Report for Assignment_3

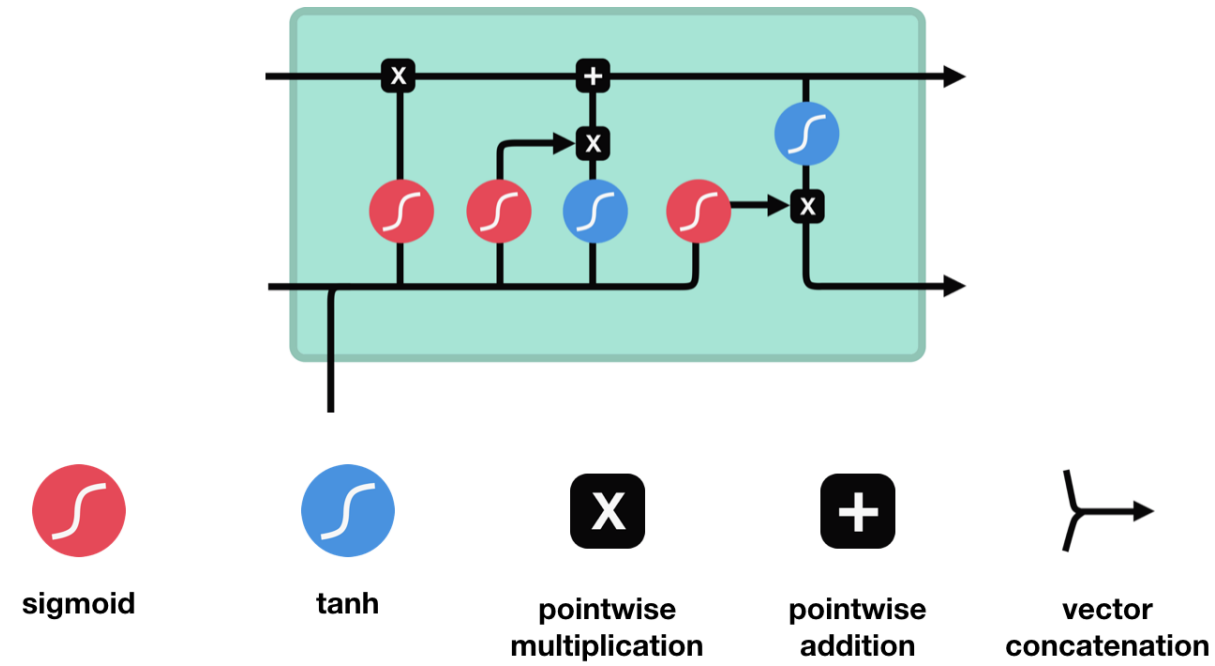
Introduction

LSTM is a variants of RNN, which introduce gating mechanism in the cell. Through extra forget gate, input gate and output gate, LSTM can control the data path. Here are the function of them:

- **Forget gate** decide what information we're going to throw away from the cell state.
- **Input gate** decides what new information we're going to store in the cell state.
- **Output gate** decides what we're going to output.

All of them use sigmoid function for activation, pushing values to be between 0 and 1. Also, there is a *tanh* gate the update the inner state of the cell.

Compare to standard RNN model, LSTM can avoid gradient vanishing problem, and is capable of learning long-term dependencies. They work tremendously well on a large variety of problems, and are now widely used.



Part 1: Differentiate LSTM

Differentiate one step of LSTM

First, I want to rewrite the format of LSTM cell for further deduction. If we throw away the concatenation matrix $\mathbf{z}$, and divide matrix $\mathbf{W}$ into two parts, $\mathbf{W}$ and $\mathbf{U}$, which are the parameters matrix for vector $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$, we have:

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f \cdot \mathbf{x}_t + \mathbf{U}_f \cdot \mathbf{h}_{t-1} + \mathbf{b}_t) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i \cdot \mathbf{x}_t + \mathbf{U}_i \cdot \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\overline{C}_t &= \tanh(\mathbf{W}_c \cdot \mathbf{x}_t + \mathbf{U}_c \cdot \mathbf{h}_{t-1} + \mathbf{b}_c) \\
C_t &= \mathbf{f}_t * C_{t-1} + \mathbf{i}_t * \overline{C}_t \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \cdot \mathbf{x}_t + \mathbf{U}_o \cdot \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t * \tanh(C_t)
\end{aligned}
$$

