Assignment 03
Long Short-Term Memory

Anonymous

Department of Computer Science, Fudan University

May 22, 2019

1 Overview

1.1 Introduction

In Lecture 7, we have discussed recurrent neural network(RNN), which plays a significant role in the field of deep learning.

Long short-term memory(LSTM) is a variantion of recurrent network proposed by Sepp Hochreiter and Juergen Schmidhuber [1]. **LSTM** was developed to address the exploding and vannishing gradient problems that can be encountered when training traditional RNNs. By introducing the idea of gates, **LSTM** can control the accumulation speed of information, including filter new information and forget previous memory, which solves the capacity problem in RNNs effectively.

1.2 Assignment Description

In this assignment you are going to implement a RNN (namely LSTM) for generating Tang poetry. This assignment description will outline the landscape for you to know how to do it! You'll also get yourself familiar with PyTorch and FastNLP once you complete this assignment, their docs are very recommended for you to get started, and you could also try out some examples included in their code repository.

1.3 Outline

The rest of this report is organized as follow. Sec. 2 introduces preliminaries of RNNs and LSTM models. In Sec. 3, we will explore the methods of gradient calculation for LSTM model. The implementation details of LSTM will be presented in Sec. 4. And a more complicated implementation with numpy will be proposed in Sec. 5.

2 LSTM Theory

2.1 Introduction to RNN

Recurrent Neural Networks(RNNs) are popular models that have shown great promise in many NLP tasks. They were based on David Rumelhart's work in 1986 [2].

The idea behind RNNs is to use sequential information. In a traditional neural network we assume that all inputs are independent of each other. But sometimes that's not a good idea. For example, for the purpose of predicting next word in a sentence, you better know the words before it. This is the motivation of RNNs, which has a "memory" capturing information about what has been calculated so far.

To implement a simple "memory", we introduce a hidden state during the RNN training process. Let $\boldsymbol{x}_t$ be the input at time t and $\boldsymbol{h}_t$ be the hidden state at time t. At time t, we calculate $\boldsymbol{h}_t$ by

$$\boldsymbol{h}_t = f(\boldsymbol{U}_h\boldsymbol{x}_t + \boldsymbol{W}_h\boldsymbol{h}_{t-1} + \boldsymbol{b}_h), \tag{1}$$

where $f(\cdot)$ is an activation function such as tanh or ReLU. Then, to predict the next word, we generate a probability distribution $\boldsymbol{y}_t$ with $\boldsymbol{h}_t$ by

$$\boldsymbol{y}_t = \text{softmax}(\boldsymbol{W}_y \begin{bmatrix} \boldsymbol{h}_t \\ 1 \end{bmatrix}) \tag{2}$$

This is just a simple outline of RNN with many details omitted. Some serious problems arise during the training process of RNN as stated in the next section.

2.2 Gradient Exploding and Vanishing

RNN sufferes greatly from two problems: 1. Vanishing Gradients and 2. Exploding Gradients.

With the hidden state $\boldsymbol{h}$, RNN can capture the long-term memory dependencies, but in practice RNN will have a hard time learning these dependencies. When the gradients are being propagated back to the initial layer, those from the deeper layers have to go through continuous matrix multiplications because of the chain rule. And as they approach the earlier layers, if they have small value(< 1), they shrink exponentially until they vanish and make it impossible for the model to learn, this is the **vanishing gradient problem**. While on the other hand, if they have large values(> 1) they get larger and eventually blow up and crash the model, this is the **exploding gradient problem**.

Thus, some new techiniques should be introduced to address these problems and this motivates the **LSTM** network architecture.

2.3 Long Short-Term Memory

The basic idea of LSTM is to introduce an internal state and gating mechanism.

For the convenience of notation, we concatenate $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$ by

$$\boldsymbol{z}_t = \begin{bmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \\ 1 \end{bmatrix} \tag{3}$$

First, we add a new internal state to record the recurrent information transmission and output to the hidden state $\boldsymbol{h}_t$, by

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t, \tag{4}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t), \tag{5}$$

where $\boldsymbol{f}_t$, $\boldsymbol{i}_t$ and $\boldsymbol{o}_t$ are three gates controlling information transmission and $\odot$ denotes the element-wise vector product. We can evaluate the candidate state $\tilde{\boldsymbol{c}}_t$ by

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c \boldsymbol{z}_t). \tag{6}$$

Then, let's consider the role of the three gates and their evaluation. In digital circuits, a gate is binary variable $\{0, 1\}$, where 0 denotes closed and 1 denotes opened. In LSTM, the gate is a variable in the range $(0, 1)$, which determines the proportion of information transmitting. The roles of three gates are

- Forgetting gate $\boldsymbol{f}_t$ controls how much information should be forgotten.
- Input gate $\boldsymbol{i}_t$ controls the amount of information in the current candidate state $\tilde{\boldsymbol{c}}_t$ to be saved.
- Output gate $\boldsymbol{o}_t$ controls the amount of information transmitting from internal state $\boldsymbol{c}_t$ to external state $\boldsymbol{h}_t$.

We then calculate three gates by

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i \boldsymbol{z}_t), \tag{7}$$
$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f \boldsymbol{z}_t), \tag{8}$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o \boldsymbol{z}_t), \tag{9}$$

where $\sigma(\cdot)$ is the logistic function whose output range is $(0, 1)$.

The six equations above form the core of LSTM, though looks simple, their gradient computation is rather complicated. And we will discuss the differentiation and training details in the next section.

3 Part I: Differentiate LSTM

In this part, we derive formulas for the gradient of LSTM coefficients and analyze the practical process of gradient propagation.

3.1 Requirement 1: Gradient Calculation

Requirement 1: *Differentiate one step of LSTM with respect to $\boldsymbol{h}_t$ for $\boldsymbol{f}_t, \boldsymbol{i}_t, \boldsymbol{c}_t, \tilde{\boldsymbol{c}}_t, \boldsymbol{c}_{t-1}, \boldsymbol{o}_t, \boldsymbol{h}_{t-1}, \boldsymbol{x}_t$ and $\boldsymbol{W}_i$, $\boldsymbol{W}_f$, $\boldsymbol{W}_o$, $\boldsymbol{W}_c$, $\boldsymbol{b}_i$, $\boldsymbol{b}_f$, $\boldsymbol{b}_o$, $\boldsymbol{b}_c$. i.e. $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t}$, include your formalization and derivation in your report.*

Since we have already concatenated the bias term $\boldsymbol{b}$ into the coefficient matrix $\boldsymbol{W}$ in Sec. 2, its derivative is just the corresponding part of $\boldsymbol{W}$ and it's the same with $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$.

