

PRML Lab3 Report

2019/5/26

工程目录

In assignment3/学号/

---src : 源代码目录

---Config.py : 模型运行参数

---generator : 唐诗生成器

---model.py : 本实验实现的模型

---run.py : 模型的运行文件

---MyUtils/

---dataset.py : 管理模型训练数据

---losses.py : 基于fastnlp实现的Loss类

---metrics.py : 基于fastnlp实现的metric类

---MyLSTM.py : 基于fastnlp和pytorch实现的LSTM类

---out : 生成结果保存位置

---checkpoints

--- best_train_model : 训练的最好模型参数文件

--- bestModel/ : fastnlp的模型保存目录

---data

---datacombine.py 对全唐诗的预处理代码

---tangshi.txt 本实验提供的小数据集

---all_poet.txt 全唐诗数据集

Part 1 - Differentiate LSTM

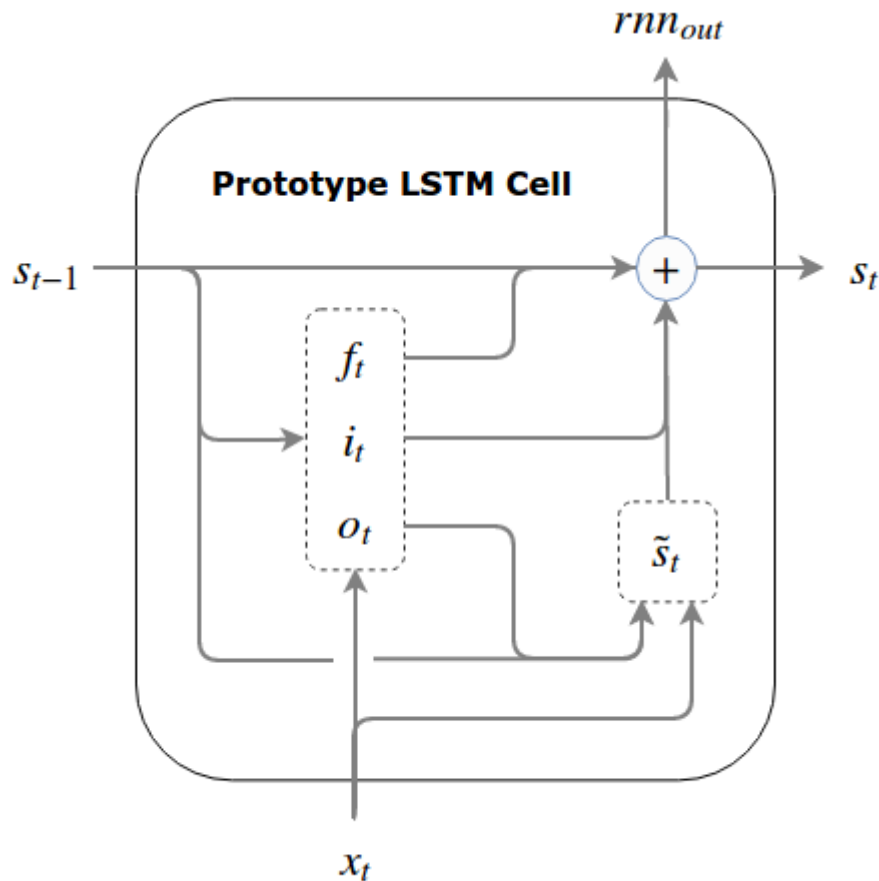
Forward

LSTM是一个时序网络，每一个时间点都经过相同的单元，其参数在同一轮次不发生改变。一个单元的前向传播内容如下：

$$\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}$$

以上参考了torch.nn中LSTM的官方代码和文档：见ref[1]

用 `s1` 表示诗句的长度，则该LSTM unit需要运行 `s1` 次才完成对一个sentence的前向传播。一轮的传播过程中所有参数矩阵不发生改变。该过程的图示如下：



