PRML Assignment 3

1 Part 1: Differentiate LSTM

1.1 Differentiate one step of LSTM with respect to h_t

First we should know some basic differential fomulas:

$$sigmoid^{'}(x) = sigmoid(x) * (1 - sigmoid(x))$$

$$tanh^{'}(x) = 1 - \tanh(x)^2$$

Next we can derive the bp formulas as below:

$$\frac{\partial h_t}{\partial o_t} = \tanh(C_t) \tag{1}$$

$$\frac{\partial h_t}{\partial C_t} = O_t * dtanh(C_t) = O_t * (1 - \tanh^2(C_t)) \tag{2}$$

$$\frac{\partial h_t}{\partial C_{t-1}} = \frac{\partial h_t}{\partial C_t}\frac{\partial C_t}{\partial C_{t-1}} = \frac{\partial h_t}{\partial C_t} * f_t = O_t * (1 - \tanh^2(C_t)) * f_t \tag{3}$$

$$\frac{\partial h_t}{\partial f_t} = \frac{\partial h_t}{\partial C_t}\frac{\partial C_t}{\partial f_t} = \frac{\partial h_t}{\partial C_t} * C_{t-1} = O_t * (1 - \tanh^2(C_t)) * C_{t-1} \tag{4}$$

$$\frac{\partial h_t}{\partial i_t} = \frac{\partial h_t}{\partial C_t}\frac{\partial C_t}{\partial i_t} = \frac{\partial h_t}{\partial C_t} * \overline{C_t} = O_t * (1 - \tanh^2(C_t)) * \overline{C_t} \tag{5}$$

$$\frac{\partial h_t}{\partial \overline{C_t}} = \frac{\partial h_t}{\partial C_t}\frac{\partial C_t}{\partial \overline{C_t}} = \frac{\partial h_t}{\partial C_t} * i_t = O_t * (1 - \tanh^2(C_t)) * i_t \tag{6}$$

$$\frac{\partial h_t}{\partial W_o} = \frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial W_o} = [o_t * (1 - o_t) * \tanh(C_t)] \cdot z^T \tag{7}$$

$$\frac{\partial h_t}{\partial W_f} = \frac{\partial h_t}{\partial f_t}\frac{\partial f_t}{\partial W_f} = [O_t * (1 - \tanh^2(C_t)) * C_{t-1} * f_t * (1 - f_t)] \cdot z^T \tag{8}$$

$$\frac{\partial h_t}{\partial W_i} = \frac{\partial h_t}{\partial i_t}\frac{\partial i_t}{\partial W_i} = [O_t * (1 - \tanh^2(C_t)) * \overline{C_t} * i_t * (1 - i_t)] \cdot z^T \tag{9}$$

$$\frac{\partial h_t}{\partial W_C} = \frac{\partial h_t}{\partial \overline{C_t}}\frac{\partial \overline{C_t}}{\partial W_f} = [O_t * (1 - \tanh^2(C_t)) * i_t * (1 - \overline{C_t}^2)] \cdot z^T \tag{10}$$

$$\frac{\partial h_t}{\partial b_o} = \frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial b_o} = \tanh(C_t) * o_t * (1 - o_t) \tag{11}$$

$$\frac{\partial h_t}{\partial b_f} = \frac{\partial h_t}{\partial f_t}\frac{\partial f_t}{\partial b_f} = O_t * (1 - \tanh^2(C_t)) * C_{t-1} * f_t * (1 - f_t) \tag{12}$$

$$\frac{\partial h_t}{\partial b_i} = \frac{\partial h_t}{\partial i_t}\frac{\partial i_t}{\partial b_i} = O_t * (1 - \tanh^2(C_t)) * \overline{C_t} * i_t * (1 - i_t) \tag{13}$$

$$\frac{\partial h_t}{\partial b_C} = \frac{\partial h_t}{\partial \overline{C_t}}\frac{\partial \overline{C_t}}{\partial b_C} = O_t * (1 - \tanh^2(C_t)) * i_t * (1 - \overline{C_t}^2) \tag{14}$$