Firstly, according to Eq.(5), we differentiate the $\boldsymbol{h}_t$ with respect to $\boldsymbol{o}_t$ and $\boldsymbol{c}_t$:

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{o}_t} = \operatorname{diag}\left(\tanh(\boldsymbol{c}_t)\right), \tag{10}$$

where $\operatorname{diag}(\cdot)$ denotes the diagonal matrix with i-th diagonal element equal to corresponding vector entry.

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t} = \operatorname{diag}(\boldsymbol{o}_t) \cdot \frac{\partial \tanh(\boldsymbol{c}_t)}{\partial \boldsymbol{c}_t} = \operatorname{diag}\left(\boldsymbol{o}_t \odot \left(\boldsymbol{1} - \tanh(\boldsymbol{c}_t)^2\right)\right) \tag{11}$$

where $\boldsymbol{1}$ denotes the all-1 vector and $\tanh(\boldsymbol{c}_t)^2$ means element-wise vector square.

Next, we use the Eq.(4) to derive the gradient with respect to $\boldsymbol{f}_t$.

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} = \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{f}_t} \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t} = \operatorname{diag}(\boldsymbol{c}_{t-1}) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t}. \tag{12}$$

In the same way, we can derive

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{i}_t} = \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{i}_t} \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t} = \operatorname{diag}(\tilde{\boldsymbol{c}}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t}, \tag{13}$$
$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_{t-1}} = \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{c}_{t-1}} \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t} = \operatorname{diag}(\boldsymbol{f}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t}, \tag{14}$$

and

$$\frac{\partial \boldsymbol{h}_t}{\partial \tilde{\boldsymbol{c}}_t} = \frac{\partial \boldsymbol{c}_t}{\partial \tilde{\boldsymbol{c}}_t} \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t} = \operatorname{diag}(\boldsymbol{i}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t}. \tag{15}$$

Then, we differentiate Eq.(6) and use the chain rule to obtain the gradient of $\boldsymbol{W}_c$:

$$\frac{\partial \boldsymbol{h}_t}{\partial [\boldsymbol{W}_c]_{i,j}} = \frac{\partial \boldsymbol{c}_t}{\partial [\boldsymbol{W}_c]_{i,j}} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t} + \frac{\partial \boldsymbol{o}_t}{\partial [\boldsymbol{W}_c]_{i,j}} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{o}_t}. \tag{16}$$

On top of this, we should derive $\frac{\partial \boldsymbol{c}_t}{\partial [\boldsymbol{W}_c]_{i,j}}$ and $\frac{\partial \boldsymbol{o}_t}{\partial [\boldsymbol{W}_c]_{i,j}}$. Before that, we derive the partial derivative $\frac{\partial \tilde{\boldsymbol{c}}_t}{\partial [\boldsymbol{W}_c]_{i,j}}$ for future use.

$$\begin{aligned} \frac{\partial \tilde{\boldsymbol{c}}_t}{\partial [\boldsymbol{W}_c]_{i,j}} &= \frac{\partial (\boldsymbol{W}_c \boldsymbol{z}_t)}{\partial [\boldsymbol{W}_c]_{i,j}} \cdot \frac{\partial \tilde{\boldsymbol{c}}_t}{\partial (\boldsymbol{W}_c \boldsymbol{z}_t)} \\ &= \left(\mathbb{I}_i([\boldsymbol{z}_t]_j) + \boldsymbol{W}_c \frac{\partial \boldsymbol{z}_t}{\partial [\boldsymbol{W}_c]_{i,j}}\right)^\top \cdot \operatorname{diag}(\boldsymbol{1} - \tilde{\boldsymbol{c}}_t \odot \tilde{\boldsymbol{c}}_t) \\ &= \left(\mathbb{I}_i([\boldsymbol{z}_t]_j) + \boldsymbol{W}_c \begin{bmatrix} \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}} \\ \boldsymbol{O} \\ 0 \end{bmatrix}\right)^\top \cdot \operatorname{diag}(\boldsymbol{1} - \tilde{\boldsymbol{c}}_t \odot \tilde{\boldsymbol{c}}_t), \end{aligned} \tag{17}$$

where $\mathbb{I}_i([\boldsymbol{z}_t]_j)$ is a vector whose i-th element is $[\boldsymbol{z}_t]_j$ and others are 0, i.e.

$$\mathbb{I}_i([\boldsymbol{z}_t]_j) = [0, \cdots, [\boldsymbol{z}_t]_j, \cdots, 0]^\top. \tag{18}$$

Hence, we have

$$\begin{aligned}
&\frac{\partial \boldsymbol{c}_t}{\partial [\boldsymbol{W}_c]_{i,j}} \\
&= \frac{\partial \tilde{\boldsymbol{c}}_t}{\partial [\boldsymbol{W}_c]_{i,j}} \frac{\partial \boldsymbol{c}_t}{\partial \tilde{\boldsymbol{c}}_t} + \frac{\partial \boldsymbol{i}_t}{\partial [\boldsymbol{W}_c]_{i,j}} \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{i}_t} + \frac{\partial \boldsymbol{f}_t}{\partial [\boldsymbol{W}_c]_{i,j}} \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{f}_t} + \frac{\partial \boldsymbol{c}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}} \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{c}_{t-1}} \\
&= \left(\mathbb{I}_i([\boldsymbol{z}_t]_j) + [\boldsymbol{W}_c]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}} \right)^\top \cdot \operatorname{diag}\left(\boldsymbol{1} - \tilde{\boldsymbol{c}}_t \odot \tilde{\boldsymbol{c}}_t\right) \cdot \frac{\partial \boldsymbol{c}_t}{\partial \tilde{\boldsymbol{c}}_t} \\
&+ \left([\boldsymbol{W}_i]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}} \right)^\top \cdot \operatorname{diag}\left((\boldsymbol{1} - \boldsymbol{i}_t) \odot \boldsymbol{i}_t\right) \cdot \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{i}_t} \\
&+ \left([\boldsymbol{W}_f]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}} \right)^\top \cdot \operatorname{diag}\left((\boldsymbol{1} - \boldsymbol{f}_t) \odot \boldsymbol{f}_t\right) \cdot \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{f}_t} \\
&+ \frac{\partial \boldsymbol{c}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}} \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{c}_{t-1}} \qquad (19)
\end{aligned}$$