图中 s_t 即表示 h_t

src- ref[2]<https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html>

Backword

task 1: Unit Differentiate

1. Differentiate one step of LSTM with respect to \mathbf{h}_t for

$\mathbf{f}_t, \mathbf{i}_t, \bar{C}_t, C_t, C_{t-1}, \mathbf{o}_t, \mathbf{h}_{t-1}, \mathbf{x}_t, W_f, W_i, W_C, W_o, b_f, b_i, b_C, b_o$. i.e. $\frac{\partial \mathbf{h}_t}{\partial \mathbf{f}_t}$, include

helper

$$\begin{aligned}\tanh(x)' &= 1 - (\tanh(x))^2 \\ \sigma(x)' &= \text{sigmoid}(x)' = \sigma(x)(1 - \sigma(x))\end{aligned}$$

Derivation

$$\frac{\partial(h_t)}{\partial(c_t)} = o_t * (1 - \tanh^2(c_t)) \quad (0)$$

$$\frac{\partial(h_t)}{\partial(o_t)} = \tanh(c_t) \quad (1)$$

$$\frac{\partial(h_t)}{\partial(\tilde{c}_t)} = \frac{\partial(h_t)}{\partial(c_t)} * \frac{\partial(c_t)}{\partial(\tilde{c}_t)} = \frac{\partial(h_t)}{\partial(c_t)} * (i_t) = o_t * (1 - \tanh^2(c_t)) * i_t \quad (2)$$

$$\frac{\partial(h_t)}{\partial(c_{t-1})} = \frac{\partial(h_t)}{\partial(c_t)} * \frac{\partial(c_t)}{\partial(c_{t-1})} = \frac{\partial(h_t)}{\partial(c_t)} * (f_t) = o_t * (1 - \tanh^2(c_t)) * f_t \quad (3)$$

$$\frac{\partial(h_t)}{\partial(i_t)} = \frac{\partial(h_t)}{\partial(c_t)} * \frac{\partial(c_t)}{\partial(i_t)} = \frac{\partial(h_t)}{\partial(c_t)} * \tilde{c}_t = o_t * (1 - \tanh^2(c_t)) * \tilde{c}_t \quad (4)$$

$$\frac{\partial(h_t)}{\partial(f_t)} = \frac{\partial(h_t)}{\partial(c_t)} * \frac{\partial(c_t)}{\partial(f_t)} = \frac{\partial(h_t)}{\partial(c_t)} * c_{t-1} = o_t * (1 - \tanh^2(c_t)) * c_{t-1} \quad (5)$$

$$\frac{\partial(h_t)}{\partial(z)} \quad (6)$$

$$\begin{aligned}&= \left[\frac{\partial(h_t)}{\partial(h_{t-1})}, \frac{\partial(h_t)}{\partial(x_t)} \right] \\&= \frac{\partial(h_t)}{\partial(o_t)} \frac{\partial(o_t)}{\partial(z)} + \frac{\partial(h_t)}{\partial(c_t)} \frac{\partial(c_t)}{\partial(z)} \\&= W_o^T \cdot \left[\frac{\partial(h_t)}{\partial(o_t)} o_t (1 - o_t) \right] + \frac{\partial(h_t)}{\partial(c_t)} \left(\frac{\partial(c_t)}{\partial(f_t)} \frac{\partial(f_t)}{\partial(z)} + \frac{\partial(c_t)}{\partial(i_t)} \frac{\partial(i_t)}{\partial(z)} + \frac{\partial(c_t)}{\partial(\tilde{c}_t)} \frac{\partial(\tilde{c}_t)}{\partial(z)} \right) \\&= W_o^T \cdot \left[\frac{\partial(h_t)}{\partial(o_t)} * o_t * (1 - o_t) \right] + \\&\quad \{ W_f^T \cdot \frac{\partial(c_t)}{\partial(f_t)} * f_t * (1 - f_t) + W_i^T \cdot \frac{\partial(c_t)}{\partial(i_t)} * i_t * (1 - i_t) + W_c^T \cdot \frac{\partial(c_t)}{\partial(\tilde{c}_t)} * (1 - \tilde{c}_t^2) \} * \frac{\partial(h_t)}{\partial(c_t)} \\&= W_c^T \cdot [\tanh(c_t) * o_t * (1 - o_t)] + \\&\quad \{ W_f^T \cdot [c_{t-1} * f_t * (1 - f_t)] + W_i^T \cdot [\tilde{c}_t * i_t * (1 - i_t)] + W_c^T \cdot [i_t * (1 - \tilde{c}_t^2)] \} * o_t * (1 - \tanh^2(c_t))\end{aligned}$$

