

# Assignment 3 Report

代码说明:

config.py 参数设置  
data.py 读取文件(每行一首诗)生成 vocabulary、train\_dataset、validate\_dataset  
model.py 用 pytorch 定义的 LSTM  
train.py 训练模型并保存  
generate.py 加载指定的模型并生成要求的诗歌  
LSTM\_numpy.py numpy 实现的 LSTM, 并与另一个 pytorch 实现的 LSTM 进行 loss 和梯度对比

## 一、Part 1

### 1. 针对 $h_t$ 对 LSTM 计算梯度

$$\frac{\partial h_t}{\partial C_t} = o_t * (1 - \tanh^2(C_t)) \quad (1)$$

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

$$\frac{\partial h_t}{\partial o_t} = \tanh(C_t) \quad (3)$$

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

$$\frac{\partial h_t}{\partial i_t} = \frac{\partial h_t}{\partial C_t} \frac{\partial C_t}{\partial i_t} = \frac{\partial h_t}{\partial C_t} * \bar{C}_t = o_t * (1 - \tanh^2(C_t)) * \bar{C}_t \quad (5)$$

$$\frac{\partial h_t}{\partial \bar{C}_t} = \frac{\partial h_t}{\partial C_t} \frac{\partial C_t}{\partial \bar{C}_t} = \frac{\partial h_t}{\partial C_t} * i_t = o_t * (1 - \tanh^2(C_t)) * i_t \quad (6)$$

$$\begin{aligned} \left[ \frac{\partial h_t}{\partial h_{t-1}}, \frac{\partial h_t}{\partial x_t} \right] &= \frac{\partial h_t}{\partial z} \\ &= \frac{\partial o_t}{\partial z} * \tanh(C_t) + \frac{\partial \tanh(C_t)}{\partial z} * o_t \\ &= W_o^T \cdot [o_t * (1 - o_t) * \tanh(C_t)] + \frac{\partial C_t}{\partial z} * o_t * (1 - \tanh^2(C_t)) \\ &= W_o^T \cdot [o_t * (1 - o_t) * \tanh(C_t)] + \left( \frac{\partial f_t}{\partial z} * C_{t-1} + \frac{\partial \bar{C}_t}{\partial z} * i_t + \frac{\partial i_t}{\partial z} * \bar{C}_t \right) \\ &\quad * o_t * (1 - \tanh^2(C_t)) \\ &= W_o^T \cdot [o_t * (1 - o_t) * \tanh(C_t)] \\ &\quad + W_f^T \cdot [f_t * (1 - f_t) * C_{t-1} * o_t * (1 - \tanh^2(C_t))] \end{aligned}$$

$$\begin{aligned}
& +W_c^T \cdot \left[ (1 - \bar{c}_t)^2 * i_t * o_t * (1 - \tanh^2(C_t)) \right] \\
& +W_i^T \cdot \left[ i_t * (1 - i_t) * \bar{c}_t * o_t * (1 - \tanh^2(C_t)) \right] \quad (7)
\end{aligned}$$

$$\frac{\partial h_t}{\partial W_f} = \frac{\partial h_t}{\partial f_t} \frac{\partial f_t}{\partial W_f} = \left[ \frac{\partial h_t}{\partial f_t} * f_t * (1 - f_t) \right] \cdot z^T = \left[ o_t * (1 - \tanh^2(C_t)) * c_{t-1} * f_t * (1 - f_t) \right] \cdot z^T \quad (8)$$

$$\frac{\partial h_t}{\partial W_i} = \frac{\partial h_t}{\partial i_t} \frac{\partial i_t}{\partial W_i} = \left[ \frac{\partial h_t}{\partial i_t} * i_t * (1 - i_t) \right] \cdot z^T = \left[ o_t * (1 - \tanh^2(C_t)) * \bar{c}_t * i_t * (1 - i_t) \right] \cdot z^T \quad (9)$$

$$\frac{\partial h_t}{\partial W_c} = \frac{\partial h_t}{\partial \bar{c}_t} \frac{\partial \bar{c}_t}{\partial W_c} = \left[ \frac{\partial h_t}{\partial \bar{c}_t} * (1 - \bar{c}_t)^2 \right] \cdot z^T = \left[ o_t * (1 - \tanh^2(C_t)) * i_t * (1 - \bar{c}_t)^2 \right] \cdot z^T \quad (10)$$

$$\frac{\partial h_t}{\partial W_o} = \frac{\partial h_t}{\partial o_t} \frac{\partial o_t}{\partial W_o} = \left[ \frac{\partial h_t}{\partial o_t} * o_t * (1 - o_t) \right] \cdot z^T = [\tanh(C_t) * o_t * (1 - o_t)] \cdot z^T \quad (11)$$