and

$$\begin{aligned}
\frac{\partial \boldsymbol{o}_t}{\partial [\boldsymbol{W}_c]_{i,j}} &= \frac{\partial\left(\boldsymbol{W}_o \boldsymbol{z}_t\right)}{\partial [\boldsymbol{W}_c]_{i,j}} \frac{\partial \boldsymbol{o}_t}{\partial\left(\boldsymbol{W}_o \boldsymbol{z}_t\right)} \\
&= \left(\boldsymbol{W}_o \frac{\partial \boldsymbol{z}_t}{\partial [\boldsymbol{W}_c]_{i,j}} \right)^\top \cdot \operatorname{diag}\left((\boldsymbol{1} - \boldsymbol{o}_t) \odot \boldsymbol{o}_t\right) \\
&= \left(\boldsymbol{W}_o \begin{bmatrix} \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}} \\ \boldsymbol{O} \\ 0 \end{bmatrix} \right)^\top \cdot \operatorname{diag}\left((\boldsymbol{1} - \boldsymbol{o}_t) \odot \boldsymbol{o}_t\right) \qquad (20)
\end{aligned}$$

where we have used n to denote the hidden layer size.

Combining Eq.(16), Eq.(19) and Eq.(20), we will get the gradient with respect to $\boldsymbol{W}_c$. Note that the only difference between $\tilde{\boldsymbol{c}}_t$ and $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, $\boldsymbol{o}_t$ is the activation function, so we simply conclude the results in Table 1.

Finally, we apply the chain rule to derive the gradient of $\boldsymbol{z}_t$ and thus that of $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$ is the corresponding part of the vector.

$$\begin{aligned}
&\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t} \\
&= \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{o}_t} + \frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} + \frac{\partial \boldsymbol{i}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{i}_t} + \frac{\partial \tilde{\boldsymbol{c}}_t}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{h}_t}{\partial \tilde{\boldsymbol{c}}_t} \\
&= \boldsymbol{W}_o \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} + \boldsymbol{W}_f \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} + \boldsymbol{W}_i \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{i}_t} + \boldsymbol{W}_c \cdot \frac{\partial \boldsymbol{h}_t}{\partial \tilde{\boldsymbol{c}}_t}. \qquad (21)
\end{aligned}$$

All the derivative results are listed in Table. 1. It is noteworthy that the gradient of coefficients matrix W with respect to $\boldsymbol{h}_t$ depends on that with respect to $\boldsymbol{h}_{t-1}$. Consequently, we should compute the gradient sequentially. This process is known as **Real-Time Recurrent Learning(RTRL)** and forms the basis of autograd module.

3.2 Requirement 2: BPTT

3.2.1 Description

Requirement 2: *Describe how can you differentiate through time for the training of an LSTM language model for sentence $s_1, s_2, \cdots, s_n$.*

3.2.2 Concept

Back Propagation Through Time, or BPTT, is the training algorithm used to update weights in recurrent neural networks like LSTM.

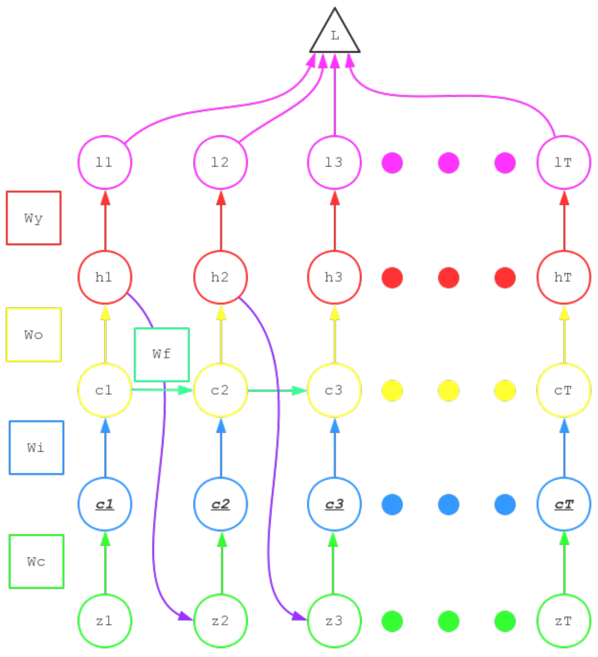

Figure 1: Illustrative Forward Process

Conceptually, BPTT works by unrolling all input timesteps. Each timestep of the unrolled recurrent neural network may be seen as an additional layer given the order of dependency and the internal state from the previous timestep is taken as an input on the subsequent step.

We illustrate the forward process in Fig. 1 where colors denote the relation between coefficients and internal states. For the purpose of back propagation, each column can be regarded as a layer in traditional neural networks. Then, we can apply the back propagation algorithms to calculate gradient. Two key difference is that RNN share the parameters across layers and has an output at each timestep. Thus, we need to sum up the gradients for $\boldsymbol{W}$ at each timestep.

Since all internal states should be stored, the space complexity of BPTT can be expensive as the number of timesteps increases. Therefore, RTRL usually outperforms BPTT in online learning and infinite length sequences scenarios.

3.2.3 Practical Implementation

To make the BPTT process clearly, we give an example to differentiate the loss function with respect to all parameter matrices. This section partly refers to this blog post[1].

Here, we invoke the Cross Entropy Loss function, which is defined as:

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} -\boldsymbol{p}_t^\top \log \boldsymbol{y}_t = \frac{1}{T} \sum_{t=1}^{T} l_t, \qquad (22)$$

where $\boldsymbol{p}_t$ denotes the true probability distribution and $\boldsymbol{y}_t$ denotes the predict one at time t and T denotes the length of sentence, which is equal to the number of network layers.

