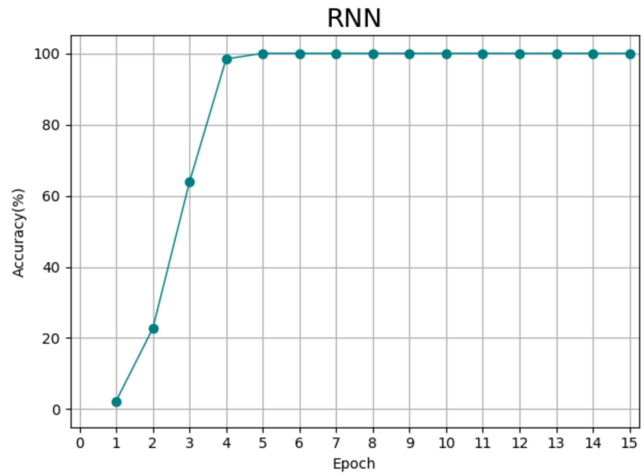
Lab4 Back-Propagation Through Time (BPTT)

0516054 劉雨恩

1. Plots show training accuracy




```
epoch  1 , error: 0.367875 , accuracy: 0.055
epoch  2 , error: 0.21175 , accuracy: 0.278
epoch  3 , error: 0.16475 , accuracy: 0.376
epoch  4 , error: 0.104125 , accuracy: 0.509
epoch  5 , error: 0.056125 , accuracy: 0.719
epoch  6 , error: 0.0 , accuracy: 1.0
epoch  7 , error: 0.0 , accuracy: 1.0
epoch  8 , error: 0.0 , accuracy: 1.0
epoch  9 , error: 0.0 , accuracy: 1.0
epoch  10 , error: 0.0 , accuracy: 1.0
epoch  11 , error: 0.0 , accuracy: 1.0
epoch  12 , error: 0.0 , accuracy: 1.0
epoch  13 , error: 0.0 , accuracy: 1.0
epoch  14 , error: 0.0 , accuracy: 1.0
epoch  15 , error: 0.0 , accuracy: 1.0
```

2. Describe how to generate data

a. Test data

在每一次iteration都產生一個2x8的input data，產生方法如下：

(1) random跑出兩個在[0,127]範圍裡的正整數。

```
a = randint(0, 127)
b = randint(0, 127)
```

(2) 將兩正整數轉換成binary string，再轉換成numpy array。而後為了使input的index小到大對應到二進位的低位到高位，將兩個array做reverse。(ex. 11000110 -> 01100011)

```
a = np.binary_repr(num=a_o, width=8)
a = np.fromstring(a, 'u1') - ord('0')
a = np.array(a)
a = np.flip(a)
b = np.binary_repr(num=b_o, width=8)
b = np.fromstring(b, 'u1') - ord('0')
b = np.array(b)
b = np.flip(b)
```

(3) 將兩個1x8的array合併成一個2x8的array，就成了forward的input data。

```
two = np.append(a[np.newaxis, :],b[np.newaxis, :],axis=0)
```

b. Ground truth data

(1) 將input data的兩正整數值相加。

(2) 轉換成binary string，再轉換成numpy array(同a(2))。

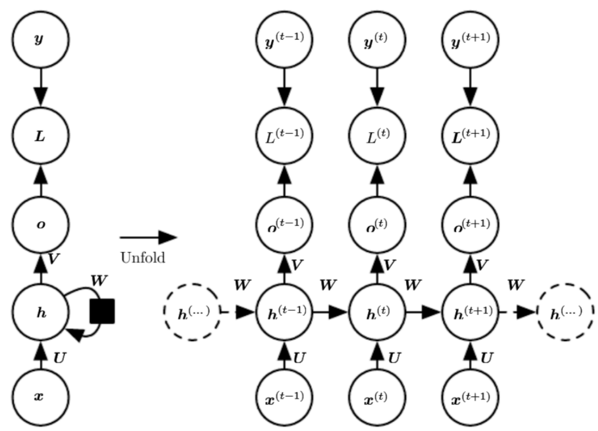
(3) 將1x8的array裡的每一個binary bit變成index，轉變成2x8的array。

