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

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

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

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

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

$$\frac{\partial h_t}{\partial i_t} = \frac{\partial h_t}{\partial C_t}\frac{\partial C_t}{\partial i_t} = \frac{\partial h_t}{\partial C_t} * \overline{C}_t = o_t * \left(1 - tanh^2(C_t)\right) * \overline{C}_t \tag{5}$$

$$\frac{\partial h_t}{\partial \overline{C}_t} = \frac{\partial h_t}{\partial C_t}\frac{\partial C_t}{\partial \overline{C}_t} = \frac{\partial h_t}{\partial C_t} * i_t = o_t * \left(1 - tanh^2(C_t)\right) * i_t \tag{6}$$

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

$$+\boldsymbol{W_C^T}\cdot\left[(\mathbf{1}-\bar{\boldsymbol{C}}_t)^2*\boldsymbol{i_t}*\boldsymbol{o_t}*\left(\mathbf{1}-\boldsymbol{tanh^2}(\boldsymbol{C_t})\right)\right]$$

$$+\boldsymbol{W_i^T}\cdot\left[\boldsymbol{i_t}*(\mathbf{1}-\boldsymbol{i_t})*\bar{\boldsymbol{C}}_t*\boldsymbol{o_t}*\left(\mathbf{1}-\boldsymbol{tanh^2}(\boldsymbol{C_t})\right)\right] \quad (7)$$

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

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

$$\frac{\partial \boldsymbol{h_t}}{\partial \boldsymbol{W_C}}=\frac{\partial \boldsymbol{h_t}}{\partial \bar{\boldsymbol{C}}_t}\frac{\partial \bar{\boldsymbol{C}}_t}{\partial \boldsymbol{W_C}}=\left[\frac{\partial \boldsymbol{h_t}}{\partial \bar{\boldsymbol{C}}_t}*(\mathbf{1}-\bar{\boldsymbol{C}}_t^{\,2})\right]\cdot\boldsymbol{z^T}=\left[\boldsymbol{o_t}*\left(\mathbf{1}-\boldsymbol{tanh^2}(\boldsymbol{C_t})\right)*\boldsymbol{i_t}*(\mathbf{1}-\bar{\boldsymbol{C}}_t^{\,2})\right]\cdot\boldsymbol{z^T}\ (10)$$

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

$$\frac{\partial \boldsymbol{h_t}}{\partial \boldsymbol{b_f}}=\frac{\partial \boldsymbol{h_t}}{\partial \boldsymbol{f_t}}\frac{\partial \boldsymbol{f_t}}{\partial \boldsymbol{b_f}}=\frac{\partial \boldsymbol{h_t}}{\partial \boldsymbol{f_t}}*\boldsymbol{f_t}*(\mathbf{1}-\boldsymbol{f_t})=\boldsymbol{o_t}*\left(\mathbf{1}-\boldsymbol{tanh^2}(\boldsymbol{C_t})\right)*\boldsymbol{C_{t-1}}*\boldsymbol{f_t}*(\mathbf{1}-\boldsymbol{f_t}) \quad (12)$$

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

$$\frac{\partial \boldsymbol{h_t}}{\partial \boldsymbol{b_C}}=\frac{\partial \boldsymbol{h_t}}{\partial \bar{\boldsymbol{C}}_t}\frac{\partial \bar{\boldsymbol{C}}_t}{\partial \boldsymbol{b_C}}=\frac{\partial \boldsymbol{h_t}}{\partial \bar{\boldsymbol{C}}_t}*(\mathbf{1}-\bar{\boldsymbol{C}}_t^{\,2})=\boldsymbol{o_t}*\left(\mathbf{1}-\boldsymbol{tanh^2}(\boldsymbol{C_t})\right)*\boldsymbol{i_t}*(\mathbf{1}-\bar{\boldsymbol{C}}_t^{\,2}) \quad (14)$$

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

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：

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

$$\mathbf{L}=\boldsymbol{L_1} \quad (17)$$

其中

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

对于h_t来说，它只对 t 时刻及以后的 loss 造成影响，因此有：

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

在反向传播时，由（7）和（19）可以得到每一个$\frac{\partial L}{\partial h_t}$，而（8）又已经得到$\frac{\partial h_t}{\partial W_f}$，由此可以计算$\frac{\partial L}{\partial W_f}$，如下所示：

$$\frac{\partial L}{\partial W_f}=\sum_{t=1}^{T}\frac{\partial L}{\partial h_t}\frac{\partial h_t}{\partial W_f} \tag{20}$$

其余参数的梯度类似可得。

二、part 2

1. 参数初始化

1.1 为什么不能将参数初始化为 0

考虑一个极端情况——输入参数每个维度的值相同。如果将参数全部初始化为 0，则在第一次迭代时，所有参数的梯度都会相等，从而在更新后每个参数的值也相等。这又会导致下一次迭代产生相同的梯度，因而无论迭代多少次，每个参数的值都保持一致，这显然不是我们想要的。