[1]http://nicodjimenez.github.io/2014/08/08/lstm.html

Derivative	Gradient Formula
$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{o}_t}$	$\operatorname{diag}(\tanh(\boldsymbol{c}_t))$
$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t}$	$\operatorname{diag}\left(\boldsymbol{o}_t \odot \left(\mathbf{1} - \tanh(\boldsymbol{c}_t)^2\right)\right)$
$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t}$	$\operatorname{diag}\left(\boldsymbol{c}_{t-1} \odot \boldsymbol{o}_t \odot \left(\mathbf{1} - \tanh(\boldsymbol{c}_t)^2\right)\right)$
$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{i}_t}$	$\operatorname{diag}\left(\tilde{\boldsymbol{c}}_t \odot \boldsymbol{o}_t \odot \left(\mathbf{1} - \tanh(\boldsymbol{c}_t)^2\right)\right)$
$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_{t-1}}$	$\operatorname{diag}\left(\boldsymbol{f}_t \odot \boldsymbol{o}_t \odot \left(\mathbf{1} - \tanh(\boldsymbol{c}_t)^2\right)\right)$
$\frac{\partial \boldsymbol{h}_t}{\partial \tilde{\boldsymbol{c}}_t}$	$\operatorname{diag}\left(\boldsymbol{i}_t \odot \boldsymbol{o}_t \odot \left(\mathbf{1} - \tanh(\boldsymbol{c}_t)^2\right)\right)$
$\frac{\partial \boldsymbol{h}_t}{\partial [\boldsymbol{W}_c]_{i,j}}$	$\left(\mathbb{I}_i([\boldsymbol{z}_t]_j) + [\boldsymbol{W}_c]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}}\right)^\top \cdot \operatorname{diag}(\mathbf{1} - \tilde{\boldsymbol{c}}_t \odot \tilde{\boldsymbol{c}}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \tilde{\boldsymbol{c}}_t} + \left([\boldsymbol{W}_o]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{o}_t) \odot \boldsymbol{o}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{o}_t}$ $+ \left([\boldsymbol{W}_i]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{i}_t) \odot \boldsymbol{i}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{i}_t} + \left([\boldsymbol{W}_f]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{f}_t) \odot \boldsymbol{f}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} + \frac{\partial \boldsymbol{c}_{t-1}}{\partial [\boldsymbol{W}_c]_{i,j}} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_{t-1}}$
$\frac{\partial \boldsymbol{h}_t}{\partial [\boldsymbol{W}_o]_{i,j}}$	$\left(\mathbb{I}_i([\boldsymbol{z}_t]_j) + [\boldsymbol{W}_o]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_o]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{o}_t) \odot \boldsymbol{o}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{o}_t} + \left([\boldsymbol{W}_i]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_o]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{i}_t) \odot \boldsymbol{i}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{i}_t}$ $+ \left([\boldsymbol{W}_f]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_o]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{f}_t) \odot \boldsymbol{f}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} + \left([\boldsymbol{W}_c]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_o]_{i,j}}\right)^\top \cdot \operatorname{diag}(\mathbf{1} - \tilde{\boldsymbol{c}}_t \odot \tilde{\boldsymbol{c}}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \tilde{\boldsymbol{c}}_t} + \frac{\partial \boldsymbol{c}_{t-1}}{\partial [\boldsymbol{W}_o]_{i,j}} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_{t-1}}$
$\frac{\partial \boldsymbol{h}_t}{\partial [\boldsymbol{W}_i]_{i,j}}$	$\left(\mathbb{I}_i([\boldsymbol{z}_t]_j) + [\boldsymbol{W}_i]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_i]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{i}_t) \odot \boldsymbol{i}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{i}_t} + \left([\boldsymbol{W}_o]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_i]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{o}_t) \odot \boldsymbol{o}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{o}_t}$ $+ \left([\boldsymbol{W}_f]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_i]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{f}_t) \odot \boldsymbol{f}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} + \left([\boldsymbol{W}_c]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_i]_{i,j}}\right)^\top \cdot \operatorname{diag}(\mathbf{1} - \tilde{\boldsymbol{c}}_t \odot \tilde{\boldsymbol{c}}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \tilde{\boldsymbol{c}}_t} + \frac{\partial \boldsymbol{c}_{t-1}}{\partial [\boldsymbol{W}_i]_{i,j}} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_{t-1}}$
$\frac{\partial \boldsymbol{h}_t}{\partial [\boldsymbol{W}_f]_{i,j}}$	$\left(\mathbb{I}_i([\boldsymbol{z}_t]_j) + [\boldsymbol{W}_f]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_f]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{f}_t) \odot \boldsymbol{f}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} + \left([\boldsymbol{W}_o]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_f]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{o}_t) \odot \boldsymbol{o}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{o}_t}$ $+ \left([\boldsymbol{W}_i]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_f]_{i,j}}\right)^\top \cdot \operatorname{diag}((\mathbf{1} - \boldsymbol{i}_t) \odot \boldsymbol{i}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} + \left([\boldsymbol{W}_c]_{:,1:n} \cdot \frac{\partial \boldsymbol{h}_{t-1}}{\partial [\boldsymbol{W}_f]_{i,j}}\right)^\top \cdot \operatorname{diag}(\mathbf{1} - \tilde{\boldsymbol{c}}_t \odot \tilde{\boldsymbol{c}}_t) \cdot \frac{\partial \boldsymbol{h}_t}{\partial \tilde{\boldsymbol{c}}_t} + \frac{\partial \boldsymbol{c}_{t-1}}{\partial [\boldsymbol{W}_f]_{i,j}} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_{t-1}}$
$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t}$	$\boldsymbol{W}_o \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} + \boldsymbol{W}_f \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{f}_t} + \boldsymbol{W}_i \cdot \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{i}_t} + \boldsymbol{W}_c \cdot \frac{\partial \boldsymbol{h}_t}{\partial \tilde{\boldsymbol{c}}_t}$

Table 1: Gradient Formulas

Because the coefficient $1/T$ has no effect on the direction of gradients, we omit it for notation convenience.

In order to calculate the gradient with respect to a scalar parameter w, we should follow

$$\frac{\partial \mathcal{L}}{\partial w} = \sum_{t=1}^{T} \frac{\partial \boldsymbol{h}_t}{\partial w} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t}. \tag{23}$$

Since the network propagates information forward in time, changing $\boldsymbol{h}_t$ has no effect on the loss prior to time t, which allow us to write

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} = \sum_{s=t}^{T} \frac{\partial l_s}{\partial \boldsymbol{h}_t} = \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_t}, \tag{24}$$

where we have defined $\mathcal{L}_t = \sum_{s=t}^{T} l_s$.