Besides, we need the derivation of sigmoid function $\sigma(x)$ and tanh function $\tanh(x)$:

$$
\begin{aligned}
y = \sigma(x) &= \frac{1}{1+e^{-x}} \\
y' = \sigma'(x) &= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} = y \cdot (1-y) \\
z = \tanh(x) &= \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = 2 \cdot \sigma(2x) - 1 \\
z' = \tanh'(x) &= 4 \cdot \sigma(2x)(1-\sigma(2x)) = 1 - z^2
\end{aligned}
$$

If we use the Chain Rule and deduce them in the following order, we can gain their gradients for the current cell:

$$
\frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} = diag[\tanh(C_t)] \tag{1}
$$

$$
\frac{\partial \mathbf{h}_t}{\partial C_t} = diag[\mathbf{o}_t * (1 - tanh(C_t))^2] \tag{2}
$$

$$
\frac{\partial \mathbf{h}_t}{\partial \overline{C}_t} = \frac{\partial \mathbf{h}_t}{\partial C_t} \cdot \frac{\partial C_t}{\partial \overline{C}_t} = diag[\mathbf{o}_t * (1 - tanh(C_t)^2) * \mathbf{i}_t] \tag{3}
$$

$$
\frac{\partial \mathbf{h}_t}{\partial C_{t-1}} = \frac{\partial \mathbf{h}_t}{\partial C_t} \cdot \frac{\partial C_t}{\partial C_{t-1}} = diag[\mathbf{o}_t * (1 - tanh(C_t)^2) * \mathbf{f}_t] \tag{4}
$$

$$
\frac{\partial \mathbf{h}_t}{\partial \mathbf{i}_t} = \frac{\partial \mathbf{h}_t}{\partial C_t} \cdot \frac{\partial C_t}{\partial \mathbf{i}_t} = diag[\mathbf{o}_t * (1 - tanh(C_t)^2) * \overline{C}_t] \tag{5}
$$

$$
\frac{\partial \mathbf{h}_t}{\partial \mathbf{f}_t} = \frac{\partial \mathbf{h}_t}{\partial C_t} \cdot \frac{\partial C_t}{\partial \mathbf{f}_t} = diag[\mathbf{o}_t * (1 - tanh(C_t)^2) * C_{t-1}] \tag{6}
$$

$$
\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_o} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} \cdot \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_o} = [\tanh(C_t) * \mathbf{o}_t * (1 - \mathbf{o}_t)] \cdot \mathbf{x}_t^\top \tag{7}
$$

$$
\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_f} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{f}_t} \cdot \frac{\partial \mathbf{f}_t}{\partial \mathbf{W}_f} = [\mathbf{o}_t * (1 - tanh(C_t)^2) * C_{t-1} * \mathbf{f}_t * (1 - \mathbf{f}_t)] \cdot \mathbf{x}_t^\top \tag{8}
$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_i} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{i}_t} \cdot \frac{\partial \mathbf{i}_t}{\partial \mathbf{W}_i} = [\mathbf{o}_t * (1 - tanh(C_t)^2) * C_t * \mathbf{i}_t * (1 - \mathbf{i}_t)] \cdot \mathbf{x}_t^\top \tag{9}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_c} = \frac{\partial \mathbf{h}_t}{\partial C_{t-1}} \cdot \frac{\partial C_{t-1}}{\partial \mathbf{W}_c} = [\mathbf{o}_t * (1 - tanh(C_t)^2) * \mathbf{f}_t * (1 - {C_{t-1}}^2)] \cdot \mathbf{x}_t^\top \tag{10}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{U}_o} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} \cdot \frac{\partial \mathbf{o}_t}{\partial \mathbf{U}_o} = [\tanh(C_t) * \mathbf{o}_t * (1 - \mathbf{o}_t)] \cdot {\mathbf{h_{t-1}}}^\top \tag{11}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{U}_f} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{f}_t} \cdot \frac{\partial \mathbf{f}_t}{\partial \mathbf{U}_f} = [\mathbf{o}_t * (1 - tanh(C_t)^2) * C_{t-1} * \mathbf{f}_t * (1 - \mathbf{f}_t)] \cdot {\mathbf{h_{t-1}}}^\top \tag{12}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{U}_i} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{i}_t} \cdot \frac{\partial \mathbf{i}_t}{\partial \mathbf{U}_i} = [\mathbf{o}_t * (1 - tanh(C_t)^2) * C_t * \mathbf{i}_t * (1 - \mathbf{i}_t)] \cdot {\mathbf{h_{t-1}}}^\top \tag{13}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{U}_c} = \frac{\partial \mathbf{h}_t}{\partial C_{t-1}} \cdot \frac{\partial C_{t-1}}{\partial \mathbf{U}_c} = [\mathbf{o}_t * (1 - tanh(C_t)^2) * \mathbf{f}_t * (1 - {C_{t-1}}^2)] \cdot {\mathbf{h_{t-1}}}^\top \tag{14}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{b}_o} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} \cdot \frac{\partial \mathbf{o}_t}{\partial \mathbf{b}_o} = \tanh(C_t) * \mathbf{o}_t * (1 - \mathbf{o}_t) \tag{15}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{b}_f} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{f}_t} \cdot \frac{\partial \mathbf{f}_t}{\partial \mathbf{b}_f} = \mathbf{o}_t * (1 - tanh(C_t)^2) * C_{t-1} * \mathbf{f}_t * (1 - \mathbf{f}_t) \tag{16}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{b}_i} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{i}_t} \cdot \frac{\partial \mathbf{i}_t}{\partial \mathbf{b}_i} = \mathbf{o}_t * (1 - tanh(C_t)^2) * C_t * \mathbf{i}_t * (1 - \mathbf{i}_t) \tag{17}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{b}_c} = \frac{\partial \mathbf{h}_t}{\partial C_{t-1}} \cdot \frac{\partial C_{t-1}}{\partial \mathbf{b}_c} = \mathbf{o}_t * (1 - tanh(C_t)^2) * \mathbf{f}_t * (1 - {C_{t-1}}^2) \tag{18}$$