$$\frac{\partial(h_t)}{\partial(W_f)} = \frac{\partial(h_t)}{\partial(f_t)} \frac{\partial(f_t)}{\partial(W_f)} = [o_t * (1 - \tanh^2(c_t)) * c_{t-1} * f_t * (1 - f_t)] \cdot z^T \quad (7)$$

$$\frac{\partial(h_t)}{\partial(W_i)} = \frac{\partial(h_t)}{\partial(i_t)} \frac{\partial(i_t)}{\partial(W_i)} = [o_t * (1 - \tanh^2(c_t)) * \tilde{c}_t * i_t * (1 - i_t)] \cdot z^T \quad (8)$$

$$\frac{\partial(h_t)}{\partial(W_c)} = \frac{\partial(h_t)}{\partial(\tilde{c}_t)} \frac{\partial(\tilde{c}_t)}{\partial(W_c)} = [o_t * (1 - \tanh^2(c_t)) * i_t * (1 - \tilde{c}_t^2)] \cdot z^T \quad (9)$$

$$\frac{\partial(h_t)}{\partial(W_o)} = \frac{\partial(h_t)}{\partial(o_t)} \frac{\partial(o_t)}{\partial(W_o)} = [\tanh(c_t) * o_t * (1 - o_t)] \cdot z^T \quad (10)$$

$$\frac{\partial(h_t)}{\partial(b_f)} = \frac{\partial(h_t)}{\partial(f_t)} \frac{\partial(f_t)}{\partial(b_f)} = o_t * (1 - \tanh^2(c_t)) * c_{t-1} * f_t * (1 - f_t) \quad (11)$$

$$\frac{\partial(h_t)}{\partial(b_i)} = \frac{\partial(h_t)}{\partial(i_t)} \frac{\partial(i_t)}{\partial(b_i)} = o_t * (1 - \tanh^2(c_t)) * \tilde{c}_t * i_t * (1 - i_t) \quad (12)$$

$$\frac{\partial(h_t)}{\partial(b_c)} = \frac{\partial(h_t)}{\partial(\tilde{c}_t)} \frac{\partial(\tilde{c}_t)}{\partial(b_c)} = o_t * (1 - \tanh^2(c_t)) * i_t * (1 - \tilde{c}_t^2) \quad (13)$$

$$\frac{\partial(h_t)}{\partial(b_o)} = \frac{\partial(h_t)}{\partial(o_t)} \frac{\partial(o_t)}{\partial(b_o)} = \tanh(c_t) * o_t * (1 - o_t) \quad (14)$$

task 2: Differentiate through time

下面考虑在一个step中的导数求解。使用Lab3 description中的符号表，记词表大小为 $N = |V|$ ，序列长度为 $T = sl$ 。记单步单变量的总Loss为 L ，在时刻 t 的loss为 l_t ，并记 t 时刻及 t 时刻以后的loss为 \tilde{L}_t ，则有：

$$\tilde{L}_t = \sum_{k=t}^T l_k; \quad L = \sum_{k=1}^T l_k$$

其中有： $l_t = y_t^T \cdot \log(\hat{y}_t)$, y_t 是真实值， \hat{y}_t 为预测值，二者皆为 $N \times 1$ 向量
 $\hat{y}_t = \sigma(W_p \cdot h_t + b_p)$

