PRML - lab3 - LSTM

Overview

本次唐诗生成的实验充分展现了LSTM的强大。实验中，对LSTM的求导、LSTM的手写实现、梯度计算与反向传播的numpy实现均是富有挑战性的工作，也让我对LSTM有了更为深刻的认识。在基本工作外，我探究了后起之秀GRU在唐诗生成任务中的表现。

Part 1

Requirement 1: Differentiate one step of LSTM with respect to h_t

本任务对h_t的求导不涉及高难度的知识，但计算量较大，并运用如下几个基本公式：

- $\frac{\partial tanh(x)}{\partial x} = (1 - tanh^2(x))$
- $\frac{\partial sigmoid(x)}{\partial x} = sigmoid(x)(1 - sigmoid(x))$

具体运算过程与结果如下：

- $\frac{\partial h_t}{\partial f_t} = o_t * \frac{\partial tanh(C_t)}{\partial f_t} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial f_t} = o_t * C_{t-1} * (1 - tanh^2(C_t))$
- $\frac{\partial h_t}{\partial i_t} = o_t * \frac{\partial tanh(C_t)}{\partial i_t} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial i_t} = o_t * \bar{C}_t * （1 - tanh^2(C_t))$
- $\frac{\partial h_t}{\partial \bar{C}_t} = o_t * \frac{\partial tanh(C_t)}{\partial \bar{C}_t} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial \bar{C}_t} = o_t * i_t * （1 - tanh^2(C_t))$
- $\frac{\partial h_t}{\partial C_t} = o_t * \frac{\partial tanh(C_t)}{\partial C_t} = o_t * (1 - tanh^2(C_t))$
- $\frac{\partial h_t}{\partial C_{t-1}} = o_t * \frac{\partial tanh(C_t)}{\partial C_{t-1}} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial C_{t-1}} = o_t * f_t * (1 - tanh^2(C_t))$
- $\frac{\partial h_t}{\partial o_t} = tanh(C_t) * \frac{\partial o_t}{\partial o_t} = tanh(C_t)$
- $$\begin{aligned}\frac{\partial h_t}{\partial h_{t-1}} &= o_t * \frac{\partial tanh(C_t)}{\partial h_{t-1}} + tanh(C_t) * \frac{\partial o_t}{\partial h_{t-1}} \\ &= o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial h_{t-1}} + tanh(C_t) * \frac{\partial \sigma(W_o \cdot z + b_o)}{\partial h_{t-1}} \\ &= o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial h_{t-1}} + tanh(C_t) * \frac{\partial \sigma(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + b_o)}{\partial h_{t-1}} \\ &= o_t * [1 - tanh^2(C_t)] * [W_{hf} * f_t * (1 - f_t) * C_{t-1} + W_{hi} * i_t * (1 - i_t) * \bar{C}_t \\ &\quad + i_t * W_{hC} * (1 - \bar{C}_t{}^2)] + tanh(C_t) * W_{ho} * o_t * (1 - o_t)\end{aligned}$$
- $$\begin{aligned}\frac{\partial h_t}{\partial x_t} &= o_t * \frac{\partial tanh(C_t)}{\partial x_t} + tanh(C_t) * \frac{\partial o_t}{\partial x_t} \\ &= o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial x_t} + tanh(C_t) * \frac{\partial \sigma(W_o \cdot z + b_o)}{\partial x_t}\end{aligned}$$

$$= o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial x_t} + tanh(C_t) * \frac{\partial \sigma(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + b_o)}{\partial x_t}$$

$$= o_t * [1 - tanh^2(C_t)] * [W_{xf} * f_t * (1 - f_t) * C_{t-1} + W_{xi} * i_t * (1 - i_t) * \bar{C}_t$$

$$+ i_t * W_{xC} * (1 - {\bar{C}_t}^2)] + tanh(C_t) * W_{xo} * o_t * (1 - o_t)$$

- $\frac{\partial h_t}{\partial W_f} = o_t * \frac{\partial tanh(C_t)}{\partial W_f} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial W_f}$

  $= o_t * \frac{\partial tanh(\sigma(W_f \cdot z + b_f) * C_{t-1} + i_t * \bar{C}_t)}{\partial W_f} = o_t * C_{t-1} * z * f_t * (1 - f_t) * [1 - tanh^2(C_t)]$

- $\frac{\partial h_t}{\partial W_i} = o_t * \frac{\partial tanh(C_t)}{\partial W_i} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial W_i}$