$$\begin{aligned}
\frac{\partial \mathbf{h}_t}{\partial \mathbf{x}_t} =& \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} \cdot \frac{\partial \mathbf{o}_t}{\partial \mathbf{x}_t} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{f}_t} \cdot \frac{\partial \mathbf{f}_t}{\partial \mathbf{x}_t} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{i}_t} \cdot \frac{\partial \mathbf{i}_t}{\partial \mathbf{x}_t} + \frac{\partial \mathbf{h}_t}{\partial \overline{C}_t} \cdot \frac{\partial \overline{C}_t}{\partial \mathbf{x}_t} \\
=& [\tanh(C_t) * \mathbf{o}_t * (1 - \mathbf{o}_t)] \cdot \mathbf{W}_o + \\
& [\mathbf{o}_t * (1 - tanh(C_t)^2) * C_{t-1} * \mathbf{f}_t * (1 - \mathbf{f}_t)] \cdot \mathbf{W}_f + \\
& [\mathbf{o}_t * (1 - tanh(C_t)^2) * \overline{C}_t * \mathbf{i}_t * (1 - \mathbf{i}_t)] \cdot \mathbf{W}_i + \\
& [\mathbf{o}_t * (1 - tanh(C_t)^2) * \mathbf{i}_t * \mathbf{f}_t * (1 - {\overline{C}_t}^2)] \cdot \mathbf{W}_c
\end{aligned} \tag{19}$$

$$\begin{aligned}
\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} =& \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} \cdot \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_{t-1}} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{f}_t} \cdot \frac{\partial \mathbf{f}_t}{\partial \mathbf{h}_{t-1}} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{i}_t} \cdot \frac{\partial \mathbf{i}_t}{\partial \mathbf{h}_{t-1}} + \frac{\partial \mathbf{h}_t}{\partial \overline{C}_t} \cdot \frac{\partial \overline{C}_t}{\partial \mathbf{h}_{t-1}} \\
=& [\tanh(C_t) * \mathbf{o}_t * (1 - \mathbf{o}_t)] \cdot \mathbf{U}_o + \\
& [\mathbf{o}_t * (1 - tanh(C_t)^2) * C_{t-1} * \mathbf{f}_t * (1 - \mathbf{f}_t)] \cdot \mathbf{U}_f + \\
& [\mathbf{o}_t * (1 - tanh(C_t)^2) * \overline{C}_t * \mathbf{i}_t * (1 - \mathbf{i}_t)] \cdot \mathbf{U}_i + \\
& [\mathbf{o}_t * (1 - tanh(C_t)^2) * \mathbf{i}_t * \mathbf{f}_t * (1 - {\overline{C}_t}^2)] \cdot \mathbf{U}_c
\end{aligned} \tag{20}$$

However, we should notice that each cell in LSTM share the same weights, so the weights has multi-influence on $\mathbf{h}_t$, which is like a chain. When we differentiate weights, we should take this factor into consideration. Fortunately, formula (20) provide a convenient way for us to calculate the derivative through every LSTM cell.

