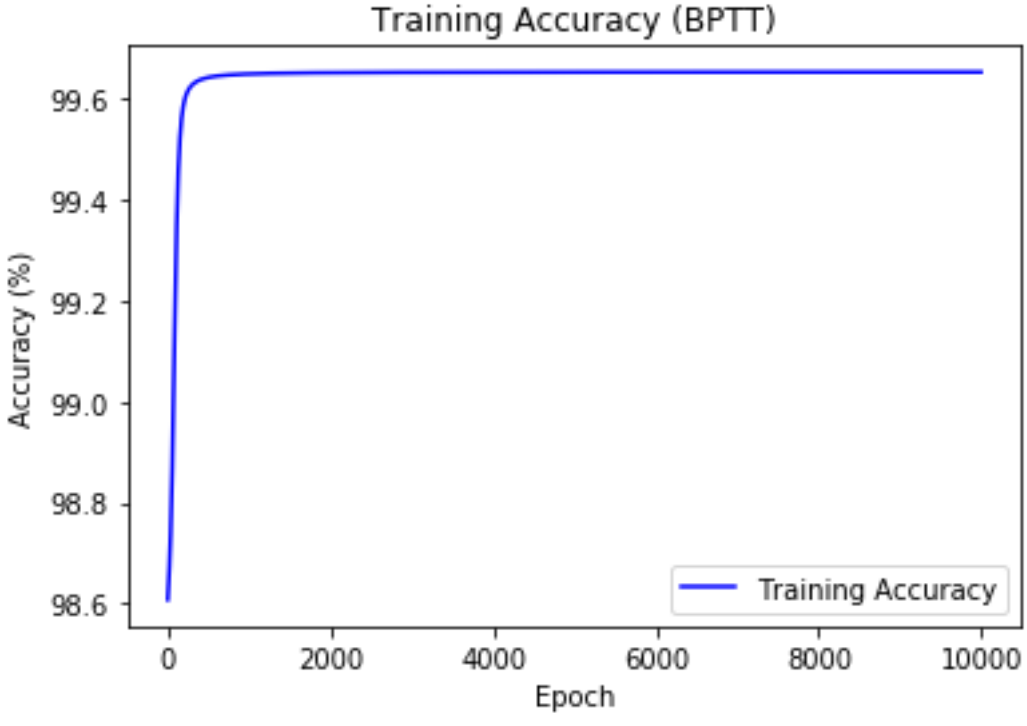
Backpropagation through time (BPTT)

陽明大學不分系二年級張凱博10612012

(一) A plot shows episode rewards of at least 10000 training episodes



(二) Describe data generator

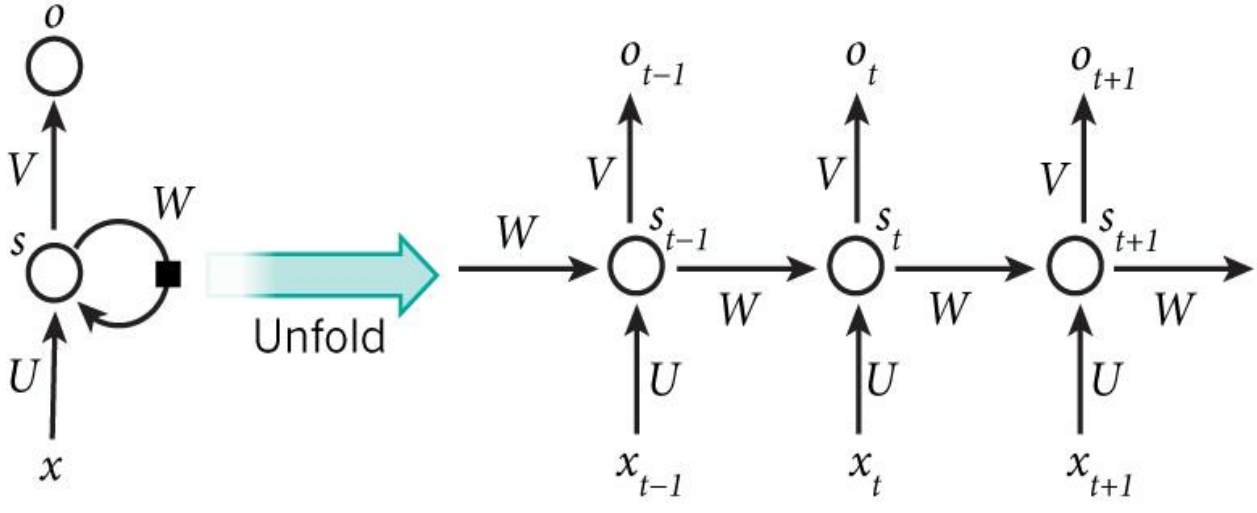
In [2]:
```
import numpy as np
import random
from datetime import datetime
from random import randint
random.seed(10)

x1 = ''.join([str(randint(0, 1)) for i in range(0, 8)])
x2 = ''.join([str(randint(0, 1)) for i in range(0, 8)])
print(x1, x2)

x1_train = np.array([int(i) for i in x1])
x2_train = np.array([int(i) for i in x2])

x_train = np.concatenate((np.transpose(x1_train), np.transpose(x2_train)), axis=0).reshape(8, 2)
y_train = list(bin(int(x1, 2) + int(x2, 2)))[-8:]
y_train = np.array([int(i) for i in y_train]).reshape(8,)
print(x_train, y_train)
print(x_train.shape, y_train.shape)
```

```
01100111 00110010
[[0 1]
 [1 0]
 [0 1]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 0]] [1 0 0 1 1 0 0 1]
(8, 2) (8,)
```