(ex. [1, 0, 1, 1, 1, 0, 1, 0] -> [[0, 1, 0, 0, 0, 1, 0, 1],

[1, 0, 1, 1, 1, 0, 1, 0]])

```
for i in range(8):
    out[sum[i], i] = 1
```

3. Explain the mechanism of forward propagation



Forward pass的網路為RNN，在每個timestep都會輸出一個利用recurrent hidden unit連接的output (如上圖)，也就是一種Sequence-to-sequence mapping。而計算方式為下列式子：

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

4. Explain the mechanism of BPTT

BPTT即是將back-propagation應用於unrolled graphs。

而BPTT和一般gradient算法的不同之處在於狀態之間的通信；亦即梯度除了按照空間結構傳播以外，還得沿著時間通道傳播，因此採用迴圈循環的方法來計算各個梯度。

其gradient計算方式整理為下列式子：

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^T (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^T (\nabla_{\boldsymbol{o}^{(t)}} L)$$
$$= \boldsymbol{W}^T \boldsymbol{H}^{(t+1)} (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \boldsymbol{V}^T (\nabla_{\boldsymbol{o}^{(t)}} L)$$

where

$$\boldsymbol{H}^{(t+1)} = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{a}^{(t+1)}}\right)^T$$
$$= \begin{bmatrix} 1-(h_1^{(t+1)})^2 & 0 & \dots & 0 \\ 0 & 1-(h_2^{(t+1)})^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1-(h_n^{(t+1)})^2 \end{bmatrix}$$
$$\nabla_{\boldsymbol{o}^{(t)}} L = \hat{\boldsymbol{y}}^{(t)} - \boldsymbol{y}^{(t)}$$

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^T (\nabla_{\boldsymbol{o}^{(\tau)}} L) = \boldsymbol{V}^T (\hat{\boldsymbol{y}}^{(\tau)} - \boldsymbol{y}^{(\tau)})$$
$$\nabla_{\boldsymbol{W}} L = \sum_t \boldsymbol{H}^{(t)} (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{h}^{(t-1)T}$$
$$\nabla_{\boldsymbol{U}} L = \sum_t \boldsymbol{H}^{(t)} (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{x}^{(t)T}$$
$$\nabla_{\boldsymbol{V}} L = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)T}$$
$$\nabla_{\boldsymbol{b}} L = \sum_t \boldsymbol{H}^{(t)} (\nabla_{\boldsymbol{h}^{(t)}} L)$$
$$\nabla_{\boldsymbol{c}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L$$

5. Describe how the code work (the whole code)

a. Parameters

epoch= 15

iterations= 1000

alpha= 0.01

b. Training

照著a.Parameters設定能夠在epoch 10以內收斂並達到accuracy 100%的結果。而training code分為以下三個部分：

(1) Generate data

同2.Describe how to generate data所述。

(2) Forward pass

依照下列式子來進行forward pass。

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \mathsf{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

	Name	Dimension (per iterations)	Dimension (per time step)
x	input	2x8	2x1
a	units	16x1	16x1
h	hidden units	16x8	16x1
o	units	2x8	2x1
ŷ	target	2x8	2x1
W	input-to-hidden units	16x16	16x16
U	hidden-to-hidden units	16x2	16x2
V	hidden-to-output units	2x16	2x16

b	bias	16x1	16x1
c	bias	2x1	2x1
(t)	time step	None.	None.