We can use recursive expression to modify the formula above. So we can deduce the modification edition:

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_o} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} \cdot \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_o} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{o}_{t-1}} \cdot \frac{\partial \mathbf{o}_{t-1}}{\partial \mathbf{W}_o} + \cdots \\ &= \sum_{i=1}^{t} [\prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\tanh(C_i) * \mathbf{o}_i * (1 - \mathbf{o}_i)] \cdot \mathbf{x}_i^\top]\end{aligned} \tag{21}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_f} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{f}_t} \cdot \frac{\partial \mathbf{f}_t}{\partial \mathbf{W}_f} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{f}_{t-1}} \cdot \frac{\partial \mathbf{f}_{t-1}}{\partial \mathbf{W}_f} + \cdots \\ &= \sum_{i=1}^{t} [\prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\mathbf{o}_i * (1 - tanh(C_i)^2) * C_{i-1} * \mathbf{f}_i * (1 - \mathbf{f}_i)] \cdot \mathbf{x}_i^\top]\end{aligned} \tag{22}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_i} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{i}_t} \cdot \frac{\partial \mathbf{i}_t}{\partial \mathbf{W}_i} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{i}_{t-1}} \cdot \frac{\partial \mathbf{i}_{t-1}}{\partial \mathbf{W}_f} + \cdots \\ &= \sum_{i=1}^{t} [\prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\mathbf{o}_i * (1 - tanh(C_i)^2) * C_t * \mathbf{i}_i * (1 - \mathbf{i}_i)] \cdot \mathbf{x}_i^\top]\end{aligned} \tag{23}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_c} &= \frac{\partial \mathbf{h}_t}{\partial C_{t-1}} \cdot \frac{\partial C_{t-1}}{\partial \mathbf{W}_c} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial C_{t-2}} \cdot \frac{\partial C_{t-2}}{\partial \mathbf{W}_c} + \cdots \\ &= \sum_{i=1}^{t} [\prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\mathbf{o}_i * (1 - tanh(C_i)^2) * \mathbf{f}_i * (1 - {C_{i-1}}^2)] \cdot \mathbf{x}_i^\top]\end{aligned} \tag{24}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{U}_o} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} \cdot \frac{\partial \mathbf{o}_t}{\partial \mathbf{U}_o} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{o}_{t-1}} \cdot \frac{\partial \mathbf{o}_{t-1}}{\partial \mathbf{U}_o} + \cdots \\ &= \sum_{i=1}^{t} [\prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\tanh(C_i) * \mathbf{o}_i * (1 - \mathbf{o}_i)] \cdot {\mathbf{h_{i-1}}}^\top]\end{aligned} \tag{25}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{U}_f} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{f}_t} \cdot \frac{\partial \mathbf{f}_t}{\partial \mathbf{U}_f} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{f}_{t-1}} \cdot \frac{\partial \mathbf{f}_{t-1}}{\partial \mathbf{U}_f} + \cdots \\ &= \sum_{i=1}^{t} [\prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\mathbf{o}_i * (1 - tanh(C_i)^2) * C_{i-1} * \mathbf{f}_i * (1 - \mathbf{f}_i)] \cdot {\mathbf{h_{i-1}}}^\top]\end{aligned} \tag{26}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{U}_i} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{i}_t} \cdot \frac{\partial \mathbf{i}_t}{\partial \mathbf{U}_i} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{i}_{t-1}} \cdot \frac{\partial \mathbf{i}_{t-1}}{\partial \mathbf{U}_i} + \cdots \\ &= \sum_{i=1}^{t} [\prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\mathbf{o}_i * (1 - tanh(C_i)^2) * C_i * \mathbf{i}_i * (1 - \mathbf{i}_i)] \cdot {\mathbf{h_{i-1}}}^\top]\end{aligned} \tag{27}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{U}_c} &= \frac{\partial \mathbf{h}_t}{\partial C_{t-1}} \cdot \frac{\partial C_{t-1}}{\partial \mathbf{U}_c} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial C_{t-2}} \cdot \frac{\partial C_{t-2}}{\partial \mathbf{U}_c} + \cdots \\ &= \sum_{i=1}^{t} [\prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\mathbf{o}_i * (1 - tanh(C_i)^2) * \mathbf{f}_i * (1 - {C_{i-1}}^2)] \cdot \mathbf{h_{i-1}}^\top]\end{aligned} \tag{28}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{b}_o} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} \cdot \frac{\partial \mathbf{o}_t}{\partial \mathbf{b}_o} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{o}_{t-1}} \cdot \frac{\partial \mathbf{o}_{t-1}}{\partial \mathbf{b}_o} + \cdots \\ &= \sum_{i=1}^{t} \prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\tanh(C_i) * \mathbf{o}_i * (1 - \mathbf{o}_i)]\end{aligned} \tag{29}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{b}_f} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{f}_t} \cdot \frac{\partial \mathbf{f}_t}{\partial \mathbf{b}_f} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{f}_{t-1}} \cdot \frac{\partial \mathbf{f}_{t-1}}{\partial \mathbf{b}_o} + \cdots \\ &= \sum_{i=1}^{t} \prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\mathbf{o}_i * (1 - tanh(C_i)^2) * C_{i-1} * \mathbf{f}_i * (1 - \mathbf{f}_i)]\end{aligned} \tag{30}$$