用random創造2個不一樣的字串，分別用變數x1和x2命名，再將它們轉化為整數陣列，但是要將兩者做binary的相加，並且符合等一下輸入RNN (BPTT)的格式，所以先將他們conactenate為8*2的array，令其為x_train，再取每一列的第二個元素做binary addition，所的結果取8位，令其位y_train。

(三) Explain the mechanism of forward propagation

Forward propagation的機制中，除了一次輸入一列2*1的x_train以外，最重要的就是先預設好hidden layer initial state (2, 16)、hidden layer state和state之間的weight (16, 16)、input和hidden layer state之間的參數U (16, 2)、activated hidden layer state和經過softmax後output之間的參數V (16, 16)、output的預設為(2, 16)，經由以下的graphical model 做運算，輸出為o 和 s，分別是時間在t不同情況下的output和state。

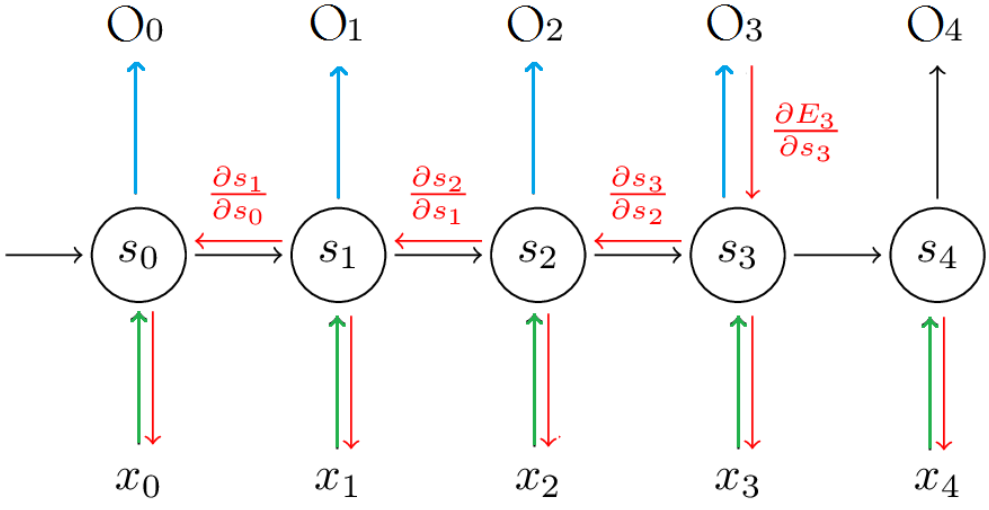


圖源: http://songhuiming.github.io/pages/2017/08/20/build-recurrent-neural-network-from-scratch/

(四) Explain the mechanism of BPTT

在Backpropogation through time中，利用cross entropy當作loss function，使用形式為L(y,o) = log(C)，再執行BPTT之前，要先執行過一遍forward propagation並且將其output和hidden state記錄起來，方便之後做BPTT時的取出，BPTT的目的是要找出RNN中最佳的U、V、W並且minize loss function，再這裡是利用SGD，就是將所有的training example做iteration並且在每一次backpropagation的時候利用loss function對U、V、W做gradient，在一輪一輪的backpropagation中，一步步minize loss function。

在程式碼中，最主要解決的問題是計算loss function對U、V、W做gradient，但是RNN的backpropagation相對一般普通神經網絡的backpropagation最不同的是，RNN的state是會隨時間點的不同而有不同的值，也因此在每個時間點會有不同的output，簡單而言RNN的backpropagation示意圖如下:



由手寫推導之後，先令**delta_o[np.arange(T), y] - = 1**以計算predicted result和ground truth的差，再令**delta_t = self.V.T.dot(delta_o[t]) * (1 - (s[t]**2))**會使得python的版面較簡潔。第一個要計算的梯度dLdV不隨時間改變，所以程式碼可以寫在backpropagation的第一個for loop外面一層，記為**dLdV += np.outer(delta_o[t], s[t].T)**。第二個要計算的梯度dLdW和第三個要計算的梯度dLdU會被時間影響，所以要寫在第二個for loop裡面，因為已有在紙本上推導，再利用上述變數delta_t帶入dLdW和dLdU，最後再每步delta_t即可，每個epoch return的dLdU, dLdV, dLdW都會再乘上learning rate，以minize loss function。

(五) Describe how the code work (the whole code)

一開始先將生成的binary data放入啟動整個forward和backward的function，一開始會先進行feed-forward propagation以獲得每個時間點所生成的output和state，方便之後做BPTT時需要，再計算每以個epoch的loss並記錄下來，之後進行BPTT，在每一個epoch更新dLdU, dLdV, dLdW，當約執行到2700輪的時候，loss值在小數點前三位即不再變動，最後將100-loss(%)，得到accuracy，即可作圖。

(六) More you want to say

在這次lab碰到最大的困難是要手刻一個BPTT的Neuron Network，這非常吃線性代數的數學底子...所以我也是花了很多時間慢慢將推導的公式看過一遍，並且參考網路資源將之轉換為程式碼實現。