

实验内容：

1.numpy实现LSTM前项传播和反向求导的过程。

2.numpy实现SGD, Momentum, Nesterov, Adagrad, Adadelta, RMSProp, Adamax, Nadam, AdaBound优化器。

part 1:

首先计算各个变量单步的梯度：

$$\begin{aligned}\frac{\partial h_t}{\partial o_t} &= \tanh(c_t) * h_t \\ \frac{\partial h_t}{\partial c_t} &= o_t * (1 - \tanh(c_t) * \tanh(c_t)) * h_t + c_t \\ \frac{\partial h_t}{\partial f_t} &= c_{t-1} * \frac{\partial h_t}{\partial c_t} \\ \frac{\partial h_t}{\partial i_t} &= \tilde{c}_t * \frac{\partial h_t}{\partial c_t} \\ \frac{\partial h_t}{\partial c_{t-1}} &= f_t * \frac{\partial h_t}{\partial c_t} \\ \frac{\partial h_t}{\partial \tilde{c}_t} &= i_t * \frac{\partial h_t}{\partial c_t}\end{aligned}$$

$$\begin{aligned}\frac{\partial h_t}{\partial w_i} &= z^T \cdot ((1 - i_t) * i_t * \frac{\partial h_t}{\partial i_t}) \\ \frac{\partial h_t}{\partial w_f} &= z^T \cdot ((1 - f_t) * f_t * \frac{\partial h_t}{\partial f_t}) \\ \frac{\partial h_t}{\partial w_o} &= z^T \cdot ((1 - o_t) * o_t * \frac{\partial h_t}{\partial o_t}) \\ \frac{\partial h_t}{\partial w_{\tilde{c}}} &= z^T \cdot ((1 - \tilde{c}_t * \tilde{c}_t) * \frac{\partial h_t}{\partial \tilde{c}_t})\end{aligned}$$

$$\begin{aligned}\frac{\partial h_t}{\partial b_i} &= \sum ((1 - i_t) * i_t * \frac{\partial h_t}{\partial i_t}) \\ \frac{\partial h_t}{\partial b_f} &= \sum ((1 - f_t) * f_t * \frac{\partial h_t}{\partial f_t}) \\ \frac{\partial h_t}{\partial b_o} &= \sum ((1 - o_t) * o_t * \frac{\partial h_t}{\partial o_t}) \\ \frac{\partial h_t}{\partial w_{\tilde{c}}} &= \sum ((1 - \tilde{c}_t * \tilde{c}_t) * \frac{\partial h_t}{\partial \tilde{c}_t})\end{aligned}$$

$$\frac{\partial h_t}{\partial a_t} = [(1 - i_t) * i_t * \frac{\partial h_t}{\partial i_t}, (1 - f_t) * f_t * \frac{\partial h_t}{\partial f_t}, (1 - o_t) * o_t * \frac{\partial h_t}{\partial o_t}, (1 - g_t * g_t) * \frac{\partial h_t}{\partial \tilde{c}_t}]$$

$$w_i = \begin{bmatrix} w_{i_h} \\ w_{i_x} \end{bmatrix}, w_f = \begin{bmatrix} w_{f_h} \\ w_{f_x} \end{bmatrix}, w_o = \begin{bmatrix} w_{o_h} \\ w_{o_x} \end{bmatrix}, w_g = \begin{bmatrix} w_{g_h} \\ w_{g_x} \end{bmatrix}$$

$$w_x = [w_{i_x}, w_{f_x}, w_{o_x}, w_{g_x}]$$

$$w_n = [w_{i_h}, w_{f_h}, w_{o_h}, w_{g_h}]$$

$$\frac{\partial h_t}{\partial x_t} = \frac{\partial h_t}{\partial a_t} \cdot w_x^T$$