$$z = concat(x_t, h_{t-1}) \tag{15}$$

$$\begin{aligned}\frac{\partial h_t}{\partial z} =& \frac{\partial h_t}{\partial o_t}\frac{\partial o_t}{\partial z} + \frac{\partial h_t}{\partial f_t}\frac{\partial f_t}{\partial z} + \frac{\partial h_t}{\partial i_t}\frac{\partial i_t}{\partial z} + \frac{\partial h_t}{\partial \overline{C_t}}\frac{\partial \overline{C_t}}{\partial z} \\ =& W_o^T \cdot (o_t * (1 - o_t) * \tanh(C_t)) + \\ & W_f^T \cdot (\frac{\partial h_t}{\partial f_t} * f_t * (1 - f_t)) + \\ & W_i^T \cdot \frac{\partial h_t}{\partial i_t} * (i_t * (1 - i_t)) + \\ & W_C^T \cdot \frac{\partial h_t}{\partial \overline{C_t}_t} * (1 - \overline{C_t}^2)\end{aligned} \tag{16}$$

1.2 Describe how to differentiate through time

Suppose the sequence length is fixed to T. vs represents the vocabulary size, l_k represents the loss at k step and L represents the total loss. L_k represents the sum of losses which are after k steps. Just as shown in the following formulas:

$$L_k = \Sigma_{t=k}^{T} l_t \tag{17}$$

$$l_t = -\Sigma_{j=1}^{vs} y_{t,j} \log \overline{y}_{t,j} \tag{18}$$

$$\overline{y} = softmax(W \cdot h_t + b) \tag{19}$$

We can realize that it is only l_k, which k is larger than t, will inference h_t. So we need to calculate the gradient reversely. Just like the fomulas in (20).

$$\begin{aligned}\frac{\partial L}{\partial h_t} &= \frac{\partial L_t}{\partial h_t} = \frac{\partial l_t}{\partial h_t} + \frac{\partial L_{t+1}}{\partial h_t} = \frac{\partial l_t}{\partial h_t} + \frac{\partial h_{t+1}}{\partial h_t}\frac{\partial L_{t+1}}{\partial h_{t+1}} \\ & if(t<T) \\ \frac{\partial L}{\partial h_t} &= \frac{\partial L_t}{\partial h_t} = \frac{\partial l_t}{\partial h_t} \\ & if(t=T)\end{aligned} \tag{20}$$

In (20), the item $\frac{\partial L_{t+1}}{\partial h_{t+1}}$ is passed from the next step, and this situation will continue until t equal to T. **So when we are calculating the gradients of the lstm cell(s), it is better for us to start from T to 1, which is contrary to the order of forwarding.**

2 Part 2: Autograd Training of LSTM

2.1 Experiments setting

- **Data**: I have collected more poetry data from github. Total poetry number is more than 5w. And they are split into training data and validation data by ratio 0.2. I use DataSet(tool of fastNLP) to pack my data and use pickle to store the objects.
- **Tools**: Pytorch, fastNLP, torchnet and pickle.
- **Python Version** 3.6
- **Higher Parameters**: embeddingSize=512, hiddenSize=512,lstm layer num=1, BatchSize=128, Optimizer=Adam learningRate=10^{-3},

2.2 Model Architecture

In this task, I simply use a single lstm layer to build my generator model. An embedding layer and a final predict layer are also added to my model. Here's a brief view of my model.

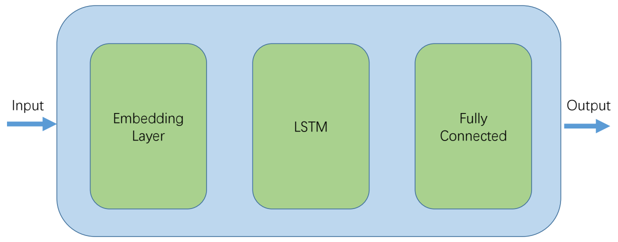


Figure 1: Poetry generator model architecture.

2.3 DataSet

I use fastNLP.DataSet to pack my poetry data and use fastNLP.Vocabulary to build the word vocabulary. During my experiments, I found if I used the provided small poetry dataset to be my training dataset, I always can not gain a well-trained model. So **I finally get more extra poetry data on github and my final poetry number is more than 50,000.** The certain details are illustrated as below.