$$\frac{\partial h_t}{\partial b_f} = \frac{\partial h_t}{\partial f_t} \frac{\partial f_t}{\partial b_f} = \frac{\partial h_t}{\partial f_t} * f_t * (1 - f_t) = o_t * (1 - \tanh^2(C_t)) * c_{t-1} * f_t * (1 - f_t) \quad (12)$$

$$\frac{\partial h_t}{\partial b_i} = \frac{\partial h_t}{\partial i_t} \frac{\partial i_t}{\partial b_i} = \frac{\partial h_t}{\partial i_t} * i_t * (1 - i_t) = o_t * (1 - \tanh^2(C_t)) * \bar{c}_t * i_t * (1 - i_t) \quad (13)$$

$$\frac{\partial h_t}{\partial b_c} = \frac{\partial h_t}{\partial \bar{c}_t} \frac{\partial \bar{c}_t}{\partial b_c} = \frac{\partial h_t}{\partial \bar{c}_t} * (1 - \bar{c}_t)^2 = o_t * (1 - \tanh^2(C_t)) * i_t * (1 - \bar{c}_t)^2 \quad (14)$$

$$\frac{\partial h_t}{\partial b_o} = \frac{\partial h_t}{\partial o_t} \frac{\partial o_t}{\partial b_o} = \frac{\partial h_t}{\partial o_t} * o_t * (1 - o_t) = \tanh(C_t) * o_t * (1 - o_t) \quad (15)$$

## 2. 考虑时间序列，在训练时如何计算梯度

记总的序列长度为  $T$ ， $l_k$  代表  $k$  时刻的 loss， $L$  代表总的 loss，而  $L_k$  表示时刻  $k$  及以后的 loss。即有：

$$L_k = \sum_{t=k}^T l_t = - \sum_{t=k}^T \sum_j y_{t,j} \log \hat{y}_{t,j} \quad (16)$$

$$L = L_1 \quad (17)$$

其中

$$\hat{y}_t = \text{softmax}(W_v \cdot h_t + b) \quad (18)$$

对于  $h_t$  来说，它只对  $t$  时刻及以后的 loss 造成影响，因此有：

$$\begin{cases} \frac{\partial L}{\partial h_t} = \frac{\partial L_t}{\partial h_t} = \frac{\partial l_t}{\partial h_t} + \frac{\partial L_{t+1}}{\partial h_t} = \frac{\partial l_t}{\partial h_t} + \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial L_{t+1}}{\partial h_{t+1}} = W_v^T \cdot (\hat{y}_t - y_t) + \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial L}{\partial h_{t+1}} & \text{if } t < T \\ \frac{\partial L}{\partial h_t} = \frac{\partial L_t}{\partial h_t} = \frac{\partial l_t}{\partial h_t} = W_v^T \cdot (\hat{y}_T - y_T) & \text{if } t = T \end{cases} \quad (19)$$



## 2.2 用 pytorch 实现 LSTM

只需按照公式定义好前向传播的操作即可，相关细节见 model.py。

## 2.3 用 numpy 实现 LSTM

在前向传播时记录好相关参数，然后在反向传播时根据时间 t 逆序求梯度即可，具体的实现见 LSTM\_numpy.py，主要是学习和参考了[http://blog.varunajayasiri.com/numpy\\_lstm.html](http://blog.varunajayasiri.com/numpy_lstm.html) 这一博客。

为了验证用 numpy 实现的 lstm 的正确性，我在 LSTM\_numpy.py 中定义了另一个用 pytorch 实现的 lstm。与 model.py 中不同的地方在于我使用 Parameter 而不是线性层，这样使得我能够初始化参数并且获得参数的梯度。因为只有在初始参数相同的情况下才能进行对比。