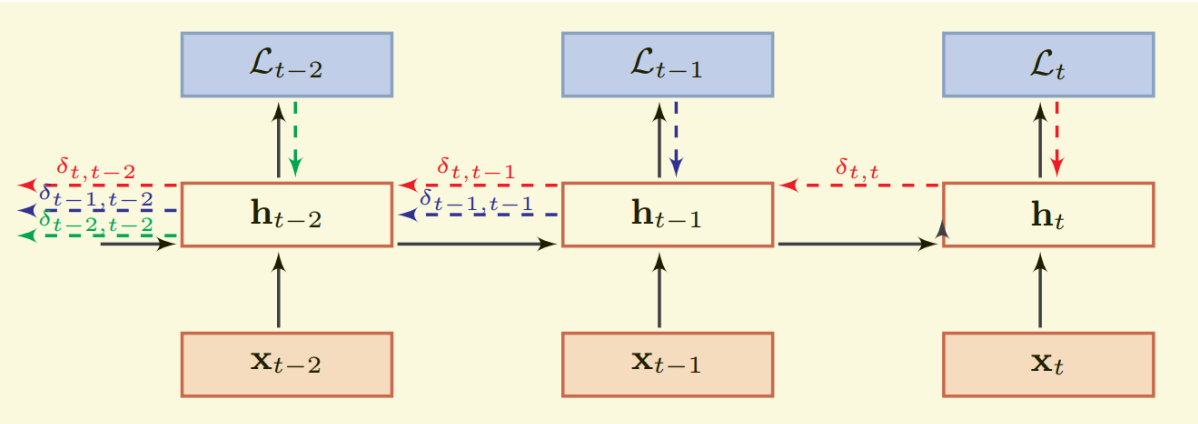
$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{b}_i} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{i}_t} \cdot \frac{\partial \mathbf{i}_t}{\partial \mathbf{b}_i} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{i}_{t-1}} \cdot \frac{\partial \mathbf{i}_{t-1}}{\partial \mathbf{b}_o} + \cdots \\ &= \sum_{i=1}^{t} \prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\mathbf{o}_i * (1 - tanh(C_i)^2) * C_i * \mathbf{i}_i * (1 - \mathbf{i}_i)]\end{aligned} \tag{31}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{b}_c} &= \frac{\partial \mathbf{h}_t}{\partial C_{t-1}} \cdot \frac{\partial C_{t-1}}{\partial \mathbf{b}_c} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial C_{t-2}} \cdot \frac{\partial C_{t-2}}{\partial \mathbf{b}_c} + \cdots \\ &= \sum_{i=1}^{t} \prod_{k=i}^{t-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} \cdot [\mathbf{o}_i * (1 - tanh(C_i)^2) * \mathbf{f}_i * (1 - {C_{i-1}}^2)]\end{aligned} \tag{32}$$

If we multiple all these item with a learning rate α , we can train our LSTM's parameters .

Differentiate through time (BPTT)

BPTT algorithm is another way to update the parameters through backward propagation. In BPTT algorithm, we consider CNN as an unrolling multi-layer feed-forward network. Especially, a "layer" corresponds to a "moment" in CNN. Then, we can calculate the gradients of every parameters. Because every layer in "unrolling" CNN share the same parameters, their actual gradients comes from the accumulation of every layers.



Here is the deduction of BPTT: First, we use cross-entropy as loss function

$$\mathcal{L} = \frac{1}{T}\sum_{i=1}^{T}\mathcal{L}_i = -\frac{1}{T}\sum_{i=1}^{T}\mathbf{t}_i^\top \log \mathbf{y}_i \tag{33}$$

where $\mathcal{L}_i$ denotes the loss, $\mathbf{t}_i$ denotes the true distribution and $\mathbf{y}_i$ denotes the predict distribution at time i. And T denotes the length of the sentence, which is equal to the number of the layer in LSTM.

