Lab4

鄭余玄

1 Introduction

In this lab, we are going to implement a native BPTT network. The main derivation of the backpropogation formula is described in the previous handwritten homework. I will introduce the entire training workflow in Section 2, and the training result is shown in Section 3.

2 Method

In each epoch, generate an input data and a ground-truth label, forward into the BPTT network, and update parameters. These steps are described in the following subsections:

2.1 Data Generation

Both input numbers has dimension of 8 which varies from $0 \sim 2^8$. In addition, since the inputs of a RNN architecture start from time 0, the input number is represented in a little-endian style, in other words, the least significant bit is in the front of memory layout (python list).

The implementation uses dictionary to convert a decimal number to a binary. First, generate a list of all possible numbers, and then unpack a number to a big-endian style binary using `np.unpackbits`. Finally, reverse the bits order using `np.flip`.

2.2 Forward

The input $\mathbf{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}\}$ consists of column digit vectors of two numbers. The initial hidden vector ($\mathbf{h}_0$) is initialized as zeros, while other variables are uniformly initialized. The loss is calculated by the mean squared error (MSE):

$$
\begin{aligned}
\mathbf{h}^{(t)} &= \tanh\big(\mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}\big) \\
\hat{\mathbf{y}}^{(t)} &= \sigma(\mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}) \\
\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) &= \sum_{t=1}^{\tau} \frac{1}{2} \left(y^{(t)} - \hat{y}^{(t)}\right)^2
\end{aligned}
$$

2.3 BPTT

The gradient of each varible is calculated as the following:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{U}} &= \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(t)}} \mathbf{x}^{(t)T} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{W}} &= \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(t)}} \mathbf{h}^{(t-1)T} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{b}} &= \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(t)}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{V}} &= \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial \mathbf{o}^{(t)}} \mathbf{h}^{(t)T} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{c}} &= \sum_{t=1}^{\tau} \frac{\partial \mathcal{L}}{\partial \mathbf{o}^{(t)}}
\end{aligned}
$$

where

$$
\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial \mathbf{o}^{(t)}} = \hat{y}^{(t)} - y^{(t)} && ,t \in \{1, \dots, \tau\} \\
&\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{a}^{(t)}} = \begin{pmatrix} 1 - \left(h_1^{(t)}\right)^2 & 0 & \dots & 0 \\ 0 & 1 - \left(h_1^{(t)}\right)^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 - \left(h_n^{(t)}\right)^2 \end{pmatrix} && ,t \in \{1, \dots, \tau\} \\
&\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(\tau)}} = \mathbf{V}^T \frac{\partial \mathcal{L}}{\partial \mathbf{o}^{(\tau)}} \\
&\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(t)}} = \mathbf{W}^T \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(t+1)}} + \mathbf{V}^T \frac{\partial \mathcal{L}}{\partial \mathbf{o}^{(t)}} && ,t \in \{1, \dots, \tau - 1\} \\
&\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(t)}} = \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{a}^{(t)}} \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(t)}} && ,t \in \{1, \dots, \tau\}
\end{aligned}
$$

Each trainable variable is updated with a fixed step size:

$$
\begin{aligned}
\mathbf{U} &\leftarrow \mathbf{U} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{U}} \\
\mathbf{W} &\leftarrow \mathbf{W} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \\
\mathbf{b} &\leftarrow \mathbf{b} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}} \\
\mathbf{V} &\leftarrow \mathbf{V} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{V}} \\
\mathbf{c} &\leftarrow \mathbf{c} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{c}}
\end{aligned}
$$

3 Result

The binary dimension is 2, the input dimension is 8, the hidden dimension is 16, the output dimension is 1, and the learning rate is 0.1. The initialization of hidden state is a zero vector, while other parameters are initialized uniformly within $(-1, 1)$.

3.1 Accuracy

The training accuracy is shown in Figure 1, which converges after 3000 epochs. The final accuracy achieves 100% all the time.

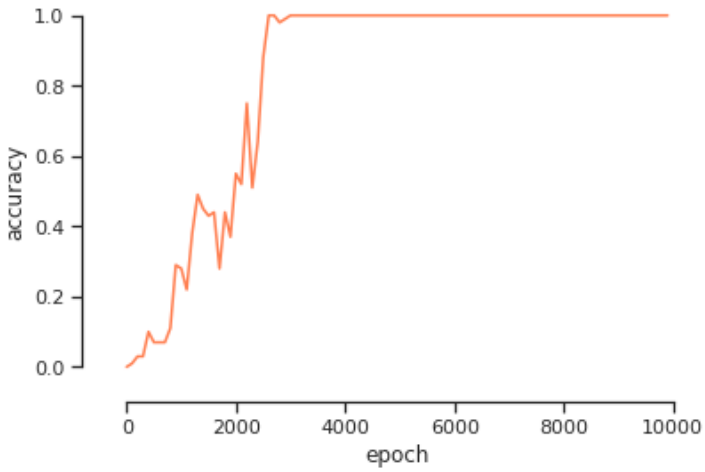

Figure 1: Training Accuracy

4 Discussion

One thing needs to be careful during the implementation is the order of input. The order of the input digit should start from the least significant bit to the most one, and therefore I represent the input numbers in the little-endian format.