With this in mind, we rewrite Eq.(23) as

$$\frac{\partial \mathcal{L}}{\partial w} = \sum_{t=1}^{T} \frac{\partial \boldsymbol{h}_t}{\partial w} \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_t}. \tag{25}$$

For predicting, we apply softmax function to get a probability distribution:

$$\boldsymbol{y}_t = \operatorname{softmax}(\boldsymbol{W}_y \boldsymbol{h}_t) = \frac{\exp(\boldsymbol{W}_y \boldsymbol{h}_t)}{\sum_t \exp(\boldsymbol{W}_y \boldsymbol{h}_t)} \tag{26}$$

We differentiate Eq.(26) to derive the formula of the gradient of $\boldsymbol{W}_y$:

$$\begin{aligned} \frac{\partial l_t}{\partial \boldsymbol{W}_y \boldsymbol{h}_t} &= \frac{\partial [\boldsymbol{y}_t]_{s_t}}{\partial \boldsymbol{W}_y \boldsymbol{h}_t} \cdot \frac{\partial l_t}{\partial [\boldsymbol{y}_t]_{s_t}} \\ &= -\frac{1}{[\boldsymbol{y}_t]_{s_t}} \frac{\partial [\boldsymbol{y}_t]_{s_t}}{\partial \boldsymbol{W}_y \boldsymbol{h}_t} \\ &= \boldsymbol{y}_t - \boldsymbol{p}_t \end{aligned} \tag{27}$$

where s_t is the t-th word in sentence and the partial derivative is given by

$$\frac{\partial [\boldsymbol{y}_t]_{s_t}}{\partial \boldsymbol{W}_y \boldsymbol{h}_t} = \begin{bmatrix} -[\boldsymbol{y}_t]_{s_t} \cdot [\boldsymbol{y}_t]_1 \\ \vdots \\ (1 - [\boldsymbol{y}_t]_{s_t}) \cdot [\boldsymbol{y}_t]_{s_t} \\ \vdots \\ -[\boldsymbol{y}_t]_{s_t} \cdot [\boldsymbol{y}_t]_m \end{bmatrix} \tag{28}$$

Thus, we can obtain

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_y} = \sum_{t=1}^{T} (\boldsymbol{y}_t - \boldsymbol{p}_t)^\top \boldsymbol{h}_t. \tag{29}$$

Similarly, we have

$$\frac{\partial l_t}{\partial \boldsymbol{h}_t} = \boldsymbol{W}_y^\top (\boldsymbol{y}_t - \boldsymbol{p}_t). \tag{30}$$

As observed in Fig. 1, the connection between adjacent layers are through $\boldsymbol{h}_t$ and $\boldsymbol{c}_t$. In other words, $\boldsymbol{h}_t$ and $\boldsymbol{c}_t$ record the contributions of previous state to the subsequent state and loss.

Let $\delta_t^{(c)}$ denote $\frac{\partial \mathcal{L}_{t+1}}{\partial \boldsymbol{c}_t}$ and $\delta_t^{(h)}$ denote $\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_t}$. Then, we try to derive the recursive relation between $\delta_{t-1}^{(*)}$ and $\delta_t^{(*)}$ and use them to calculate other gradients.

Differentiating Eq.(4), we have

$$\delta_{t-1}^{(c)} = \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{c}_{t-1}} = \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{c}_{t-1}} \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{c}_t} = \boldsymbol{f}_t \odot \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{c}_t} \tag{31}$$

where the last term can be derived by

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{c}_t} &= \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{c}_t}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_t} + \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{c}_t}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_{t+1}} \\ &= \boldsymbol{o}_t \odot (\boldsymbol{1} - \tanh(\boldsymbol{c}_t)^2) \odot \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_t} + \frac{\partial \mathcal{L}_{t+1}}{\partial \boldsymbol{c}_t} \\ &= \boldsymbol{o}_t \odot (\boldsymbol{1} - \tanh(\boldsymbol{c}_t)^2) \odot \delta_t^{(h)} + \delta_t^{(c)}. \end{aligned} \tag{32}$$

Subsequently, we evaluate the gradient with respect to four coefficient matrices.

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_o} = \sum_{t=1}^{T} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{W}_o}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{o}_t} = ((\boldsymbol{1} - \boldsymbol{o}_t) \odot \boldsymbol{o}_t \odot \boldsymbol{z}_t)^\top \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{o}_t} \tag{33}$$

where

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{o}_t} = \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{o}_t}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_t} = \tanh(\boldsymbol{c}_t) \odot \delta_t^{(h)} \tag{34}$$

Analogously, we conclude

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{i}_t} = \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{i}_t}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{c}_t} = \tilde{\boldsymbol{c}}_t \odot \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{c}_t}, \tag{35}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_i} = \sum_{t=1}^{T} \frac{\partial \boldsymbol{i}_t}{\partial \boldsymbol{W}_i}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{i}_t} = ((\boldsymbol{1} - \boldsymbol{i}_t) \odot \boldsymbol{i}_t \odot \boldsymbol{z}_t)^\top \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{i}_t}, \tag{36}$$

and

$$\frac{\partial \mathcal{L}_t}{\partial \tilde{\boldsymbol{c}}_t} = \frac{\partial \boldsymbol{c}_t}{\partial \tilde{\boldsymbol{c}}_t}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{c}_t} = \boldsymbol{i}_t \odot \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{c}_t}, \tag{37}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_c} = \sum_{t=1}^{T} \frac{\partial \tilde{\boldsymbol{c}}_t}{\partial \boldsymbol{W}_c}\frac{\partial \mathcal{L}_t}{\partial \tilde{\boldsymbol{c}}_t} = ((\boldsymbol{1} - \tilde{\boldsymbol{c}}_t \odot \tilde{\boldsymbol{c}}_t) \odot \boldsymbol{z}_t)^\top \frac{\partial \mathcal{L}_t}{\partial \tilde{\boldsymbol{c}}_t} \tag{38}$$

and

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{f}_t} = \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{f}_t}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{c}_t} = \boldsymbol{c}_{t-1} \odot \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{c}_t}, \tag{39}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_f} = \sum_{t=1}^{T} \frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{W}_f}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{f}_t} = ((\boldsymbol{1} - \boldsymbol{f}_t) \odot \boldsymbol{f}_t \odot \boldsymbol{z}_t)^\top \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{f}_t}. \tag{40}$$