Firstly, I pack the data into a single csv file. Then I use *read_csv*() function in fastNLP to read the whole dataset. For every poetrys, I firstly add "$< start >$" and "$< end >$" at the begining and the end of them. Then I choose to fix the length of each poetry to 120. If the poetry length is less then 120, it will be filled with "$< pad >$" at its head. Else if the poetry is longer than 120, I just extract the first 120 characters of it. Finally, I split the dataset to 80% and 20% as train dataset and develop dataset. All these operations are done by using fastNLP.DataSet.

2.4 Embedding Initialization

We should not initialize the embedding weigths to zeros. Once we do that, every time we call the backward function, gradients of the elements in the same embedding vector will be the same. And so their updated weigths will always be the same. That's not good.

How to initialize. If the initialization is not initialize well, we might also face some problems. We should think of the situation that if the input of the sigmoid function is very big, since the sigmoid function is too soft when the abs(input) is big enough, the training procedure will become very slow. In order to alleviate these problems, I have some ideas of initializing.

We can try evenly distributed and initialize all the weights into a small range. standard normal distribution can also be considered. What's more, Xavier initialization is also a good method to do such thing.

2.5 Early Stop by Callback & Perplexity Calculation

Early stop is already implemented in fastNLP, and I just need to import the class and use it with fastNLP.trainer. **The patient times of early stop I set is 10.**

Perplexity observation on validation set is mainly to reflect whether our model is overfitting on the training dataset. I imitate the AccuracyMetric class and implement my own PPMetric class.

By the way, when calculating the perplexity, it is better to avoid looping operations, instead, change them into matrics operations. The validation speed will be a lot faster. **there is a trick to calculate the perplexity values.** The original perplexity formula is shown as bellow. If we simply use this formula to calculate the perplexity, the value of $\prod_{i=t}^{N} y_i$ may be too small that the computer may not record. What's more, it isn't convenient to calculate $\prod$ and $\sqrt[N]{T}$. In order to avoid these problems, I utilize log and exp to transform the formula into a Σ form, which is more convinient to do calculation on GPUs.

$$origin : PP(S) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} y_i}}$$

$$new : PP(S) = \exp(\log PP(S)) = \exp(-\frac{1}{N}\Sigma_{i=1}^{N} \log y_i)$$

There are some important details while calculating the perplexity metric.

1. Not to use crossEntropy function to calculate the perplexity.. At the begining, I found the calculation form of perplexity is very similar to which of the crossEntropy function. So I want to ultilize the F.crossEntropy function to help me calculate perplexities and I firstly use formula (21) to do this. However, after a lot of experiments and deep thinkings, I found there exists problems.

If our inputs are batch by batch, the crossEntropy loss function will automatically average the losses by the first dimension of the input(which is the batch size). This operation is irreversible and will loss the variance information. **Using Jensen Inequality can prove what we gain is just a lower bound of the real perplexity.** Proving processes are as bellow.

$$\exp\left(crossEntropy\right) = \exp(-\frac{1}{B*N}\Sigma_{k=1}^{B}\Sigma_{i=1}^{N} \log y_{ki}) \tag{21}$$

$$\begin{aligned} Perplexity = \frac{1}{B}\Sigma_{k=1}^{B} PP(S_k) &= \frac{1}{B}\Sigma_{k=1}^{B} \exp(-\frac{1}{N}\Sigma_{i=1}^{N} \log y_{ki}) \\ &\geq \exp(-\frac{1}{B*N}\Sigma_{k=1}^{B}\Sigma_{i=1}^{N} \log y_{ki}) \\ &= exp(crossEntropy) \end{aligned} \tag{22}$$

2. We should ignore the influence of the padding items. Since the padding items are meaningless for our task, we shouldn't actually consider $< pad >$ as real parts of the sentences. Once we calculate the perplexity together with the padding items, the metric values, especially which of the short poetrys, will be seriously influenced. As the character after *pad* is always *pad*, and the *pad* accounts for a large proportion of all characters, the loss will becomes smaller and so as the perplexity metric.