对两个模型的 loss 和部分梯度作差，部分结果如下（运行 LSTM\_numpy.py）：

```
tensor(-0.0025)
tensor([-5.2951e-07, -5.2951e-07, -5.2951e-07, -5.2951e-07, -5.2951e-07,
        -5.2951e-07, -5.2951e-07, -5.2951e-07, -5.2951e-07, -5.2951e-07,
         9.5871e-08,  8.7890e-07,  4.4735e-07, -1.1621e-06,  4.6280e-07],
        dtype=torch.float64)
tensor(0.0000)
tensor([-4.4520e-08, -4.4520e-08, -4.4520e-08, -4.4520e-08, -4.4520e-08,
        -4.4520e-08, -4.4520e-08, -4.4520e-08, -4.4520e-08, -4.4520e-08,
         4.7985e-08,  2.1580e-08, -9.2922e-09,  1.4257e-07,  3.5014e-07],
        dtype=torch.float64)
tensor(-0.0006)
tensor([-6.3917e-06, -6.3917e-06, -6.3917e-06, -6.3917e-06, -6.3917e-06,
        -6.3917e-06, -6.3917e-06, -6.3917e-06, -6.3917e-06, -6.3917e-06,
         1.0330e-05,  7.8659e-06,  6.1481e-06,  1.2380e-05,  1.0132e-07],
        dtype=torch.float64)
tensor(-0.0002)
tensor([-1.4694e-07, -1.4694e-07, -1.4694e-07, -1.4694e-07, -1.4694e-07,
        -1.4694e-07, -1.4694e-07, -1.4694e-07, -1.4694e-07, -1.4694e-07,
         1.2914e-07,  7.2123e-08,  5.9219e-08,  8.1490e-08,  8.8183e-08],
        dtype=torch.float64)
```

可以看到两者几乎是一致的。（由于精度原因存在少许误差）

## 2.4 模型训练

为了使用 fastnlp 进行训练，我阅读了大量 fastnlp 的源码。在理解了 fastnlp 的 Vocabulary、Dataset、Trainer、Loss、Metric、Callback 后，我了解到了一些使用 fastnlp 中的 Trainer 需要注意的点：

- 数据需要定义“input”、“target”等 field，以便 fastnlp 识别使用的参数
- 模型的输出需要是一个字典，以便 fastnlp 进行参数识别
- 如果需要在训练时进行其它操作，可定义一个继承 callback 的类，并重载相关函数
- 如果需要对模型进行评估，需提供验证集（dev\_data）
- 为了计算相关指标，需提供的一个继承 MetricBase 的类。例如本实验需要计算 perplexity 值，因此我实现了一个 PerplexityMetric 以满足要求。为了防止 perplexity 值溢出，需要先取 log，再进行 exp 操作
- 提供 metric\_key（来自 metric 类的输出）以决定最好的 model

在熟悉 fastnlp 后，只需设置好相关参数，即可进行训练，同时将训练好的模型保存到设置的路径。细节见 train.py。

## 2.5 训练及生成结果

### ● 参数设置：

learning_rate	seq_len	batch_size	input_size	hidden_size	optimizer	patience
1e-3	120	128	512	512	Adam	10

其中 patience 指的是 earllystop 中允许评估指标 metric 连续更差(跟当前最好的相比)的次数。

### ● 训练结果：

**Perplexity: 22.7**

**日：**日往江河浩淼淼，几惨荆州被茫茫。赤霄南将万滩波，闭门开锁四十千。中央共气桃花落，一去习池通重译。紫气早随南汴度，上天星汉三十年。金陵朝逐采香盖，服装物候佩银鞍。一阳宫殿门堪指，莫感娄朱窄羸马。圣寿一为十五年，五言徒振世难玉。仙人圆下似柳垂，十旬岁尽死江湄。