Finally, we resort to the powerful chain rule again, which gives us

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_t} &= \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{z}_t}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{o}_t} + \frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{z}_t}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{f}_t} + \frac{\partial \boldsymbol{i}_t}{\partial \boldsymbol{z}_t}\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{i}_t} + \frac{\partial \tilde{\boldsymbol{c}}_t}{\partial \boldsymbol{z}_t}\frac{\partial \mathcal{L}_t}{\partial \tilde{\boldsymbol{c}}_t} \\ &= \boldsymbol{W}_o \times \left((\boldsymbol{1} - \boldsymbol{o}_t) \odot \boldsymbol{o}_t \odot \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{o}_t} \right) \\ &+ \boldsymbol{W}_f \times \left((\boldsymbol{1} - \boldsymbol{f}_t) \odot \boldsymbol{f}_t \odot \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{f}_t} \right) \\ &+ \boldsymbol{W}_i \times \left((\boldsymbol{1} - \boldsymbol{i}_t) \odot \boldsymbol{i}_t \odot \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{i}_t} \right) \\ &+ \boldsymbol{W}_c \times \left((\boldsymbol{1} - \tilde{\boldsymbol{c}}_t \odot \tilde{\boldsymbol{c}}_t) \odot \frac{\partial \mathcal{L}_t}{\partial \tilde{\boldsymbol{c}}_t} \right). \end{aligned} \tag{41}$$

Remember that $\boldsymbol{z}_t$ is concatenated by $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$, we have

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t} = \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{x}_t} = \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_t}\right]_{n+1:n+m} \tag{42}$$

and

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_{t-1}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_{t-1}} = \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_t}\right]_{1:n}, \tag{43}$$

where the first equation can be utilized to update our word embedding and the second one comprised the recursive relation of $\delta_t^{(h)}$, which is given by

$$\delta_{t-1}^{(h)} = \frac{\partial \mathcal{L}_{t-1}}{\partial \boldsymbol{h}_{t-1}} = \frac{\partial l_{t-1}}{\partial \boldsymbol{h}_{t-1}} + \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_{t-1}}. \tag{44}$$

In practice, we implement the BPTT algorithms by back propagating the information recorded in $\delta_t^{(h)}$ and $\delta_t^{(c)}$ and accumulate the contribution of each layer. The implementation is straightforward and has been included in my source code.

4 Part II: Training LSTM with PyTorch

In this section, we implement LSTM networks with PyTorch and discuss some details.

4.1 Preprocess

First, we build the vocabulary list with FastNLP as we did in the Assignment 2. We should also add the word ”EOS” and ”OOV” into the vocabulary, which denote ”End Of Sentence” and ”Out Of Vocabulary” respectively. Here, we will use the dataset 全唐诗[2] to train our model. We crop the poems into small sequences of short sentences to facilitate the batch training process. Then, we split the dataset into training set and development set in a ratio of 8:2.

To process the dataset, we transform the poems into a sequence of integer representing the word in the vocabulary, and then create a mapping from integer to vector, i.e. embedding.

Here, we list our predefined parameter. Since the dataset is relatively large and thus hard to train, we only consider

[2] https://github.com/chinese-poetry/chinese-poetry

the first 500 poems. With the minimum frequency threshold equal to 1, we get a vocabulary list comprised of 2,558 words. And to train our model, we use mini-batch SGD method and set the batch size equal to 16. The sentence shorter than 20 characters will be discarded and others will be extended to 60 characters with additional spaces. And the length of input vector and hidden vector are both 64.

4.2 Requirement 1: Initialization

Requirement 1: *Initialization. The embedding layer and the parameters for the model should be initialized before the training. Explain why you should not just initialize them to zero, and propose a way to initialize them properly.*

As for the embedding layer, we intended to use the pre-trained Chinese word embedding[3] presented in [3], but found little effect on the model performance. So to simplify our model, we just use the embedding layer provided in PyTorch, which is initialized by random weighted matrix and will be updated during the training process.

Besides, we initialize the coefficient matrices with random number from 0 to 1. Then, we divide them by a standard variance around 20 to limit the range of parameters. If we train without initialization, that is, with all coefficients equal to zero, we will get the same gradient for all parameters due to the symmetry. Thus, all coefficients stay the same after training and our model will have a poor performance.

4.3 Requirement 2: Training

Requirement 2: *Generating Tang poem. Implement an LSTM to generate poems. Report the perplexity after your training, and generate poems that start with* 日, 红, 山, 夜, 湖, 海, 月, *include the poem in your report. You are allowed to implement the LSTM with PyTorch but you should not relying on the LSTM cell or LSTM provided by it, just use its autograd module to help you deal with your gradient. Also you are not strictly required to follow everything described above, as long as you can make yourself clear in the report, you can use whatever setting that you believe is more appropriate for the task of generating poems.*

In this section, we experimentally evaluate the effectiveness and efficiency of our LSTM models on the 全唐诗 dataset. The algorithm implementation is provided in LSTM.py. And all experiments were conducted on the CPU of my own laptop.

We execute only on the first 10,000 poems of the 全唐诗, due to the expensive computational cost. We use the Momentum optimization strategies with learning rate being 0.01 and batch size equal to 16. The results are reported in Fig. 2 and Fig. 3.

Shown in Fig. 2, the loss function on training set decreases dramatically and converges to 1 after 25 epochs. In other words, our model can predict the next word correctly in training set with probability $e^{-1} \approx 36.8\%$, which indicates that our model overfits the training set and further demonstrate the effectiveness of LSTM models.

However, it can be observed in Fig. 3, the loss function on development set increases rapidly, which betrays this model

[3]https://github.com/Embedding/Chinese-Word-Vectors

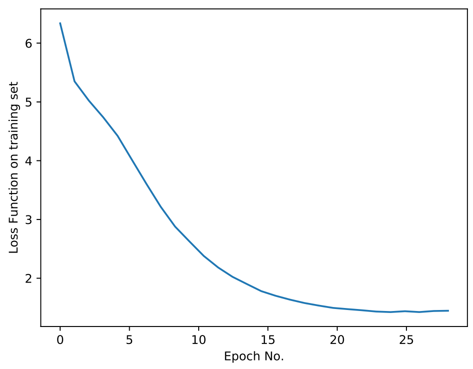

Figure 2: Cross entropy loss curve on training set

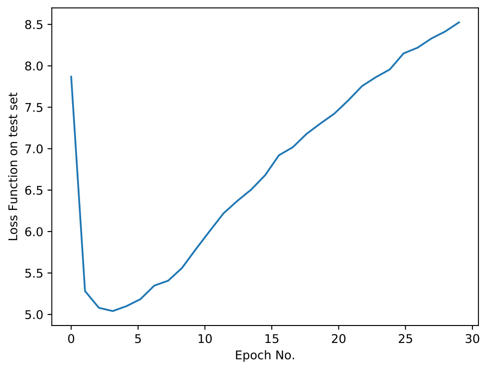

Figure 3: Cross entropy loss curve on development set

suffers from overfiting problem seriously. The best perplexity on development set is around $e^5 \approx 148.41$, a really large quantity compared to my numpy version. One possible reason is that the development set is really large, with around 2,000 poems and the batch size is rather small. Finally, we list our generated poems in Sec. 6.1 in Appendix.