每epoch會計算1000 iterations，並結算一次error和accuracy。(error: binary bit的正確率，accuracy: 數字的正確率)

(3) Backward pass

計算各項weight的gradient來進行backward pass，
如下列式子：

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^T (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^T (\nabla_{\boldsymbol{o}^{(t)}} L)$$
$$= \boldsymbol{W}^T \boldsymbol{H}^{(t+1)} (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \boldsymbol{V}^T (\nabla_{\boldsymbol{o}^{(t)}} L)$$

where

$$\boldsymbol{H}^{(t+1)} = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{a}^{(t+1)}}\right)^T$$
$$= \begin{bmatrix} 1-(h_1^{(t+1)})^2 & 0 & \dots & 0 \\ 0 & 1-(h_2^{(t+1)})^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1-(h_n^{(t+1)})^2 \end{bmatrix}$$

$$\nabla_{\boldsymbol{o}^{(t)}} L = \hat{\boldsymbol{y}}^{(t)} - \boldsymbol{y}^{(t)}$$

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^T (\nabla_{\boldsymbol{o}^{(\tau)}} L) = \boldsymbol{V}^T (\hat{\boldsymbol{y}}^{(\tau)} - \boldsymbol{y}^{(\tau)})$$

$$\nabla_{\boldsymbol{W}} L = \sum_t \boldsymbol{H}^{(t)} (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{h}^{(t-1)T}$$

$$\nabla_{\boldsymbol{U}} L = \sum_t \boldsymbol{H}^{(t)} (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{x}^{(t)T}$$

$$\nabla_{\boldsymbol{V}} L = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)T}$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \boldsymbol{H}^{(t)} (\nabla_{\boldsymbol{h}^{(t)}} L)$$

$$\nabla_{\boldsymbol{c}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L$$

τ為最後一個timestep。

c. Testing

將測資放入train好的RNN model裡，看其output出來的結果的正確性。而測資產生的方法有兩種。

(1) user輸入兩個input。

```
---Testing 1(input 1000 to break)---

input 1st integer:111
111
input 2nd integer:127
127
#####Result: 111 + 127 = 238#####
```

(2) 利用迴圈將所有可能數字都跑過一遍，127x127總共16129個數字，最後能得到一個right number/total number的accuracy。

```
---Testing 2---
#####Result: 0 + 0 = 0#####
#####Result: 0 + 1 = 1#####
#####Result: 0 + 2 = 2#####
#####Result: 0 + 3 = 3#####
#####Result: 0 + 4 = 4#####
#####Result: 0 + 5 = 5#####
#####Result: 0 + 6 = 6#####
#####Result: 0 + 7 = 7#####
#####Result: 0 + 8 = 8#####
#####Result: 0 + 9 = 9#####
#####Result: 0 + 10 = 10#####
#####Result: 0 + 11 = 11#####
#####Result: 0 + 12 = 12#####
#####Result: 0 + 13 = 13#####
#####Result: 0 + 14 = 14#####
#####Result: 0 + 15 = 15#####
#####Result: 0 + 16 = 16#####
#####Result: 0 + 17 = 17#####
#####Result: 0 + 18 = 18#####
#####Result: 0 + 19 = 19#####
#####Result: 0 + 20 = 20#####
#####Result: 0 + 21 = 21#####
#####Result: 0 + 22 = 22#####
#####Result: 0 + 23 = 23#####
```

略.

```
#####Result: 127 + 107 = 234#####
#####Result: 127 + 108 = 235#####
#####Result: 127 + 109 = 236#####
#####Result: 127 + 110 = 237#####
#####Result: 127 + 111 = 238#####
#####Result: 127 + 112 = 239#####
#####Result: 127 + 113 = 240#####
#####Result: 127 + 114 = 241#####
#####Result: 127 + 115 = 242#####
#####Result: 127 + 116 = 243#####
#####Result: 127 + 117 = 244#####
#####Result: 127 + 118 = 245#####
#####Result: 127 + 119 = 246#####
#####Result: 127 + 120 = 247#####
#####Result: 127 + 121 = 248#####
#####Result: 127 + 122 = 249#####
#####Result: 127 + 123 = 250#####
#####Result: 127 + 124 = 251#####
#####Result: 127 + 125 = 252#####
#####Result: 127 + 126 = 253#####
#####Result: 127 + 127 = 254#####
Accuracy: 1.0
```