  $= o_t * \frac{\partial tanh(f_t * C_{t-1} + \sigma(W_i \cdot z + b_i) * \bar{C}_t)}{\partial W_i} = o_t * \bar{C}_t * z * i_t * (1 - i_t) * [1 - tanh^2(C_t)]$

- $\frac{\partial h_t}{\partial W_C} = o_t * \frac{\partial tanh(C_t)}{\partial W_C} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial W_C}$

  $= o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * tanh(W_C \cdot z + b_C))}{\partial W_C} = o_t * i_t * z * (1 - {\bar{C}_t}^2) * [1 - tanh^2(C_t)]$

- $\frac{\partial h_t}{\partial W_o} = tanh(C_t) * \frac{\partial o_t}{\partial W_o} = tanh(C_t) * \frac{\partial \sigma(W_o \cdot z + b_o)}{\partial W_o}$

  $= tanh(C_t) * z * o_t * (1 - o_t)$

- $\frac{\partial h_t}{\partial b_f} = o_t * \frac{\partial tanh(C_t)}{\partial b_f} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial b_f}$

  $= o_t * \frac{\partial tanh(\sigma(W_f \cdot z + b_f) * C_{t-1} + i_t * \bar{C}_t)}{\partial b_f} = o_t * C_{t-1} * f_t * (1 - f_t) * [1 - tanh^2(C_t)]$

- $\frac{\partial h_t}{\partial b_i} = o_t * \frac{\partial tanh(C_t)}{\partial b_i} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial b_i}$

  $= o_t * \frac{\partial tanh(f_t * C_{t-1} + \sigma(W_i \cdot z + b_i) * \bar{C}_t)}{\partial b_i} = o_t * \bar{C}_t * i_t * (1 - i_t) * [1 - tanh^2(C_t)]$

- $\frac{\partial h_t}{\partial b_C} = o_t * \frac{\partial tanh(C_t)}{\partial b_C} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial b_C}$

  $= o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * tanh(W_C \cdot z + b_C))}{\partial b_C} = o_t * i_t * (1 - {\bar{C}_t}^2) * [1 - tanh^2(C_t)]$

- $\frac{\partial h_t}{\partial b_o} = tanh(C_t) * \frac{\partial o_t}{\partial b_o} = tanh(C_t) * \frac{\partial \sigma(W_o \cdot z + b_o)}{\partial b_o}$

  $= tanh(C_t) * o_t * (1 - o_t)$

Requirement 2: Back Propagation Through Time

时序反向传播算法要求我们不仅仅考虑单步的反向传播，而是将时序因素放到反向传播的过程中，借助隐藏状态h_t与c_t帮助进行反向传播公式的推导。

- 首先，我们尝试计算出h_t与c_t的梯度：

  $\delta_h^t = W^T(\hat{y} - y)$

  $\delta_c = \frac{\partial L}{\partial C^{(t+1)}} \frac{\partial C^{(t+1)}}{\partial C^{(t)}} + \frac{\partial L}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial C^{(t)}}$

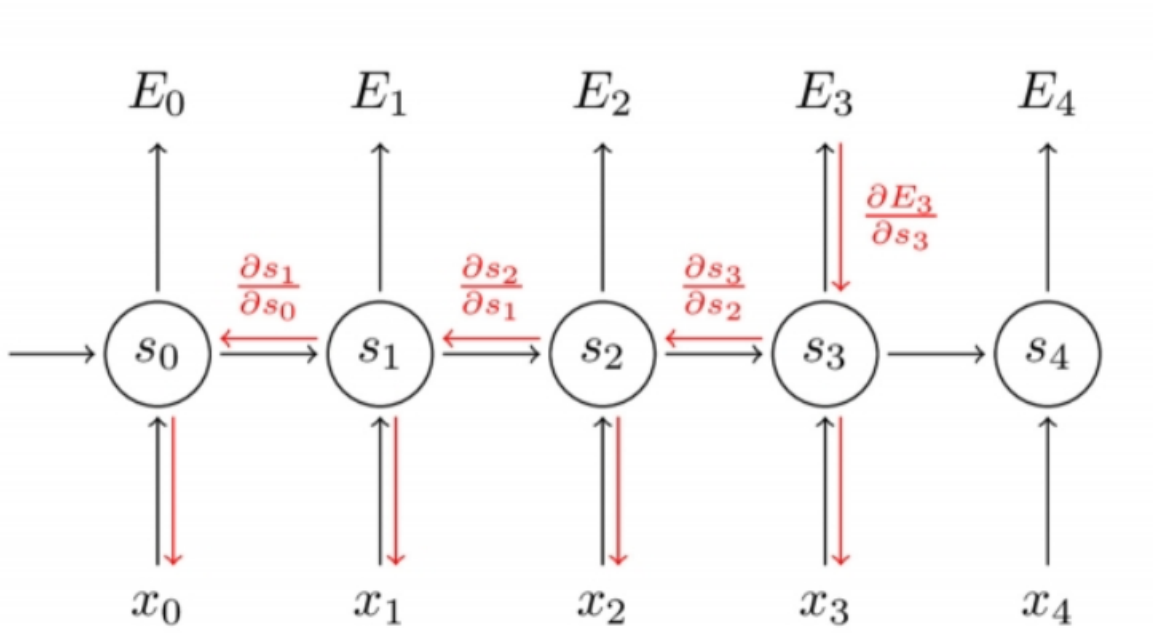
$$= \delta_C^{(t+1)} \odot f^{(t+1)} + \delta_h^{(t)} \odot o^{(t)} \odot (1 - tanh^2(C^{(t)}))$$