$$\frac{\partial h_t}{\partial h_{t-1}} = w_h^T$$

实现代码如下：

```
def step_backward(self, dnext_h, dnext_c, cache):
    i_t, f_t, o_t, g_t, next_c, next_h, x, prev_h, prev_c, wx, wh, b = cache

    do_t = np.tanh(next_c) * dnext_h
    dc_t = o_t * (1 - np.tanh(next_c) * np.tanh(next_c)) * dnext_h + dnext_c
    df_t = prev_c * dc_t
    di_t = g_t * dc_t
    dprev_c = f_t * dc_t
    dg_t = i_t * dc_t

    dai = (1 - i_t) * i_t * di_t
    daf = (1 - f_t) * f_t * df_t
    dao = (1 - o_t) * o_t * do_t
    dag = (1 - g_t * g_t) * dg_t
    da = np.hstack((dai, daf, dao, dag))

    dwx = np.dot(x.T, da)
    dwh = np.dot(prev_h.T, da)
    db = np.sum(da, axis=0)
    dx = np.dot(da, wx.T)
    dprev_h = np.dot(da, wh.T)

    return dx, dprev_h, dprev_c, dwx, dwh, db
```

整个序列的反向求导过程代码如下：

```
def backward(self, dh):
    N, T, H = dh.shape
    i_t, f_t, o_t, g_t, next_c, next_h, x, prev_h, prev_c, wx, wh, b = self.cache[T-1]

    D = x.shape[1]

    dprev_h = np.zeros((N, H))
    dprev_c = np.zeros((N, H))
    dx = np.zeros((N, T, D))
    dh0 = np.zeros((N, H))
```

```

dwx = np.zeros((D, 4*H))
dwh = np.zeros((H, 4*H))
db = np.zeros((4*H, ))

for i in range(T-1, -1, -1):
    step_cache = self.cache[i]
    dnext_h = dh[:, i, :] + dprev_h
    dnext_c = dprev_c
    dx[:, i, :], dprev_h, dprev_c, dwx_t, dwh_t, db_t = \
        self.step_backward(dnext_h, dnext_c, step_cache)
    dwx += dwx_t
    dwh += dwh_t
    db += db_t

dh0 = dprev_h

return dx, dh0, dwx, dwh, db

```

part 2:

模型的超参如下:

vocabulary size: 2508

batch size: 32

sentence length: 128

hidden size: 128

input size: 128

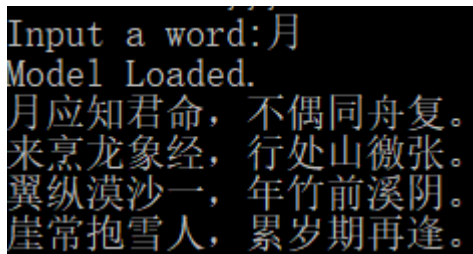
详见config.json文件

实验报告:

1.如果将模型参数全部初始化为零，后续运算的乘法操作将无法产生有效的梯度（始终为0），使得无法通过反向求导对模型进行优化，但是对于偏置项，可以初始化为0。embedding层如果全部初始化为0，会出现词的特征之间没有区分度，模型后续的优化无法正常实现。

一般常见的做法使用指定参数的正态来对参数进行初始化，不同类型的层所指定的正态分布参数不同，如卷积层和全连接层一般指定均值为0，方差为0.02，如果带有Batch Normalize的话指定均值为1，方差为0.02。

2.生成的唐诗如下:



```

Input a word: 月
Model Loaded.
月应知君命，不偶同舟复。
来烹龙象经，行处山微张。
翼纵漠沙一，年竹前溪阴。
崖常抱雪人，累岁期再逢。

```

```
Input a word:红
Model Loaded.
红害罢骢吟，到泉壑仙舟。
君为朝落孤，英尤见皇之。
蟠未遑摇泽，深花疏旗掣。
有细腰青嶂，但愿长安本。
```

```
Input a word:山
Model Loaded.
山愚外去遥，闻牛惟我理。
花珍暖香筇，载揖龙舍近。
暮但飞萧萧，我材到泉绕。
津低离楚襄，宅人累累谷。
```

```
Input a word:夜
Model Loaded.
夜雪我未机，者田心茫负。
海亭高堂方，自天可涉抚。
头插二外丸，千岁一举竹。
昼里无隔新，幄青嶂见昨。
```

