**Lab4: Back-Propagation Through Time (BPTT)**

# Lab Objective:

In this project, you are going to implement RNN from scratch. The optimization method is BPTT (Back-Propagation Through Time), which is mentioned on page 9, chapter 10. You may also need to implement every necessary function by using Numpy or pure Python.

Important Date:

1. Experiment Report Submission Deadline: 4/25 (Thu) 12:00

2. Demo date: 4/25 (Thu)

Turn in:

1. Experiment Report (.pdf)

2. Source code

Notice: zip all files in one file and name it like「DLP\_LAB4\_your

studentID\_name.zip」, ex: 「DLP\_LAB4\_0756172\_鍾嘉峻.zip」

# Lab Description:

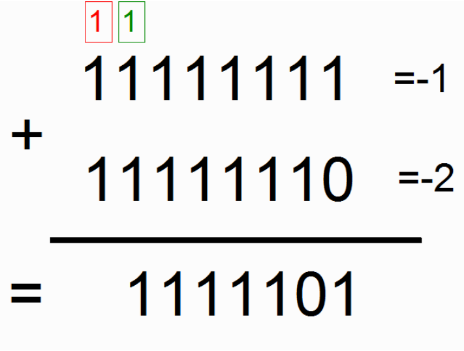
* Understand RNN model
* Basic Structure
* Forward Propagation
* Understand BPTT
* Gradient vanish & explosion problem
* Compare with Back-Propagation

# Requirements:

* Implement RNN network
* Construct the neural network
* Forward Propagation
* Back-Propagation Through Time
* Only Numpy or pure Python
* Apply your RNN model into Binary Addition task
* Do the Binary Addition task with your RNN model

# Task – Binary Addition:

* Introduction: The goal is trying to predict the result of two binary numbers addition
* Binary Number:
* Each Number is less than 256/2 ( total eight digits)
* You can use Numpy to create numbers
* Actions:
* A + B = C
* Given A and B (each of them will less than 256/2), predict the correct answer C



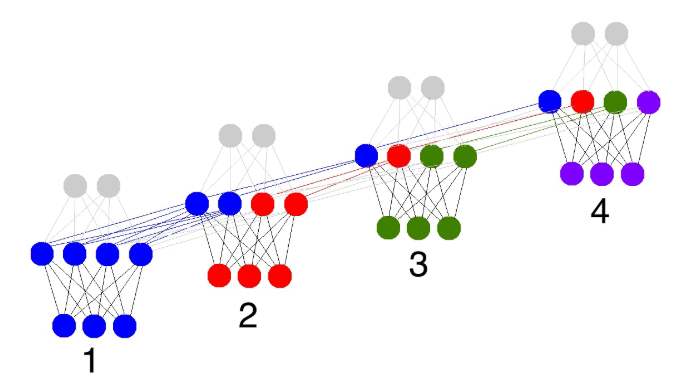
Picture 1. Example of binary addition

* Error:
* Simply count how many digits are different between the ground truth and your result
* Accuracy:
* Count how many correct answers in the last 1000 iterations

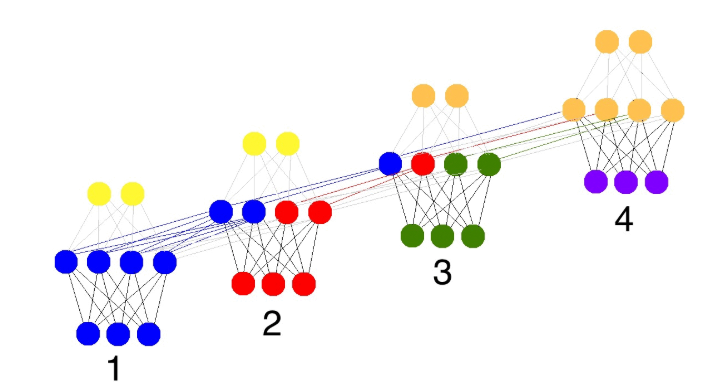
# Implementation Details:

## Network Architecture

* RNN model
  + - Binary Dimension: 8
    - Input Dimension: 2
    - Hidden Dimension: 16
    - Output Dimension: 1
* Training Parameters:
  + - Iteration: 20000
    - Alpha: 0.1
* Spending Time: in a few minutes
* Forward Propagation



* BPTT



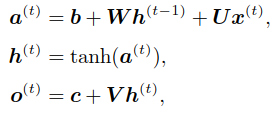
# Methodology:

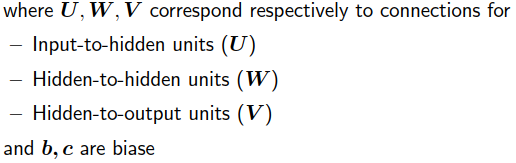
## Algorithm – RNN model



The above diagram shows an RNN being unrolled (or unfolded) into a full network. By unrolling, we mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word. The formulas that govern the computation happening in an RNN are as follows:

* + - x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
    - s_t is the hidden state at time step t. It’s the “memory” of the network. s_t is calculated based on the previous hidden state and the input at the current step: s_t=f(Ux_t + Ws_{t-1}). The function f usually is a nonlinearity such as [tanh](https://reference.wolfram.com/language/ref/Tanh.html) or [ReLU](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)).  s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
    - o_t is the output at step t. For example, if we wanted to predict the next word in a sentence, it would be a vector of probabilities across our vocabulary. o_t = \mathrm{softmax}(Vs_t).
* **Forward Propagation**

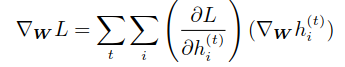


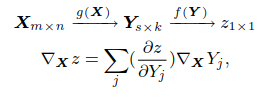


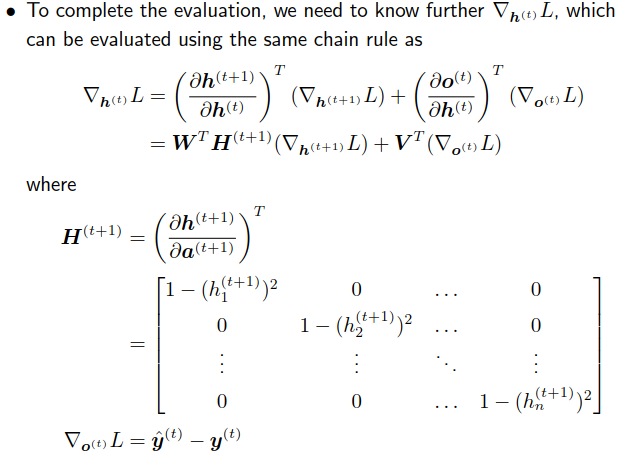
* **BPTT**

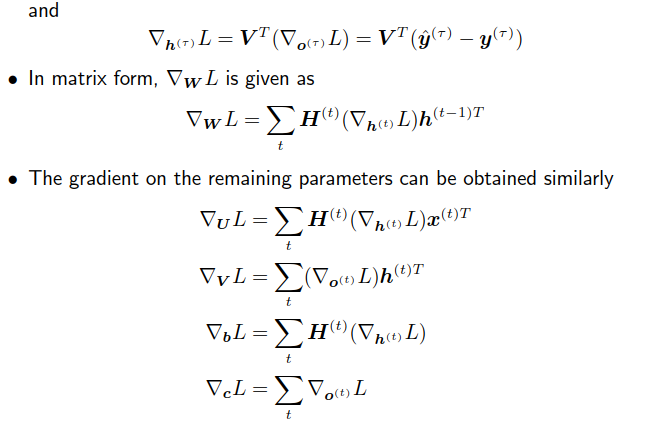
To compute  , we observe that:

* The immediate child nodes of W are all h(t)’s, and
* The chain rule for tensorsa can be applied to arrive at









Rule of Thumb:

* The error should decrease very fast if you write the correct code.
* Don’t set training iteration too large, the error should be close to 0 after 10000 iterations.

# Scoring Criteria:

# Report (70%)

* + A plot shows episode rewards of at least 10000 training episodes (10%)
  + Describe how to generate data? (10%)
  + Explain the mechanism of forward propagation (20%)
  + Explain the mechanism of BPTT (20%)
  + Describe how the code work (the whole code) (10%)
  + More you want to say

# Performance(30%)

# Accuracy \* 100

# References:

1. M.P. Cuéllar and M. Delgado and M.C. Pegalajar (2006). An Application of Non-linear Programming to Train Recurrent Neural Networks in Time Series Prediction Problems. Enterprise Information Systems VII. Springer Netherlands. pp. 95–102.
2. Pangolulu “Rnn From Scratch.” Retrieved from Github: <https://github.com/pangolulu/rnn-from-scratch>
3. Kjw0612. “Awesome Recurrent Neural Networks” Retrieved from Github: <https://github.com/kjw0612/awesome-rnn>