我们发现，c_t起到了时序反向传播的作用，而h_t只作用于当前层。

- 接着，我们利用h_t与c_t推导其它需要计算梯度的参数公式；
- 在实际问题中，我们需要在正向传播后存储各中间状态，包括h_t, f_t, C_t等。对于输出序列，我们利用上述提到的各梯度参数计算手段从后向前计算梯度并更新参数，达到优化模型的效果。如：

$$\frac{\partial L}{\partial W_f} = \sum_{t=1}^{tfinal} \frac{\partial L}{\partial C_f^t} \odot \frac{\partial C_f^t}{\partial f^t} \odot \frac{\partial f^t}{\partial W_o}$$

下面给出一个宏观的BPTT链式求导图辅助理解（图片来自网络）：



Part 2

Requirement 1: Initialization

在embedding层面，我直接选取torch.nn.embedding帮助操作，故在此未涉及初始化问题；在模型层面，涉及到一系列W_x, b_x参数的初始化。参数的初始化可以理解为模型的"起跑线"。参数全零初始化不是一种好的选择：经过正向传播与反向传播后，参数的不同维度更新相同，导致模型性能很低。一个好的参数初始化能够加速收敛，达到好的泛化效果，而不导致梯度消失与梯度下降。当下的深度学习模型有几种流行的初始化方法：

- 随机初始化：随机初始化权重。这是简单而常用的方法，但一旦随机分布选择不当，随机初始化的参数落在无梯度或低梯度的范围内，会有较差的表现。因此，在实际应用中，随机初始化往往需要变形。
- 标准初始化：$W_{i,j} \sim U(-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}})$。标准初始化保证输出具有相同的分布，提高训练的收敛速度。
- Xavier初始化：$W_{i,j} \sim U(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}})$。保持输入和输出的方差一致，但是没有考虑激活函数对输出数据分布的影响。
- He初始化： $W_{i,j} \sim U(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}})$。 考虑了RELU对输出数据分布的影响，保证输入和输出的方差一致，是一种相对高性能的初始化方法。
- Pre-train初始化：即fine-tuning方法，将预训练模型的参数作为模型的初始化，相对以上几种方法具有更好的表现，在现阶段深度学习中具有广泛的应用。

在本实验中，我采用类标准化方法初始化二维矩阵参数，保证矩阵各参数在梯度范围内：

```
W(size1, size2) = torch.rand(size1,size2) * np.sqrt(2/(size1 + size2))
```

Requirement 2: Generating Tang Poem

Preprocessing

Dataset

在本次实验中，我选取了全唐诗作为数据集，并进行了去除标点、化繁体为简体的清洗操作。在清洗完成后，将数据集按8：2划分为测试集（46090条）与验证集（11523条）。

Vocabulary

在构建词表上，我采用fastNLP的Vocabulary实现更快速而简便的操作，如通过vocab.add()添加词、vocab.to_index()与vocab.to_word()实现词与index的转换等，在此不做赘述。