对于 l_k ，其只与 $t \leq k$ 时刻的 h_t 相关，因而有：

$$\frac{\partial L}{\partial h_t} = \frac{\partial \tilde{L}_t}{\partial h_t} = \frac{\partial l_t}{\partial h_t} + \frac{\partial L_{t+1}}{\partial h_t} = \frac{\partial l_t}{\partial h_t} + \frac{\partial L_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} = W^T [y^T (1 - \hat{y}_t)] + \frac{\partial L_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t}$$

所以有递推公式如下：

$$\begin{aligned} \frac{\partial L}{\partial h_t} &= W^T [y^T (1 - \hat{y}_t)] + \frac{\partial L_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t}, \text{ while } t < T \\ &= \frac{\partial l_t}{\partial h_t} = W^T [y^T (1 - \hat{y}_t)], \text{ while } t = T \end{aligned}$$

$\frac{\partial h_{t+1}}{\partial h_t}$ 已经在前一部分求得(6)，所以可以通过它计算所有的 $\frac{\partial L}{\partial h_t}$

同时对于 $\frac{\partial L}{\partial W_*}$ 和 $\frac{\partial L}{\partial b_*}$ ，我们可以通过下述关系求解：

$$\frac{\partial L}{\partial W_*} = \sum_{t=1}^T \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial W_*}$$

$$\frac{\partial L}{\partial b_*} = \sum_{t=1}^T \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial b_*}$$

Part 2 - Autograd Training of LSTM

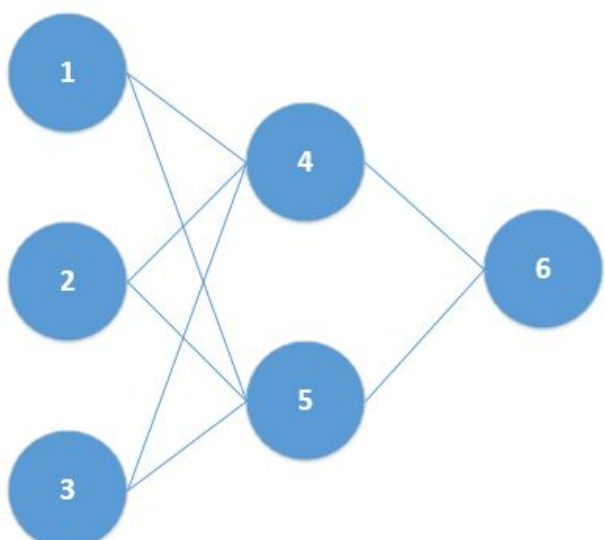
task 1: Initialization

Why we should not just initialize them to zero?

对于单层的神经网络来说，全零初始化方法是基本不会有问题的，但是当网络的层数增多，全零初始化就会有很大问题。全零初始化会导致第一次前向传播得到的所有隐藏层和输出层的值皆为0，从而导致在反向传播的过程中网络学不到知识（具体可体现在不同维度的参数更新值相同，或者大量参数得不到更新），从而模型的效果很差。

下面参考`ref[3]`，通过一个例子来阐释这件事：

假设我们要训练这样一个简单的网络，并给予右图的初始化参数：

网络结构	参数矩阵及初始化
	$W1 = \begin{bmatrix} w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $W2 = [w_{64} \quad w_{65}] = [0 \quad 0]^T$

可以得出第一次前向传播后

$$z_4 = W_{41} * x_1 + W_{42} * x_2 + W_{43} * x_3 = 0; z_5 = W_{51} * x_1 + W_{52} * x_2 + W_{53} * x_3 = 0$$

使用激活函数后有 $a_4 = f(z_4)$ $a_5 = f(z_5)$,有 $a_4 = a_5 = f(0)$, 网络输出 $a_6 = W_{64} * a_4 + W_{65} * a_5 = 0$

定义Loss为 $Loss = \frac{1}{2} (y - a_6)^2$

反向传播后，我们可以知道节点4，5的梯度相同，因而更新后有

$$w_{64} = 0 + \Delta w = \Delta w$$

$$w_{65} = 0 + \Delta w = \Delta w$$

同理亦可得 $W_{41} = W_{42} = W_{43} = W_{51} = W_{52} = W_{53}$,而且由于此次相同,后边的所有轮次,他们依然相同,这就导致每一层中所有节点学到的是一样的,导致网络学习能力巨差。