Since $\mathcal{L}$ is a function of $\mathbf{h}_t$,we have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{1}{T}\sum_{i=t}^{T}\frac{\partial \mathcal{L}_i}{\partial \mathbf{h}_t} \tag{34}$$

Besides, since every $\mathcal{L}_i$ is a function of $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{b}$ (which is corresponding to the $\mathbf{x}$ in the formula 35), we should use full derivative when we calculate the gradients.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \sum_{t=1}^{T}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}\cdot\frac{\partial \mathbf{h}_t}{\partial \mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=t}^{T}\frac{\partial \mathcal{L}_i}{\partial \mathbf{h}_t}\cdot\frac{\partial \mathbf{h}_t}{\partial \mathbf{x}} \tag{35}$$

For prediction, we apply a softmax to get the probability distribution.

$$\mathbf{y}_t = softmax(\mathbf{W}_y\cdot\mathbf{h}_t) = \frac{exp(\mathbf{W}_y\cdot\mathbf{h}_t)}{\sum exp(\mathbf{W}_y\cdot\mathbf{h}_t)} \tag{36}$$

So, we can calculate the gradient of the loss. We write the loss function in detail first.

$$\begin{aligned}
\mathcal{L}_t &= -\mathbf{t}_t^\top \log(\mathbf{y}_t) = -\mathbf{t}_t^\top \log(\frac{exp(\mathbf{W}_y\cdot\mathbf{h}_t)}{\sum exp(\mathbf{W}_y\cdot\mathbf{h}_t)}) \\
&= -\begin{bmatrix} t_t^1 & t_t^2 & \cdots & t_t^n \end{bmatrix}\begin{bmatrix} \mathbf{W}_y^{(1)}\mathbf{h}_t - \log(\sum exp(\mathbf{W}_y\mathbf{h}_t)) \\ \mathbf{W}_y^{(2)}\mathbf{h}_t - \log(\sum exp(\mathbf{W}_y\mathbf{h}_t)) \\ \cdots \\ \mathbf{W}_y^{(n)}\mathbf{h}_t - \log(\sum exp(\mathbf{W}_y\mathbf{h}_t)) \end{bmatrix} \\
&= -\sum_{k=1}^{n}\mathbf{t}_t^k(W_y^{(k)}\mathbf{h}_t - \log(\sum exp(\mathbf{W}_y\mathbf{h}_t)) \\
&= -\mathbf{t}_t^\top\mathbf{W}_y\mathbf{h}_t + \log(\sum exp(\mathbf{W}_y\mathbf{h}_t))
\end{aligned} \tag{37}$$

We can quickly calculate the gradient with respect to $\mathcal{L}_t$ for the LSTM output vector $\mathbf{h}_t$.

$$\begin{aligned}
\frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t} &= -(\mathbf{t}_t^\top\mathbf{W}_y)^\top + \mathbf{W}_y^\top\frac{exp(\mathbf{W}_y\mathbf{h}_t)}{\sum exp(\mathbf{W}_y\mathbf{h}_t)} \\
&= \mathbf{W}_y^\top(\mathbf{y}_t - \mathbf{t}_t)
\end{aligned} \tag{38}$$

Similarly, we have

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial \mathbf{W}_y} &= -\mathbf{t}_t^\top \mathbf{h}_t + \frac{exp(\mathbf{W}_y \mathbf{h}_t)}{\sum exp(\mathbf{W}_y \mathbf{h}_t)}^\top \mathbf{h}_t \\ &= (\mathbf{y}_t - \mathbf{t}_t)^\top \mathbf{h}_t\end{aligned} \tag{39}$$

With the formula above, now we start to propagate backward. Take $\mathbf{h}_t$ an instance for further explanation.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} = \frac{1}{T}\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} = \mathbf{W}_y^\top (\mathbf{y}_T - \mathbf{t}_T) \tag{40}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_{T-1}} &= \frac{1}{T}\left[\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_{T-1}} + \frac{\partial \mathcal{L}_{T-1}}{\partial \mathbf{h}_{T-1}}\right] \\ &= \frac{1}{T}\left[\frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T}\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} + \frac{\partial \mathcal{L}_{T-1}}{\partial \mathbf{h}_{T-1}}\right]\end{aligned} \tag{41}$$

Formula 40 and 41 show the relations between 2 adjacent layer of the unrolling LTSM. The derivative of loss $\mathcal{L}$ with respect to the output vector $\mathbf{h}_{t-1}$ is depend on the latter. The first item of formula 41 is the element wise product of formula 40 and 20. Therefore, we should calculate the last layer of LSTM, and propagate backward to calculate gradient of every parameters. We can calculate every derivative of loss $\mathcal{L}$ with respect to every $\mathbf{h}_i$ in LSTM recursively.