Model

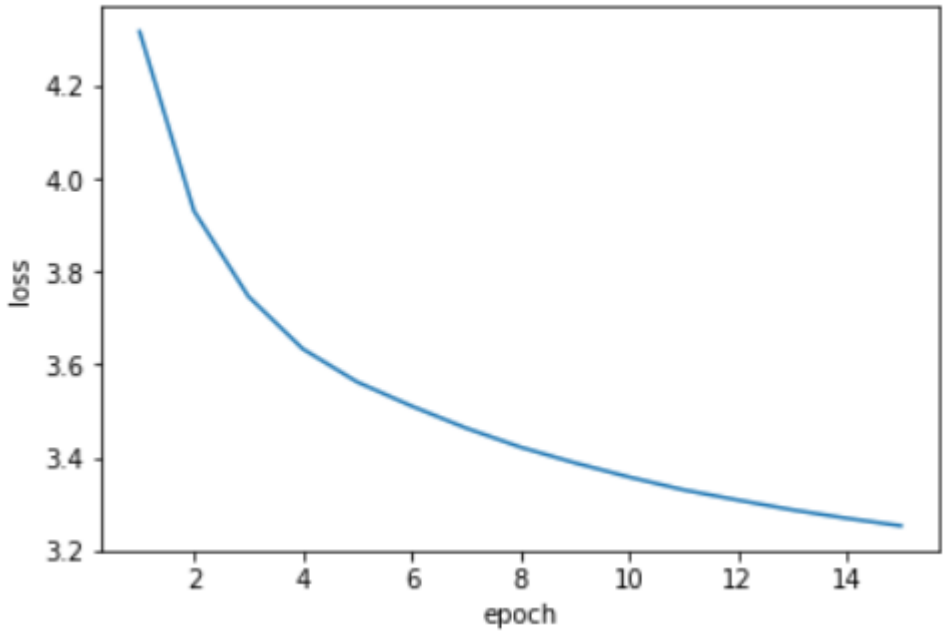
在模型的构建中，有如下几个值得注意的小细节：

- softmax层的使用与否。为了将输出转化为概率，我们希望采用softmax层达到目标。但由于pytorch的crossEntropyLoss已经内置地帮我们完成此操作，在模型输出时不必多套用一层softmax以影响计算时间（虽然对结果没有本质影响），而只有当我们在其它需要使用概率的时候再使用softmax层。
- 何时转置矩阵、是否转置矩阵。这是看起来很简单但非常耗时的工作，通过在草稿纸上标出各矩阵形状并在程序运行过程中实时输出矩阵shape会帮助我们减少错误。
- early stop机制的使用。为了防止过拟合、更高效率地训练模型，我们引入perplexity度量模型的好坏，并在perplexity不再提升时停止训练。

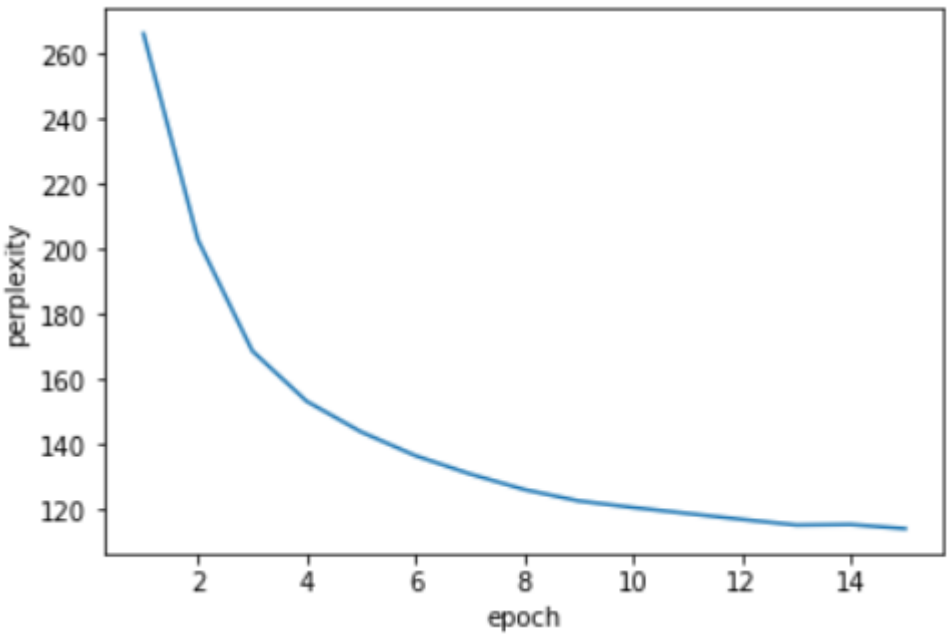
在最终实验上，各参数的选取如下：

- Vocabulary Size: 9250
- Batch Size: 64
- Sentence Length: 80
- Hidden Size: 128
- Cell Size: 128
- Input Size(Embedding Size):128
- learning rate: 0.001