Methods to Initialization

除了全零初始化,神经网络有很多初始化方法,如下:

- 随机初始化方法
- 正态初始化方法
- Xavier 初始化方法
- He 初始化方法

根据Pytorch官方文档的说明[ref1],pytorch提供了三种初始化方法,分别为常数初始化、正态初始化和Xavier初始化,在Parameter类使用的初始化方法为正太初始化法,其中nn.Embedding采用了标准正态分布进行初始化,而LSTM类的各个参数矩阵采用的是 $\mu(-\sqrt{k}, \sqrt{k})$,其中 $k = \frac{1}{\text{hidden_size}}$

• NOTE

All the weights and biases are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{\text{hidden_size}}$

picture src:<https://pytorch.org/docs/stable/nn.html>

本实验虽然没有直接采用Pytorch的LSTM实现,但是依然采用了其一些Linear结构等基本结构,这些结构的参数初始化方法与nn.LSTM的初始化方法相同。

task 2: Generating

DataSet	lr	optim	bs	embed_dim	hidden	sl	epoch	ppl
tangshi.txt	1e-3	Adam	256	256	256	50	289	84.7134
全唐诗	1e-3	Adam	256	256	256	50	132	37.4163
tangshi.txt	1e-3	Adam	32	256	256	50	200	18.657

以下为小数据集（本实验提供的数据）的生成结果

- 日敛红烟湿艳姿，行云飞去明星稀。目似击，接要心已领。仿佛识鲛人，开轩琴月孤。
- 红三百首甘，山腰度石关。屡迷青嶂合，时爱绿萝闲。宴息花林下，高谈竹屿间。寥寥隔尘事，疑是入鸡山。
- 山河县柳林边，河桥晚泊船。文叨才子会，官喜故人连。笑语同今夕，轻肥异往年。晨风理归棹，吴楚各依然。
- 夜将进酒，将进酒，酒中有毒鸩主父，言之主父伤主母。母为妾地父妾天，仰天俯地不忍言。佯为僵踣主父前，主
- 湖醉渡十年春，牛渚山边六问津。历阳前事知何实，高位纷纷见陷人。
- 海亭秋日望，委曲见江山。江山历全楚，河洛越成周。道路疲千里，乡园老一丘。知君命不偶，同病亦同忧。
- 月青丝暮雨，万株杨柳拂波垂。蒲根水暖雁初浴，梅径香寒蜂未知。辞客倚风吟暗淡，使君回马湿旌旗。江上列，

以下为全唐诗的生成结果

- 日月長江夜，風流水上春。山中無定處，雲路杳難尋。山色連天外，雲帆落照中。

- 红明天上，天子建寅。九重重重慶，金殿敞延英。風雪初晴日，高樓望月明。雲開山色暮，雲樹夕陽天。
- 山中有客兮，不可忘。氓畢屠，綏者耻。我有疆，螯賊生。不見人，不用武，不用武。不知何處，于之之。
- 夜雨滴滴滴，滴滴滴滴滴。不知何處去，不是人間事。
- 湖上月，犁牛角，城頭角，弓箭射。不見沙，北邙路岐路。行人不得見，死處不可見。君王不相識，不是他人哭。
- 海上千里道，人間萬里遊。山中有人事，江上無人期。一別一時別，孤舟獨自歸。山中無定處，不見白雲歸。
- 月落照江水，清風吹玉笛。清風吹玉笛，吹管弦歌聲。夜夜月中夜，風吹玉笛聲。