So in my implementation, I have ignored $< pad >$ and consider the original poetrys only.

Also I have done some comparative experiments to verify my statements.

Cal by crossEntropy	Pad Ignore	Perplexity Value
True	False	7.9
False	False	22.1
False	**True**	**104.4**

Table 1: Different kinds of ways to calculate the perplexity. The first two ways are just figure out two lower bounds of the real perplexity. And only the third calculating way is standard.

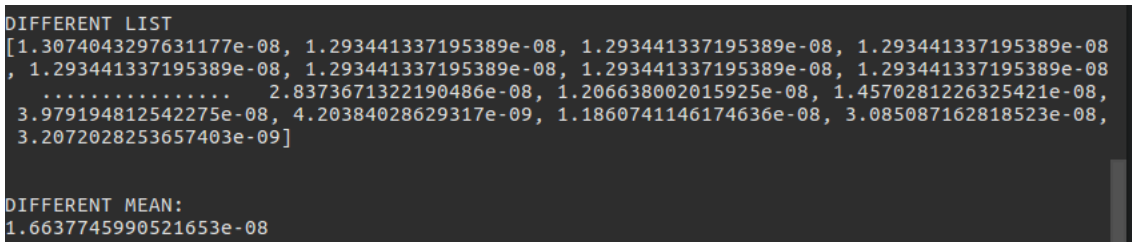
2.6 Loss Calculation Tip – exclude $< pad >$

It is obvious that the padding character has no meaning for our generation at all. **So when we calculate the prediction loss, the padding items should be ignored.** This is a necessary because the *pading* items may bring the problems of data imbalance and useless supervisory information for our task.

2.7 LSTM implemented by Numpy

I have implemented a simple lstm code by hand. In order to validate the correctness of my back propagation process, I use numerical method to calculate the gradient of each Variables.

The result shows that the order of magnitude of the gradient differences is 10-8, which verifies my numpy back propagation process is right. Results are shown in Figure 2.



```
DIFFERENT LIST
[1.3074043297631177e-08, 1.293441337195389e-08, 1.293441337195389e-08, 1.293441337195389e-08
, 1.293441337195389e-08, 1.293441337195389e-08, 1.293441337195389e-08, 1.293441337195389e-08
   ................   2.8373671322190486e-08, 1.206638002015925e-08, 1.4570281226325421e-08,
 3.979194812542275e-08, 4.20384028629317e-09, 1.1860741146174636e-08, 3.085087162818523e-08,
 3.2072028253657403e-09]


DIFFERENT MEAN:
1.6637745990521653e-08
```

Figure 2: This figure shows us the difference between numerical gradients and my own calculated gradients.

2.8 Optimize Algorithms

2.8.1 Different Optimize Algorithms

I have tried SGD, SGD with momentum and Adam algorighm to optimize my model parameters. Graphs are shown as Table 2 and Figure 3. As we can see, the Adam algorithm converges fastestly, while SGD algorithm converges slowestly. After adding the momentum item to SGD, its convergence rate becomes faster and more stable. Another algorithm, Adagrad, is also well performed. It converges very fast at the begining, which may because of the higher learning rate I set. But the final perplexity on validation set shows that Adam Optimizer can achieve a better solution.

2.8.2 How will different optimizers influence the way I implement my gradient calculation

Since different algorithms just add more information on the basis of the original gradient information, we just need to simply add more attributes to store those messages. For example, if we want to implement the Adam Algorithm, we just need to add **momentum**, **v**, **g**, **beta** etc, and modify the update formula.

Optimize Algorithm	Learning Rate	Batch Size	Loss	Perplexity	Best Step
SGD	1e-1	128	4.479	126.0331	15330
SGD (momentum=0.9)	5e-1	128	4.268	114.7476	3570
Adagrad	1e-2	128	4.096	106.4710	4650
Adam	1e-3	128	3.981	104.3631	3570

Table 2: Optimizer Comparison, ignore $< pad >$ when calculating loss and perplexity.