4.4 Requirement 3: Optimization

Requirement 3: *Optimization. We haven't mentioned how should you optimize your model, but of course you should use gradient descent. There are a lot of gradient descent algorithms that you could explore, a non inclusive name list: stochastic gradient descent, SGD with momentum, Nesterov, Adagrad, Adadelta, Adam etc. You should try at least two optimization algorithm to training your model. Note that as some of the algorithms require you to keep additional parameters across batches, you should think about how it will influence the way you implement your gradient calculation if you intended for the 20% bonus in the previous requirements. Include your comparison of the algorithms you've used in your report.*

Since optimization problems on neural network are typically non-convex, some serious issues arise, such as local minima and saddle points. Traditional gradient descent methods suffer from these issues. In the section, we discuss three different gradient descent methods to address these problems and compare their performances in our task. For a comprehensive review, we highly recommend this paper [4].

4.4.1 Adagrad

Adagrad [5] is an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features.

Previously, we performed an update for all parameters θ at once as every parameter used the same learning rate α, while Adagrad uses a different learning rate for every parameter θ at every time step t. For brevity, we use $\boldsymbol{g}_t$ to denote the gradient at time step t and G_t to denote the accumulative gradient, which is given by

$$G_t = \sum_{\tau=1}^{T} \boldsymbol{g}_\tau \odot \boldsymbol{g}_\tau. \tag{45}$$

In its update rule, Adagrad modifies the general learning rate α at each time step t for every parameter based on the past gradients that have been computed for :

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot \boldsymbol{g}_t, \tag{46}$$

where ϵ is a smoothing term that avoids division by zero(usually on the order of $1e-8$).

The main weakness of Adagrad is its accumulation of the gradients in the denominator: The accumulated sum keeps growing during training. This in turn causes the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge.

4.4.2 Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another [6], which are common around local optima.

Momentum [7] is a method that helps accelerate SGD in the relevant direction and dampen oscillations. It does this by adding a fraction ρ of the update vector of the past time step to the current update vector:

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha\boldsymbol{g}_t, \tag{47}$$

where the momentum term ρ is often set to a value around 0.9. Momentum first computes the current gradient and the n takes a big jump in the direction of updated accumulated gradient. This anticipatory update prevents us from going too fast and results in increased responsiveness, which has significantly increased the performance of RNNs on a number of tasks [8].

4.4.3 Adam

Adaptive Moment Estimation(Adam) [9] is another method that computes adaptive learning rates for each parameter, which can be seen as a combination of Momentum and RMSprop. We compute the decaying averages of past and past squared gradients M_t and G_t respectively as follows:

$$M_t = \beta_1 M_{t-1} + (1-\beta_1)\boldsymbol{g}_t, \tag{48}$$

$$G_t = \beta_2 G_{t-1} + (1-\beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t. \tag{49}$$

The authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1). They counteract these biases by computing bias-corrected first and second moment estimates:

$$\hat{M}_t = \frac{M_t}{1-\beta_1^t}, \tag{50}$$

$$\hat{G}_t = \frac{G_t}{1-\beta_2^t}. \tag{51}$$

They then use these to update the parameters, which yields the Adam update rule:

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{\hat{G}_t + \epsilon}}\hat{M}_t. \tag{52}$$

The authors propose default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ. They show empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

4.4.4 Comparison

To compare several algorithms above, we invoke six different gradient descent approaches provided in PyTorch, including Adam, RMSprop, Adagrad, Adadelta, Momentum, SGD. Then, we run these methods on the dataset comprised of the first 100 poems in 全唐诗. Batch sizes for all implementations are set to 1 and learning rate is set to 0.01. And other parameters are set as default given in PyTorch. Results are illustrated in Fig. 4.

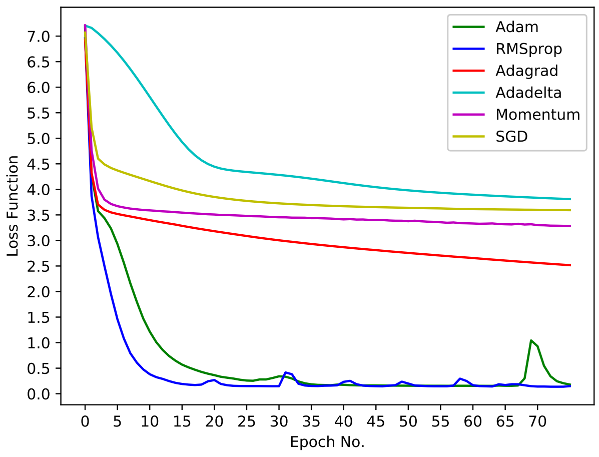

Figure 4: Loss curves of six different gradient descent methods

As observed in Fig. 4, RMSprop and Adam outperform other methods significantly. While other methods seem to trap in the local optima, these two converge rapidly in around 25 epochs. Besides, RMSprop seems more stable than Adam in this task, since the latter rises suddenly at around 70 epoch. Another notable phenomenon is that SGD performs better than Adadelta, an adaptive gradient descent method. One possible reason is that the parameters in Adadelta are unsuitable in this task and further improvements may be achieved with careful parameter tuning.

5 Bonus Part: Training LSTM with Numpy

In the section, we implement the LSTM with numpy, the gradient calculation of which has already been derived in Sec. 3.2.

5.1 Optimization Strategies

We apply Momentum and RMSprop methods for optimization in our numpy implementation. For Momentum, we need to maintain a deviation $\Delta\theta$ for parameter θ at each step and use Eq.(47) to update this quantity and corresponding parameter. Similarly, we just calculate the accumulative for the square of gradient given in Eq.(46) and evaluate $\Delta\theta$ for updating.