模型的loss随epoch的变化如下：



验证集的perplexity随epoch的变化如下：



可以看出，训练集loss的下降与验证机perplexity的下降是基本对应的，这也在某种程度上验证了perplexity计算方法的正确性。

值得注意的是，验证集的perplexity相较训练集仍然有较大差距，如14个epoch后对部分训练集的perplexity如下图所示，均在50以下：

```
Epoch: 14 batch index:712 loss: 3.182948 perplexity:24.11774445
日日初生雪花深满院扉白石黄昏日照尘埃月明
<class 'torch.Tensor'>
Epoch: 14 batch index:713 loss: 3.209539 perplexity:24.76766968
日日高楼上西阳入户宫女里新春酒酒醒酒杯中
<class 'torch.Tensor'>
Epoch: 14 batch index:714 loss: 3.557715 perplexity:35.08295441
日暮暮春来日夕何人去相见独不穷归思无事处
<class 'torch.Tensor'>
Epoch: 14 batch index:715 loss: 3.487598 perplexity:32.70728683
日落东城春尽是长安一家家子落花来水水清流
```

这也是显然的，模型的泛化能力还有待提升。

Generation

在实验说明中，提到了通过改变temperature term的方式增加诗歌产生的多样性。在实验中，我也尝试了另一种为了提高多样性的手段，即对某一输入选取概率最高的10个输出，从中挑选一个作为实际输出结果并重新放入模型进行下一个字的产生，也达到了不错的诗歌产生效果。

日（五言）：日照南湖月，照寒风流夜。水连秋雁寒，来夜月中孤。

日（七言）：日照南州县北望，云外秋天上清阴。远近烟霞日落秋，风日晚树苍龙鸟。

红（五言）：红烛满楼春，风吹露滴红。莲花下红豔，妆不得归心。

红（七言）：红尘中夜雨一生，春日晚花边春水。满城中月照新声，怨恨啼残雪愁肠。

山（五言）：山寺一片云，飞雪满池波。浪断船江岸，烟雨露秋云。

山（七言）：山下水边秋夜来，无定处云端月落。花满目流光不如，云外路遥遥望山。

夜（五言）：夜月秋光照，寒灯白草深。空思长风急，吹箫鼓鼙鼓。

夜（七言）：夜月高低复相望，空房秋水上青天。一日回头宿夜风，起来迟雨夜愁猿。

湖（五言）：湖上一叶花，秋草绿阴野。岸斜日夜寒，山静云林鸟。

湖（七言）：湖山水木落青枫，树阴满岸春烟水。岸头人生一路傍，林间不堪归去去。

海（五言）：海风飘叶落，花时满城南。山里一夜江，城客一年花。

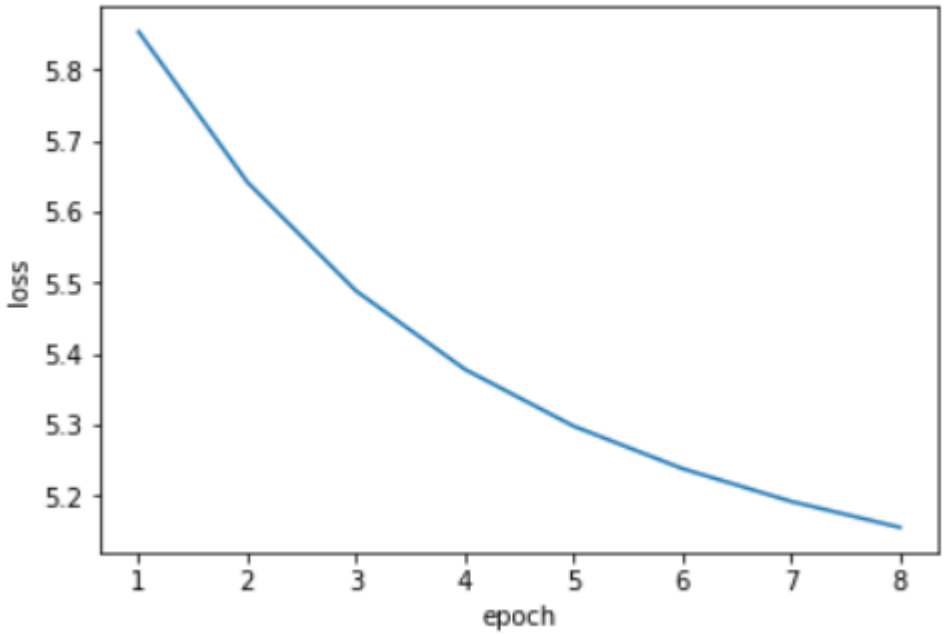
海（七言）：海日长城路人家，万壑寒无心已自。有风生自得长城，窟何年事不知此。

月（五言）：月满秋光照，白苹花下绿。阴生野色满，牀香满眼开。

月（七言）：月出西楼月下流，莺鸣风烟暖满头。春酒一壶头不醉，狂人不敢辞相看。

numpy version

在numpy.py中，给出了numpy版本的LSTM，其主要难点在于BPTT的实现。为了验证numpy版本的正确性，我们同样绘制了numpy版本的loss曲线图：



