Lab4 : Back Propagation Through Time

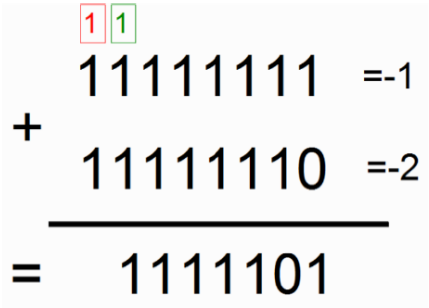
0516069 翁英傑

1. Introduction

In the training stage of a neural network, it relies on a technic called gradient decent. While calculating the gradient of the loss regarding to each weight, using back propagation will ease the computation, which the gradient is calculate from the output layer to the input layer. But in RNN, the weight is shared between every time steps. The calculation of back propagation became complicated. BPTT is the special back propagation of Recurrent Neural Network.

2. Experiment setups

A. Task: Binary addition



We take two at most 7-digits binary numbers and predict the result of addition. Each time step, a digit of both numbers is consider as input.

B. Data Generation (10%):

Random generate two numbers between 0 to 127 and compute the addition. Then turn all three numbers into binary with build-in function bin( ) in python.

B. Forward propagation (20%):

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},$$
$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}),$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},$$

Each time step calculates along the formula.

C. Back Propagation through time (20%):

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

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

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

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

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

Also compute the formula where

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

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

$$\boldsymbol{H}^{(t+1)} = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{a}^{(t+1)}} \right)^T$$

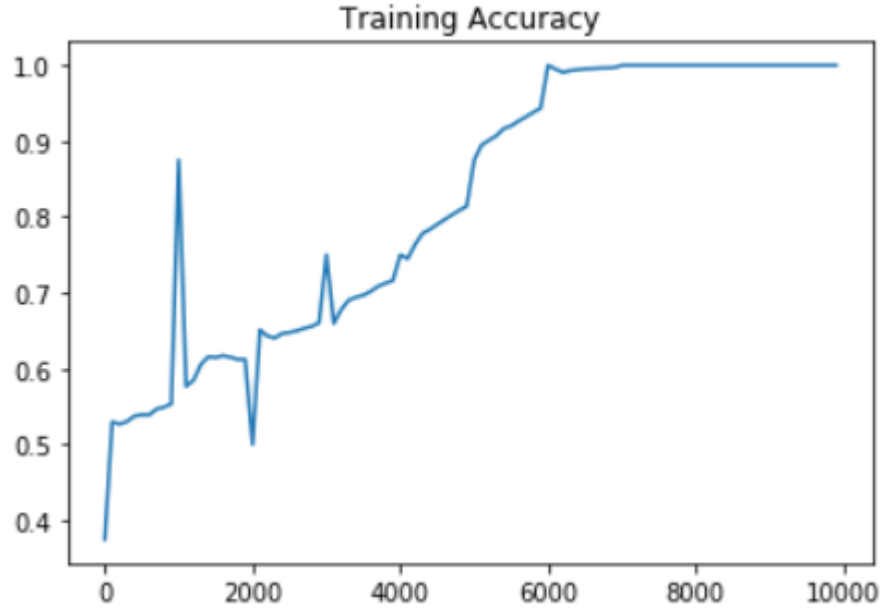
$$= \begin{bmatrix} 1 - (h_1^{(t+1)})^2 & 0 & \dots & 0 \\ 0 & 1 - (h_2^{(t+1)})^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 - (h_n^{(t+1)})^2 \end{bmatrix}$$

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

D. Describe the code (10%):

1. Define hyper parameters.
2. Define data generator.
3. Define useful math functions.
4. Define the RNN class.
5. Just fucking train it.

3. Results of testing (10%)




```
[0 0 1 0 1 0 1 0] + [0 0 1 0 1 1 1 1] = [0 1 1 0 1 1 1 0] ( [0 1 0 1 1 0 0 1] )
[0 0 1 1 0 1 0 0] + [0 1 1 1 0 0 0 0] = [0 1 1 0 1 1 1 0] ( [1 0 1 0 0 1 0 0] )
[0 1 0 1 1 0 1 0] + [0 0 1 1 0 1 0 1] = [0 1 1 0 1 1 1 0] ( [1 0 0 0 1 1 1 1] )
[0 1 1 1 1 0 1 1] + [0 1 0 0 1 0 0 0] = [0 1 1 0 1 1 1 0] ( [1 1 0 0 0 0 1 1] )
[0 1 1 0 0 1 0 1] + [0 0 0 1 0 0 0 0] = [0 1 1 0 1 1 1 0] ( [0 1 1 1 0 1 0 1] )
[0 1 0 1 0 1 0 0] + [0 0 0 0 1 1 0 0] = [0 1 1 0 1 1 1 0] ( [0 1 1 0 0 0 0 0] )
[0 0 0 0 0 0 0 0] + [0 0 1 1 1 0 0 0] = [0 1 1 0 1 1 1 0] ( [0 0 1 1 1 0 0 0] )
[0 0 1 0 0 1 0 1] + [0 1 0 1 0 0 1 1] = [0 1 1 0 1 1 1 0] ( [0 1 1 1 1 0 0 0] )
[0 0 0 0 1 0 1 1] + [0 1 0 0 0 0 0 1] = [0 1 1 0 1 1 1 0] ( [0 1 0 0 1 1 0 0] )
[0 0 1 0 0 1 0 1] + [0 0 0 1 1 1 0 0] = [0 1 1 0 1 1 1 0] ( [0 1 0 0 0 0 0 1] )
test accuracy : 100.0 %
```

Ten result for showing the result of the RNN and the accuracy