经过实验发现，即使输入参数每个维度的值不同，将参数初始化为 0 依然会导致参数的梯度非常接近，并且很小。下图反映了参数的变化情况：

```
tensor([[ 0.0025, -0.0023, -0.0023,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.0022, -0.0020, -0.0020,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.0023, -0.0020, -0.0020,  ...,  0.0000,  0.0000,  0.0000],
```

```
tensor([[ 0.0092, -0.0089, -0.0089,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.0091, -0.0089, -0.0089,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.0091, -0.0089, -0.0089,  ...,  0.0000,  0.0000,  0.0000],
```

1.2 如何初始化参数

有许多初始化参数的方法，例如：

- Xavier initialization：使得每一层输出的方差尽量相等
- 均匀分布
- 正态分布

通过查看源码可以知道，pytorch 在初始化 nn.embedding 层的参数时，采用正态分布（normal），而初始化 LSTM 或者线性层的参数时，采用均匀分布（uniform）。

2. 生成唐诗

2.1 数据集

本次实验的数据集来自《深度学习框架 pytorh：入门与实践》一书，作者对来自 https://github.com/chinese-poetry/chinese-poetry 的大约 5 万首诗歌进行了繁转简等操作。我对此进行预处理后生成了 poetry.csv，以便使用 **fastnlp** 进行加载。

在构建训练集等时，需要控制每个 sentence 长度一致，以便以 batch 的方式进行训练。这就需要进行截断或补全，具体做法如下图所示：

```
<pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad>
<pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad>
<pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><START>一
宿金山寺，超然离世羣。僧归夜船月，龙出晓堂云。树色中流见，钟声两岸闻。翻思在朝市，终日醉醺醺。<EOS>
```

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 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 earlystop 中允许评估指标 metric 连续更差的次数。

- **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，服裘物候佩银鞍。一阳宫殿门堪指，莫感娄朱窜羸马。圣壽一为十五年，五言徒振世难玉。仙人圆下似柳垂，十旬岁尽死江湄。

红：红鲤毫鬟管，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，水土塞虫萦。

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，飞妓一丸篆。怪宝论心柁，侬侬礼选多。故人多志士，世道欲何为？到开缄旂结，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，不羡幽人境。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。飞溜真蕲笔，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。骞如垂萝景，喷蔓拂衣裳。受恩似出儒，属孝徙周余。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。

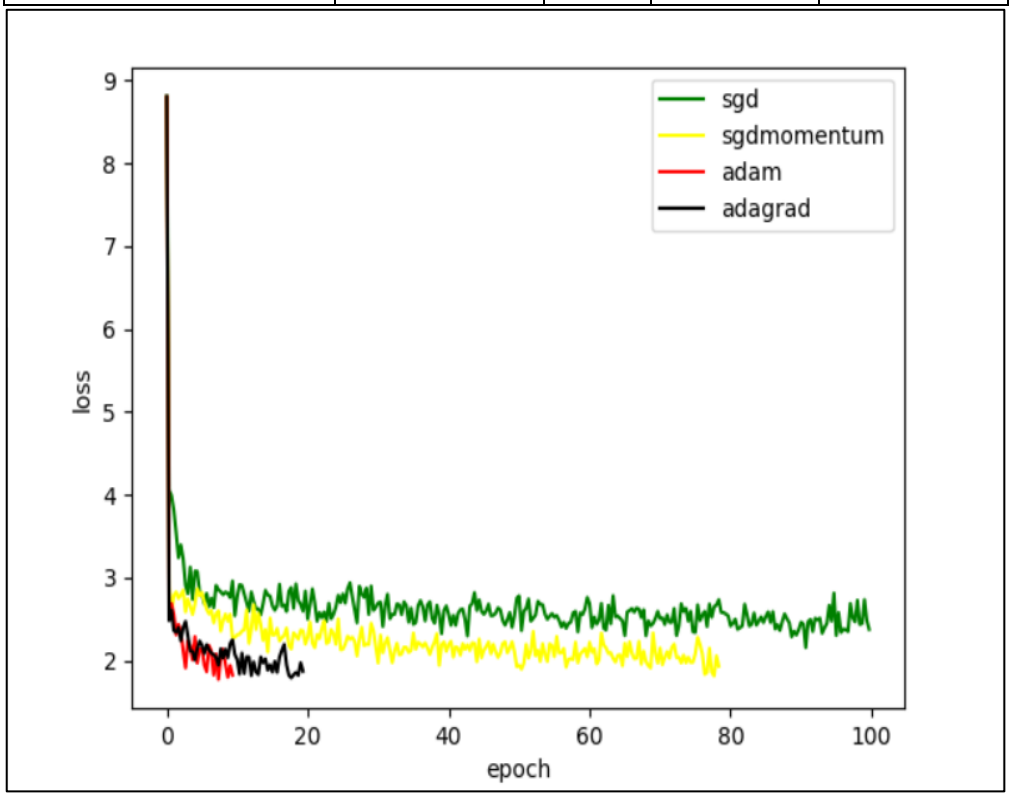
3. 优化器

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。