**红：**红鲤鼋鼍管，天寒水云长。边笳送客梦，翅拂到楼中。未断兴将尽，江初渐渐晡。同宫怜得步，有恨舞狂歌。他日班霜节，拂身随去羞。纵能兼上树，清朔可伤情。事别千年镜，晨兢倒露风。梅蹊拱种栗，水土塞虫紫。

**山：**山下逢秋戒，登封忝肩舆。冲融引凤妓，织女指参郎。绮帐姚兰蚁，仙旌到晓珠。错红掉尾鸟，飞妓一丸簪。怪宝论心柁，依依礼选多。故人多志士，世道欲何为？到开絨旂结，垂柳竞纵饶。兹地一通绝，泉清在剡溪。

**夜：**夜长月应妍，越人情不相。况有青山客，不羨幽人境。遂性无姓名，意若何惮遂。我为州田人，尊与浮云废。我病不得期，小夫迷白日。百年取怪笑，渐尔先携手。不复问税人，富心如活女。膏腴利一粒，贫病无足罪。

**湖：**湖结绿光阴，幽窗遥白苹。跻攀容易简，一为取幽襟。六境休直客，闲寻晴阁回。飞溜真薪笔，芝杯度周兰。有封揽为功，自见空山元。豫章情疏实，寂寂坐凉行。机尚不欺璧，甘恋方丈仍。结念天下望，在寻履东邻。

**海：**海国风吹韶，皇楼结云埋。因汎指魏阙，华盖戡咸阳。发匣轻溟直，龙骧万转森。所传念俗格，俯着浮云房。是庆石台存，余宫积暮沙。蹇如垂萝景，喷蔓拂衣裳。受恩似出儒，属孝徒周余。不知私与二，诚已窃其歌。

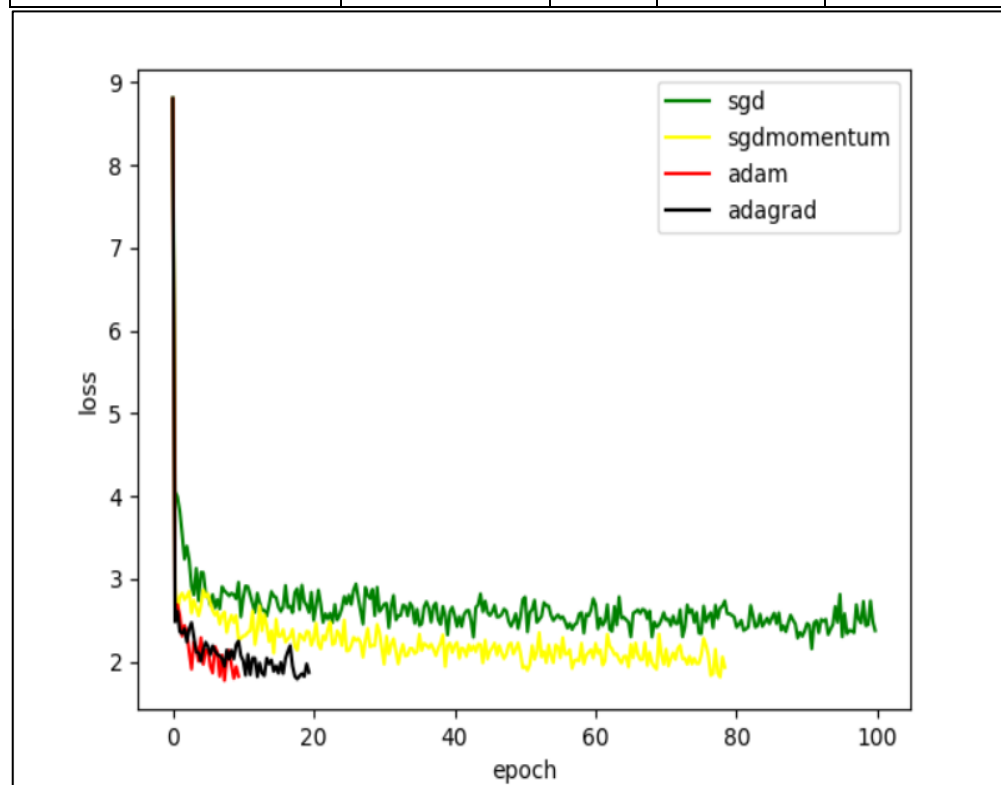
**月：**月黑满城阡，胡无复火城。巨鼇乃绝柱，华嶽发青霄。中乱凝明日，西陵马息寒。云涛唯折桂，魂特一边城。云色连州地，南阳翠树遥。韶看沧浪没，万里烟微连。原野狐吞噬，河寒杀直馨。马融东去笛，沙昊晓郊楼。

## 3. 优化器

### 3.1 优化器对比

本次实验使用了 Adam、Adagrad 、SGD、SGD with momentum 等优化方法，参数设置和模型表现及 loss 曲线如下：

optimizer	learning_rate	loss	perplexity	stop_epoch
SGD	1e-1	2.31	35.9	>200
SGD(momentum=0.9)	1e-1	2.10	27.5	77
Adagrad	1e-2	1.86	23.5	20
Adam	1e-3	1.83	22.7	10



从实验结果来看，SGD 方法收敛速度很慢，未能在我设定的最大 epoch 停止，很可能陷入了鞍点。SGD with momentum 则比 SGD 收敛效果要好，但相比 Adagrad 和 Adam 则逊色很多。Adagrad 和 Adam 均能够很快收敛，但相比较而言 Adam 表现更为稳定，收敛速度也更快。

### 3.2 不同优化方法对梯度计算的影响

由于这些优化方法只是在原有的梯度上增加了一些额外信息，因此只需根据相应的计算公式，对每个变量增加一些额外的参数来保存这些信息即可。我在 LSTM.numpy 中采用了 Adam 算法。核心代码如下所示：

```
for p in self.lstm_cell.get_parameters():
    p.t += 1
    p.m = beta1 * p.m + (1-beta1)*p.d
    p.g = beta2 * p.g + (1-beta2)*p.d*p.d
    mb = p.m/(1-beta1**p.t)
    gb = p.g/(1-beta2**p.t)
    p.v -= learning_rate * mb/(np.sqrt(gb+1e-8))
```

我在 2.3 已经说明了它的正确性。