可以看到，在8个epoch以后loss下降到了5左右，整体的下降趋势与pytorch版本较为接近，但仍然有区别。区别的原因主要有二，一方面是因为采取了数据量较小的验证集作为模型输入（采取与pytorch相同的输入规模时，pytorch版本的15个epoch跑完后numpy版本才进行了3个epoch，训练极为缓慢，故为提高效率采取了较小的验证集）；另一方面在于Optimization方式的朴素，没有引入动量等提高训练速率的方式。

虽然在8个epoch后loss较高且没有下降到平缓的位置，numpy版本生成的唐诗也初显make sense的一面（虽然相比pytorch版本还有不小差距），给出部分生成的结果：

湖：湖自月间生，年君是月薄。处年君处草，卯有一处草。

月：月素家是日，月鹅暮草俪。天白白有山，归家知处时。

Requirement 3: Optimization

在本次实验中，选取了Adam与Nesterov进行对比。下面简要描述此两种方法的区别：

- Adam：利用梯度的一阶矩估计和二阶矩估计动态调整学习率，使每一次迭代学习率在确定范围内：

$$m_t = \mu * m_{t-1} + (1 - \mu) * g_t$$

$$n_t = \nu * n_{t-1} + (1 - \nu) * g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \mu^t}$$

$$\hat{n}_t = \frac{n_t}{1 - \nu^t}$$

$$\Delta\theta_t = -\frac{\hat{m}_t}{\sqrt{\hat{n}_t + \epsilon}} * \eta$$

- Nesterov：梯度在大的跳跃后，修正当前梯度，避免前进太快，同时提高灵敏度：

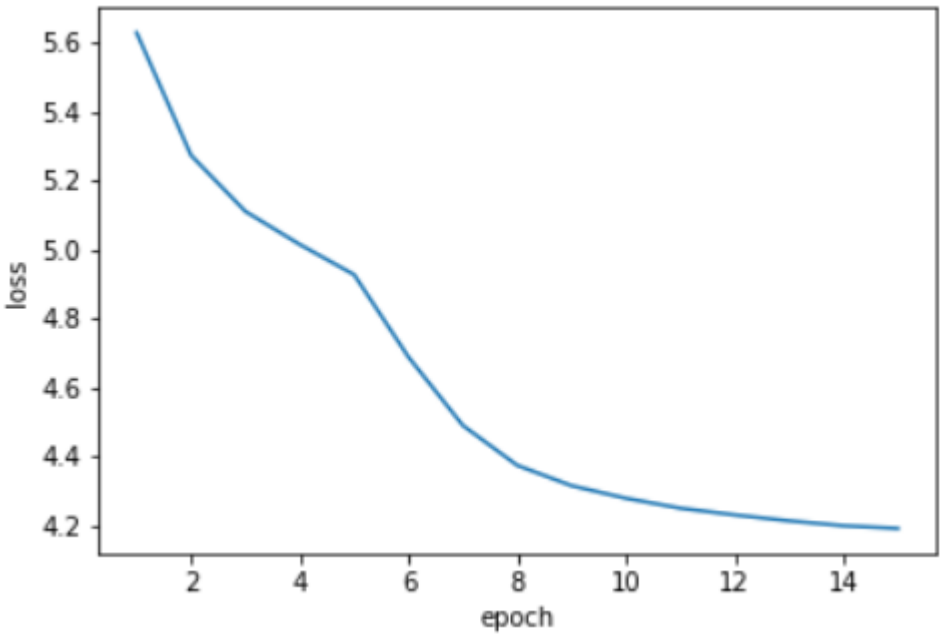
$$g_t = \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta * \mu * m_{t-1})$$

$$m_t = \mu * m_{t-1} + g_t$$

$$\Delta\theta_t = -\eta * m_t$$

采用Adam优化的结果在requirement2中已呈现，其中除learning rate设置为0.001以外其余参数均设置为默认值。

采用Nesterov优化时，我们需要额外设置momentum与dampening，为了体现momentum对优化的影响，在实验中将其设置为较高的数值0.9。loss下降的趋势如下图所示：