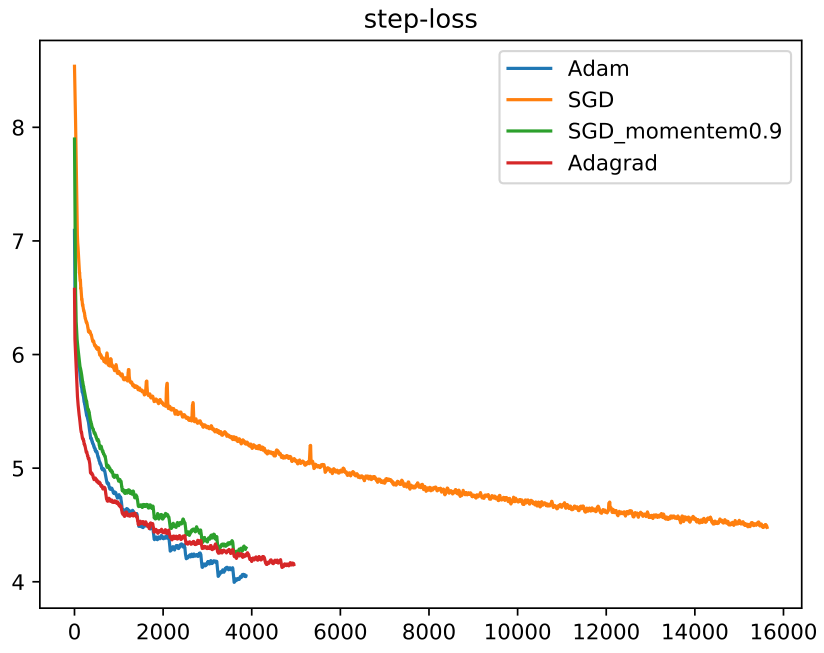


Figure 3: This is the loss cerve of appling four different optimizers. Early stop technique is used to judge whether it is proper to stop our training processes.

2.9 Generation Results

My strategy of generation:

- When generating the poetry, I use torch.multinomial function and choose words by probability.
- τ in softmax function is set smaller than 1.0, which is a good way to reduce the gap between different categories and make my generations more variant.
- Poetry sentence number is limited less than 8.
- If the highest score class corresponds to a punctuation type, I will output the punctuation directly rather than randomly select.

The follow table is the poetry I generated. Parameters I set is shown in Table 3

Optimize Algorithm	Learning Rate	Batch Size	Hidden Size	Input Size	Patience
Adam	1e-3	128	512	512	10

Table 3: Parameters Setting.

Start Word	τ Set	Poetry
日	0.6	日月和风动，山童戏鸟行。青山临水府，白日对秋岑。 树影孤舟里，松秋月满陂。青苔非我病，云暗雨过三。
	0.8	日夕阳寂历，乘兴方受厘。天子正萧洒，怀君杜鹃啼。 关河通汉月，吴蜀罗汉家。朱旗赤银火，星映碧流光。
红	0.6	红叶满空房，川静无言语。山中且无事，夜静月中寝。 有月看水精，清风来相见。客心自然灯，夜静无昼夜。
	0.8	红皓满庭花，琴台声渐知。黄昏云外寺，花在前山春。 结茅高岩壑，别地青春颜。树密和烟障，花移篱畔开。
山	0.6	山上青冥漠，春风汨渚田。凝章若云水，晶色映青天。 山果宜寒水，青苔掩翠屏。旧游无一事，故国又晴空。
	0.8	山酒重言展，飞云独坐清。高难销远别，岁晚何萧条。 徒为垢净宅，共作白头翁。若欲生穷伏，成心随我形。
夜	0.6	夜深秋月下，读书卷帘帷。翻思在襟思，露重松身殊。 山下多为雨，前朝树不开。开轩俯清净，引领褰帷幄。
	0.8	夜律堂堂静，松香自有人。黄昏时出旧，水色迥如流。 中有花正发，千株与绿珠。碧空垂象薄，留影若言空。
湖	0.6	湖阔水高天，云山几披雾。遥闻旧传书，月下怯不息。 吾知有殊人，客使君臣道。故人相逢迎，素手携手笑。
	0.8	湖外宫槐花满枝，肺肝尽写上飞红。儿童事伴多情态，长爱飘零半待新。 应笑杜陵与神女，全家流水向东流。君家少年厌头白，行脚群儿今日悲。
海	0.6	海阳为客，寂寞自然。神鬼不见，中有中肠。 阳和气爽气，锦缠合已。气和天地合，百神忽然。
	0.8	海内昔偏亲，南山五十年。河阳落军北，北斗横流水。 越王乃同年，开扉独行径。年年十二月，日暮归山里。
月	0.6	日月明湖岸，悠悠古道流。瞿塘都护远，高会远方归。 北阙朝骑远，天边月晓迎。东风天意气，从望帝乡心。
	0.8	月满婵娟，急如弦上。雁门苍，清夜月。 平生风雨，自然惊雁。散发枯，忆波红。