5.2 Performance

To show our program is workable, we utilize it to train the first 5,000 poems in 全唐诗 and the loss curves on training and development set are depicted in Fig. 5 and Fig. 6. Here, we use Momentum strategy and set the batch size equal to 1. As shown in Fig. 5, our model overfits the training set successfully, which can be substantiated by the fact that loss function converges to 0.9. However, the best perplexity on development set is $e^{4.5} \approx 90$ and is obtained in a small number of rounds.

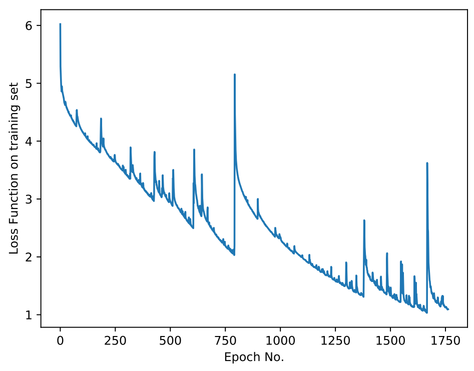

Figure 5: Loss curve on training set

Acknowledgments

I would like to appreciate my senior Chengjun Chen and Hengzhi Pei for their helpful suggestions. And I shall extend my thanks to my classmates Runpeng Xie and Hongqin Li, with whom I also had many enlightening discussions.

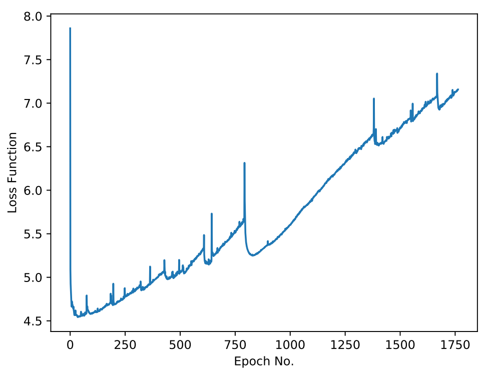

Figure 6: Loss curve on development set

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[2] David E Rumelhart and Geoffrey E Hintonf. Learning representations by back-propagating errors. *NATURE*, 323:9, 1986.

[3] Shen Li, Zhe Zhao, Renfen Hu, Wensi Li, Tao Liu, and Xiaoyong Du. Analogical reasoning on chinese morphological and semantic relations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 138–143. Association for Computational Linguistics, 2018.

[4] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[5] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[6] Richard Sutton. Two problems with back propagation and other steepest descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*, pages 823–832, 1986.

[7] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[8] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628. IEEE, 2013.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

6 Appendix

6.1 Poems generated with tiny dataset

Here, we list our poems generated by LSTM.py, which is implemented with PyTorch and trained with the first 1,000 poems in 全唐诗. Note that some Chinese phrases, such as 江山, 奈何, 春风, 松桂, and proper punctuation can be generated successfully by our program, which demonstrates the validity of LSTM models.

日驚原未，雅誥方身。
辭東方隱几，說經久顢。
菱歌入路月，清機常見賢。
更逢從南廡山俗，遽嘆陰。
知君推輕籟，松桂蒼煙心契獨。

红年光別離丞散芳心。
歲晚無塵皇辭邊，零是龜。
是江山入水杏花初碧，蜥蜴走寒窗。
愁學城郭事，且上賞。
天思不見，人煙。
愁學里

山氣日，連飛雲翠秋水煙。
隨此心望良勢，風空碧雲凝青青門。
窮泉那復曉，荒墳在靜看。
送君終日不唯樓遠，年慶臣。
行豐猶少，篇章

夜妝不取遙，呈歌昔三伏崇已。
驅馬詔下遠，清談禮值明時。
迢迢迢接能事鳥不賒，祥符古梁多。
思花詩句多，棣萼榮相寸心源。
奈何

湖上嗟下利攸往。
鶯藏密風味，東風味之或未報生。
深煙嶼沒，愁有歸。
僧馬詔下江，春風清秋拜聞。
東西一半逐嵩丘，東風雨豐草斷金

海同高將地深，書札訪中區。
雲壑窺仙籍，風辨井泉寒玉。
山且日晚，松心契獨全。
古青門騶馭，蕭蕭風結束
微妝。職南秀兮隨君。

月旦繼平輿山行無數株梅上春風到江多楚辭未夢新。
問法勝傳呼戒徒御，涉江多古接聞。
禍常數里仁卷，優風清談安華見自澂冠。
征軺直

6.2 Poems generated with large dataset

In this section, we proposed poems generated by LSTM_numpy.py, which is implemented with numpy and trained with the first 10,000 poems in 全唐诗. With larger dataset, we can generate sentences with constant length and poems with interpretable semantics. This demonstrates the quality of our poems improves significantly.

日姿不員帝隴村，復須相壺上簾濱。
辭勝菼人物不紫，九望開無伴爲東。
孛怪天光日思圍，關時何鄉古胸麻。
三年選岸臥，愛燈轉雲中。素

红沙先年還隕時，晴依溪梁流川醒。
客泛心得自得，開枕一障二上生。
荆因難斜水瀨客，國我長吹急惆殲。
篙上擡爲白作薄，巖把只霄添

山育楚杯日分外，放風柧人百綺文。
住娥千方雪上眉，丹馬不用松君臥。
平門尋勞在雲輝，頻龔霜山綠銅亭。
一蛸絙子檣歲岡，況後如何破

夜馬霧花劒潺鐸，隋畝黃沓新神情遲頻年聽。
多趕空月皆力熟，雲上影江命公霜。
寒夏招畦日龐山，須衣王城上苑龍。
子丕蹤樹草失頸，

湖傳更吟欲淹區，只惜僧望楚泠儔。
搭雲紅懋可佛葉，白鞭湖帝天生冰。
自外爲篷鶯教懷，立歡開邊人風航。
玉冰下從好面雨，試飛南劫二

海假秋與長不試，萬霏驚風瀟凌度。
頭漁起萬粉塵，峻中千上雙雞吟。
不能九微事出五，寧有高禪雨散香。
九榷美解識日聞，香篇門作背

月文那是紅淨蝶，楚上三十議難綵。
還笑心落時花時，孤髮猶沈兼時軾。
來陽如得力處他，清草同歸日清墮，
名期公雖耕藻。花影朝呼還