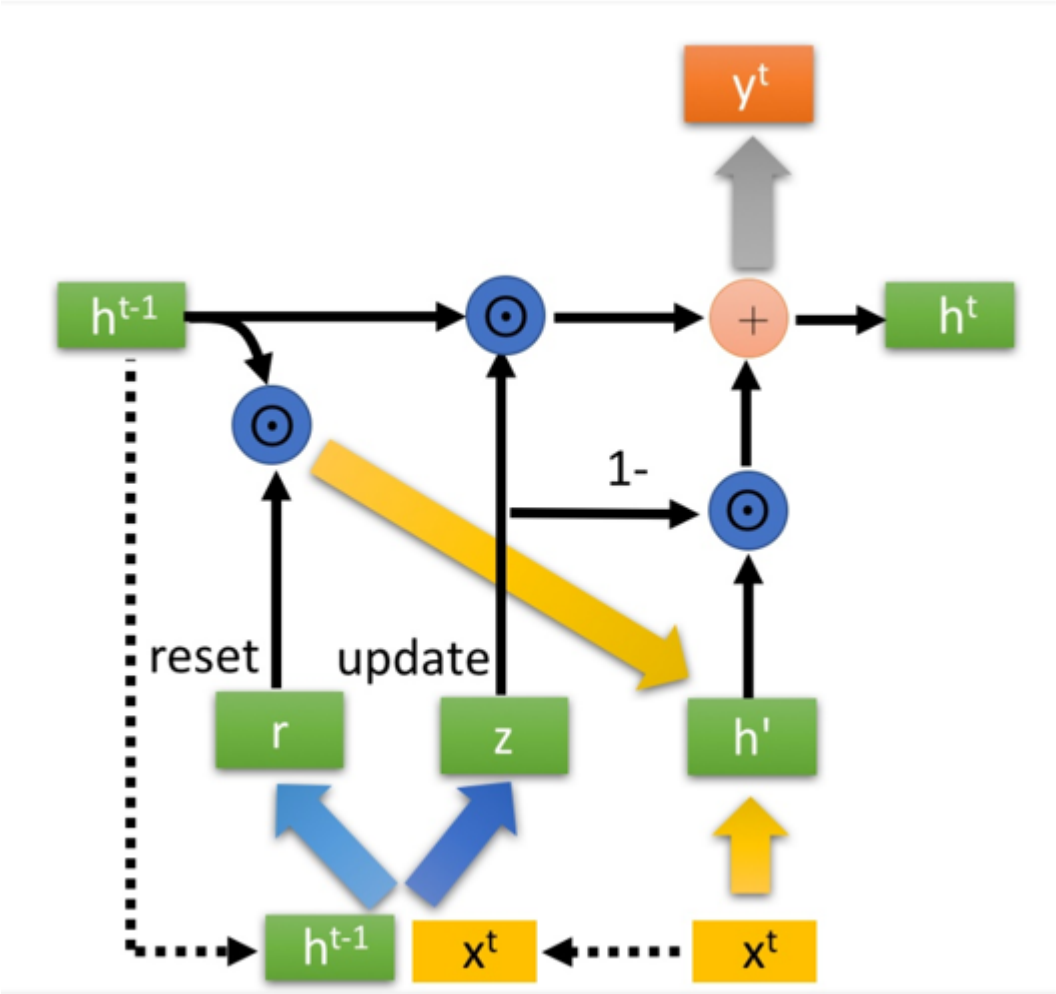


可以看到，loss相较adam版本有不小差距，在15个epoch后才到4左右；且非常突出的一点是loss的一阶导数并不是单调的，这和momentum有关。在学习率较小的时候，适当的momentum能够起到一个加速收敛速度的作用；在在学习率较大的时候，适当的momentum能够起到一个减小收敛时震荡幅度的作用。但实际上的训练中，momentum不会设置到0.9如此夸张的数值：momentum较大时，原本能够正确收敛的情况却不一定能"刹住车"。这时decay的重要性就显示出来，不过在此不做展开。