Part 2: Autograd Training of LSTM

Preprocess

With the help of **FastNLP** library, we can build our own train set, data set and vocabulary quickly. Here is some work I've done for preprocessing:

- Add character $ at the end of the sentence , which is regarded as the end of a sentence.
- Call function *Dataset.apply()* to divide the sentence into words.
- Use class **Vocabulary** build a vocabulary automatically, which maps words into int, and set min_freq = 1. Besides, the word outside vocabulary will be map to character .

I preprocess two dataset: a toy_data which come from document *tangshi.txt* and a full_data which come from 全唐诗. I process them in two ways:

- For toy_data: Consider there're only a few poems in the data, I have to preserve all of them. Besides, poems in dataset vary in length, so it's hard to train them in batch. I decide to train them one by one when I use toy_data.
- For full_data: There're nearly 58,000 poems in 全唐诗, which costs too much time to train the model if we take all of them. For simplicity, I choose the poems that have 8 sentence and there're 5 words in each sentence. There're 11585 poems in full_data after preprocessing. Because the poems in dataset has the same length, it's convenient for us to train them in batch.

Initialization

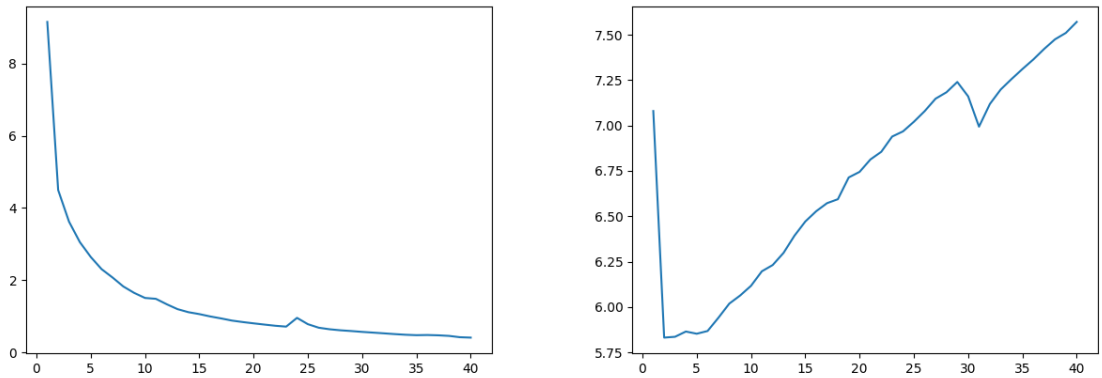
There're 3 layers in my model: an embedding layer, a single LSTM layer and an output layer. As for embedding layer and output, I choose to use the Embedding model and Linear model offered by PyTorch, which is initialized by random weight matrix. Besides, we initialize the parameters of LSTM with random number from 0 to 1. If we train without initialization, all the matrix will be equal to zero. We will get the same gradient for all parameters. Therefore, all parameters stay the same after training and our model will have a poor performance.

Training

The implement of LSTM is in **source.py**. I implement a LSTM layer with the help of Pytorch. I set parameters for full_data:

- data_dim = 256
- hidden_dim =128
- sentence_length =48
- batch_size = 20
- learning_rate = 1e-3

Shown in Figure 1, the loss function on training set decreases dramatically and converge to 1. In other words, our model can predict the next word correctly in training set with probability $e^{-1} = 36.8$. However, the best perplexity in development set is nearly $e^{5.8} = 330.3$. The big difference between training set and development set indicate our model may suffer over-fitting problem.



> Figure 1 The left is the loss in training set, and the right is the loss in development set.

Optimization

Adagrad

Adagrad is an optimization method which use different learning rate with respect to different parameters. It take the history gradients into consideration, which allow a big step when parameters change frequently and a small step when parameters change infrequently.

$$w^{t+1} \leftarrow w^t - \frac{\alpha}{\sqrt{\sum_{i=1}^{t}(\mathbf{g}^i)^2 + \epsilon}}\mathbf{g}^t$$

where ϵ is a term avoiding division by zero.

Its advantage is that parameters with small gradients can quickly descent. The weakness, however, is that the square of the gradient will accumulate, which will lead to learning rate decrease quickly and finally become very small. At that time, the algorithm is no longer able to improve its performance through training.