可以发现，数据集增大对于模型的改进有很大帮助，全唐诗生成的结果要通顺很多。

对全唐诗的预处理

全唐诗的数据来源参考了Lab3 Description中给定的网址，获取数据后，经过简单的分析发现，数据集中有一些噪音，主要在两个方面：

1. 部分Json的paragraphs内容只有一个句号
2. 很多Json的paragraphs属性中包含了注释等内容，如以括号开头的诗歌创作背景等

前一个问题不大，因为57000首诗只有很少部分存在1情况，但是后一种问题较大，最后生成的诗歌也有很多注释，这并不是我们想要的（而且由于注释的括号的存在，导致注释的部分看起来很奇怪）。所以预处理部分去除了包括括号在内的大量注释型符号及注释内容。预处理的代码可详见 `data/datacombine.py`。

同时不管是全唐诗还是小数据集，可以发现大部分诗歌的长度都在50左右，50-100间的不到10%，高于100的微乎其微，所以训练时设定max_seq_len为50，最长的生成结果max_gen_len也设置为50.超过的去掉，不够的在前面补<pad>

task 3: Optimization

下面的优化器都是在小数据集上使用的（全唐诗训起来太耗时了）

optimizer模型	parameter	bacth	epcoh	loss	ppl
Adam	lr=0.001,betas=(0.9, 0.999), eps=1e-08, weight_decay=0,amsgrad=False	32	200	0.593	18.6507
SGD with momentum	lr=0.001,momentum=0.8	32	97(early-stop)	7.4960	1806.2608
SGD with momentum	lr=0.001,momentum=0.8	8	200	6.2929	565.3969
Nesterov	lr=0.001,momentum=0.8	8	200	6.3059	565.3531
Adadelat=	lr=1.0,rho=0.9,eps=1e-6,weight_decay=0	32	200	2.4923	41.2261

比较：在相同的周期下，Adam的收敛速度要大于Adadelat，同时模型的ppl值也更好，但让这里可能也有参数设置的原因，但时间关系，没有进行进一步的调优。而SGD算法的表现不是很好，采用batch_size=32,很快就early stop了(设置patience=10，即10个周期ppl不更好)，于是把batch_size调为8，发现也不太好，模型生成大量padding,且loss下降极慢。原因在于数据集太小SGD不是很适用。（只与early stop的原因，应该是由于SGD本身的抖动性，导致模型early stop过早），Neaterov使用了Pytorch.optimizer.SGD中是能nesterov的方法实现，因为他是基于SGD，其实本质上在本实验上效果也不好。（他们生成的结果都很短）

Appendix

Bonus part

使用了全唐诗进行训练，结果前面已经放出，数据的预处理等部分在data/文件夹下。（时间关系，numpy版本的还没写完，如果后其实现后，将会补充上去）

Reference

[1]. <https://pytorch.org/docs/stable/nn.html>

[2]. <https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html>

[3]. [≥](#)

[4]. https://blog.csdn.net/weixin_39845112/article/details/80045091

Classification

本次实验基于pytorch实现了LSTM模块，并分别实现了pytorch的运行框架和fastnlp的结构。本实验的过程中主要使用的是pytorch版本的运行结果（为了方便理解内部运行方式）。fastnlp的版本已实现并通过测试。

在运行时，可以选择不同的运行方式: `python run.py --mode[train, train_torch_lstm, train_fastnlp]`。其中的 *train_torch_lstm* 为直接使用 `torch.LSTM` 模块的运行方法，其存在意义在于检验模型正确性。

为了更加方便管理代码，本实验借鉴了网上的一些神经网络代码组织方法，使用了Config.py以及parser模型进行参数传递，方便高效训练。

本实验的代码baseline参考了ref[4]这边博客的内容，他是一本教程书中的结构，我在参考它的结构的基础上，手动实现了所有逻辑。其中生成部分，本实验要求的生成方式由自己码出来，藏头诗部分直接copy的源代码。另外LSTM的实现参考了Pytorch的源码，由于目前的pytorch该部分代码是C++实现的，所以，也阅读了相关的文档与解读。

由于这是第一次使用Pytorch和fastnlp，在学习上耗费了大量的时间，不过好在两个工具的源码和文档都很清晰，所以学起来虽然耗时却不那么痛苦。