在Numpy版本，为了简便，采取更为熟悉的MBGD的优化方式，相较Adam等方法更为朴素，不过对小数据集也有不错的效果。

Part3： Exploration of GRU

GRU是2014年提出的模型，比LSTM少了一个门控，因其更高的运算效率与更少的参数在很多情况下成为LSTM的替代品。其内部结构如下图所示：

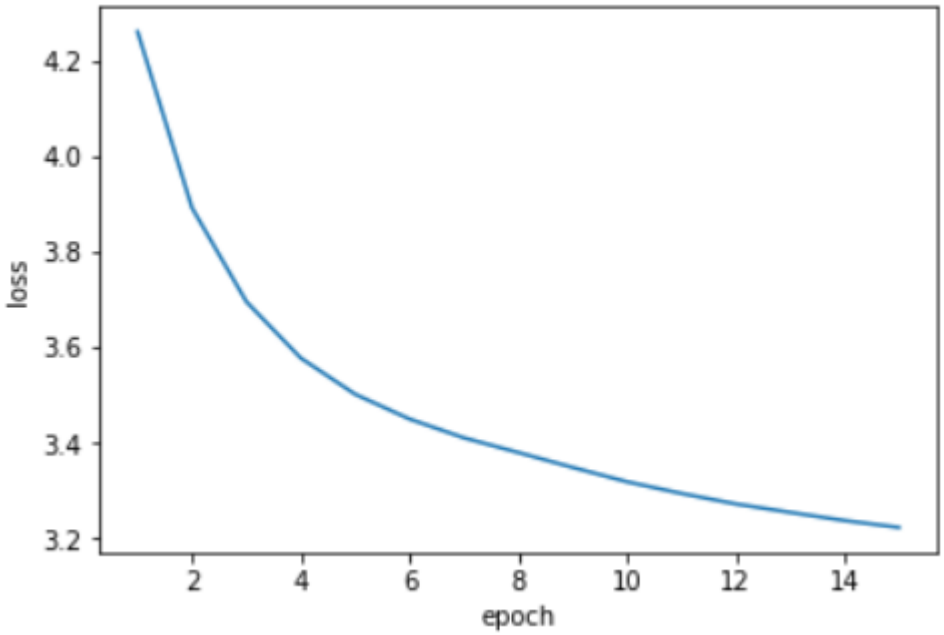


下给出其内部公式计算：

$$
\begin{aligned}
X &= [x^t, h^{t-1}] \\
r &= sigmoid(W^r X + b^r) \\
z &= sigmoid(W^z X + b^z) \\
h^{t-1'} &= h^{t-1} \odot r \\
X' &= [x^t, h^{t-1'}] \\
h' &= tanh(W^h X' + b^h) \\
h^t &= z \odot h^{t-1} + (1 - z) \odot h'
\end{aligned}
$$

GRU与LSTM最大的不同之处在于其h^t的计算中既包括了遗忘信息，又包括了记忆信息，z与1-z使得遗忘与记忆达到一种近似平衡的效果。在model.py文件内的GRUPoem类实现了手写GRU的内容，主要的注意点和LSTM还是相似的。

使用GRU的loss曲线如下所示：



从对测试集的perplexity也可以看到，其训练结果确实与LSTM相差无几：

```
Epoch: 14 batch index:710 loss: 3.228539 perplexity:25.24276161
Epoch: 14 batch index:711 loss: 3.723380 perplexity:41.40409851
Epoch: 14 batch index:712 loss: 3.196178 perplexity:24.43894005
Epoch: 14 batch index:713 loss: 3.542916 perplexity:34.56756592
Epoch: 14 batch index:714 loss: 3.216498 perplexity:24.94063377
Epoch: 14 batch index:715 loss: 3.159600 perplexity:23.56117630
Epoch: 14 batch index:716 loss: 3.320469 perplexity:27.67333794
Epoch: 14 batch index:717 loss: 3.385342 perplexity:29.52807808
Epoch: 14 batch index:718 loss: 2.767691 perplexity:15.92182636
Epoch: 14 batch index:719 loss: 3.052448 perplexity:21.16709328
```

GRU模型也能产生完全不逊于lstm模型的诗歌结果：

月：月上楼边望，江山万重春。风月正摇曳，花中开月明。

雪：雪压云山下草生，无限人生尽几年。春风满衣香白露，新月满窗间不见。

说明了GRU模型的正确性。在效率上，同时开始的GRU与LSTM，当GRU运行完15个epoch时，LSTM只到13个epoch，确实能够在一定程度上体现效率差距。

Clarification

1. 由于数据较大，在提交版本中已将其ignore。如需检查运行情况，可直接将https://github.com/chinese-poetry/chinese-poetry/tree/master/json文件夹放到与main.py同级的位置。
2. 文件结构：

   prepareData.py：数据预处理

   model.py：LSTM与GRU的pytorch模型

   GRUPoem.py：GRU版本训练

   numpyLSTM.py：numpy版本的LSTM模型及训练

source.py：LSTM-pytorch训练

langconv.py&zh_wiki.py：提供繁体转简体的支持

3. 模型在本人实验室的gpu上训练，若需验证cpu版本需要将代码中的cuda转为cpu运行。
4. 代码若有部分令人费解的顺序，可能原因是从ipynb转py文件产生的，已尽量避免此问题并为代码块功能加以注释。
5. 报告中部分图片来源于网络。