RMSProp

RMSProp is a variants of Adagrad, it takes some small change compared to Adagrad. It leads into a hyper-parameter ρ to control how much history information is acquired.

$$r \leftarrow \rho r + (1-\rho)\mathbf{g} * \mathbf{g}$$
$$w^{t+1} \leftarrow w^t - \frac{\alpha}{\sqrt{r+\epsilon}}\mathbf{g}^t$$

Given that the neural networks are all non-convex, RMSProp performs better under non-convex conditions, changing the gradient accumulation to a moving average of exponential decay to discard distant past history.

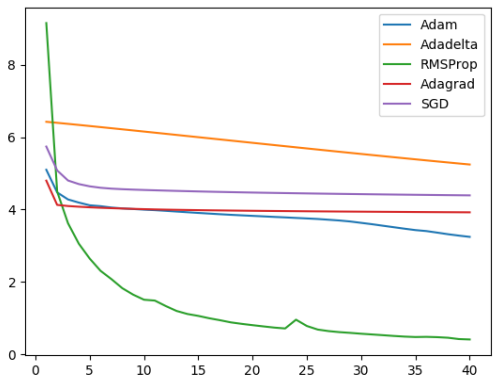
Adam

Adam algorithm is a combination of Momentum and RMSProp. It not only take momentum as a part of gradient, but also adapt its learning rate automatically.

$$M_t = \beta_1 M_{t-1} + (1-\beta_1)\mathbf{g}_t$$
$$G_t = \beta_2 G_{t-1} + (1-\beta_2)\mathbf{g}_t * \mathbf{g}_t$$
$$\hat{M}_t = \frac{M_t}{1-\beta_1^t}$$
$$\hat{G}_t = \frac{G_t}{1-\beta_2^t}$$
$$w^{t+1} \leftarrow w^t - \frac{\alpha}{\sqrt{\hat{G}_t+\epsilon}}\hat{M}_t$$

Comparison

To compare serveral algorithm above, we test five different gradient descent algorithms in PyTorch, including Adam, RMSProp, Adadelta, SGD and Adagrad. We run the first 160 poems in 全唐诗. Batch size is set to 20 and learning rate is set to 0.01. Result was showed in figure below.



> Figure 2 It shows different loss descent curve of different gradient descent algorithm.

Figure show that RMSProp perform best. Loss in RMSprop algorithm descent faster than others. Especially, an interesting phenomenon is that while Adadelta, Adam and Adagrad stop descent, SGD still has tendency getting lower.

Result

Poems generated with toy_data

> 月落辕门鼓角鸣，魏阙衡门路自分。

天子呼来成船精，恐汝后时难独立。
堂上书生空白头，日夜更望官军至。

日旗龙旆想飘扬，一索功高缚楚王。
直是超然五湖客，忍待明年莫仓卒。

红三百首，五陵谁唱与春风景。

山碎迹边。长安布衣谁比数，采撷细琐升中堂。

夜深应隔禁墙无时，采撷细琐升中堂。

湖卫身东意，与人一心成大功。

海亭秋日望，仍唱胡歌饮都市。

Poems generated with full_data

月皎昭阳殿，乘舆夜清风。郁涵碧，清疑待花时。
旧山头起佳垂，银陈酌长。莫见牀游申衔，天子凤皇一日。
夜夜星夕，一朝俱若三。连且欣去，如在与宦还。
相乐清跸清物，自莫九华藏。

日晚宜春陌，开轩柳初开。璧风送重凉黄昏，天山猿树雪。
寒气蔼看，参差间早梅。

红粉青楼曙，黄金谢鸣。偶室明深浅浪，风怀入云霄。
日唱灯，霜雪复多。君为谁第欢，何情感即人。情感绪，德道悬。

山气朝来爽，溪流芳光辉轶。佳草未尽，卷今乘尔远，三秋芳甸。
莫辨啼猿树，垂树与帷愁人。自意满既佳气，路梦里此，相望在成车。

夜久星沈没，更深月影斜。裙轻才动珮，鬟薄不胜花。
细风吹宝袂，日寒彩渐催。龙蛇林成极，挑德作清声。
忽见津将里，切丝为南。年年催为谁，不值楚极高唐春。
别离应易阳，澄与自然愧年。

湖碧行舟青山晴，云宫塞草树，云动马红。连沈虽乡情，非是为别离。
不知情纡阳，岂知故人仙舟。

海郡雄蛮落，津亭壮。激此欢先心，所思绝为欢情。
岑心与心。更无情，不爱华滋。相有乘无尽百丈曲，天下凤凰。

Reference & Sources

- Dataset of 全唐诗: https://github.com/chinese-poetry/chinese-poetry
- Understading LSTM Networks: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- 长短时记忆网络(LSTM): https://zybuluo.com/hanbingtao/note/581764