Table 4: **Generation Result. Don't ignore pad symble when calculating the loss and perplexity.**

Start Word	τ Set	Poetry
日	0.6	日出未有家，到长愁里看。一朝省郎官，俱在家家园。 愿君始知处，今日为我俦。吾道本非智，亦非求妄间。
	0.8	日落海阳天，生无独不到。所嗟生复苦，难与人伦已。 杀身戮归去，煎熬竞来息。知己尚独贫，长安是贫士。
红	0.6	红树有烟雨，秋声入耳目。高楼自掩扉，日高眠易水。 行人不得见，此意难重重。乱鸟出不足，孤巢摇荡桨。
	0.8	红萼季数里，一片情同游。去向边庭深，纵横遥复昏。 虽有儿女怨，又见李人行。景物随步履，云萝志难重。
山	0.6	山水生春光，夏天生绿泽。岁晏不相见，从来酒不易。 朱颜未知日，浅水随风疾。前事自成空，老夫在何处。
	0.8	山岳竹墙绿，四隅诗自开。鸡鸣凤吹起，白日下花台。 燕子随燕雀，饥乌遂相呼。儿童筋力弱，畏握双花须。
夜	0.6	夜光照如练，白玉不生色。我无主人意，岁晚相思量。 不见花枝死，春深无处多。路傍人不见，花枝不肯欺。
	0.8	夜伯家家本姓都，地中夜丧早秋凉。春衣尽日西京口，山里长安山月中。 排口月明中夜至，乌间风月夜初回。繁华卷水如风桂，更忆春光傍小春。
湖	0.6	湖外闻缲丝，吹铙飞上马。黄鹂鸣凤凰，秋月思杨柳。 春风如不起，日暮青云枝。所怀不可见，此意无由疑。
	0.8	湖水流初落，殷红带露盘。墙高花柳岸，春入柳花亭。 我入幽池在，何由见护人。酒杯随酒满，愁杀醉吟行。
海	0.6	海上将军占五兵，白日长驱使者谁。东风吹马空城里，遥指北山云去宿。 万里提携酒与马，今朝不与世人嗟。下有安危不顾者，日暮相思愁杀人。
	0.8	海燕衔蓬上，归来暗长台。高低映朱绂，摇落白云中。 门掩关河气，金岚碧海西。无由汉祖羽，回顾托中流。
月	0.6	月出风云母，轻盈玉女霜。玉罗裁锦帐，红袖掩金杯。 玉帐歌声咽，金盘贮锦茵。高低楼下望，疑是水银沙。
	0.8	月出崔嵬十日余，北风竿下拂尘埃。君今夜步萧萧索，左垅东风吹白草。 岸间作尉古贤人，曾城夔龙下西土。湖雨每随流水去，船头已在青莲花。

Table 5: **Generation Result. Ignore pad symble when calculating the loss and perplexity.**

3 Code Illustration

Filename	Function
packdata.py	解析csv（一行一诗），利用fastNLP的dataSet处理打包数据，存为pkl格式
utils.py	工具函数定义
config.py	参数设置
model.py	模型定义
train.py	模型训练与保存
generate.py	加载模型,生成古诗
lstmNumpy.py	numpy版本lstm的梯度验证

Table 6: Code illustration.