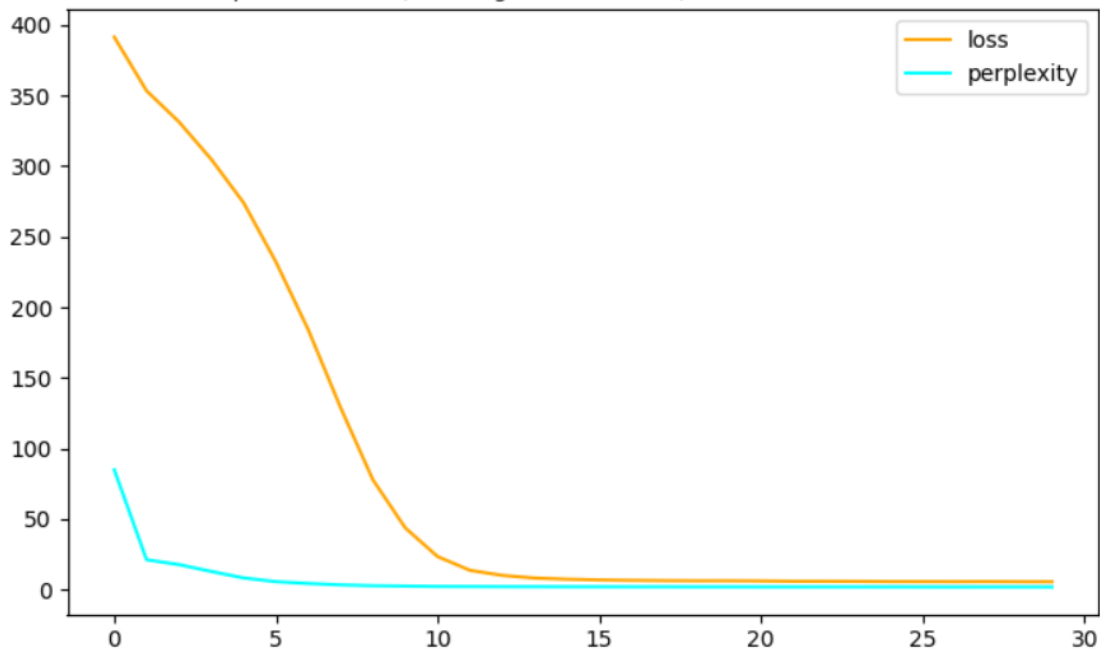
```
Input a word:湖
Model Loaded.
湖载揖海中，迷邦荆负楚。
魂千岩琴歌，林栖雨劝主。
说主缚湿望，夫雪复来圣。
罢病濡连我，观鱼传无人。
```

```
Input a word:海
Model Loaded.
海笙惟根去，承又同耕者。
敌如园荆负，鸣千刀日酹。
听白生何施，君命之育搏。
囚风龙旂翻，有细慢谷未。
```

由于用numpy实现的优化器仅仅依照优化算法的原始公式，而pytorch中的实现的优化器做了进一步的优化，如加入weight_decay这一正则项，所以梯度不会完全一致。但是，对于提供的tangshi.txt中的数据，模型的training loss可以降低到很小的值，过拟合训练集：

loss and perplexity curves in training procedure

optimizer: Adam, learning rate: 0.010000, test loss: 1412.291565



```
The epoch : 30
The epoch costs time: 57.08 s
The training loss is: 5.609158
The perplexity is: 1.931278
Model Saved.
```

3.实现了SGD, Momentum, Nesterov, Adagrad, Adadelata, RMSProp, Adamax, Nadam, AdaBound共9个优化器, 详见optim.py文件。

除教材中提到的6个优化器之外, 额外实现了3个:

1. Adamax是Adam的一种变体, 学习率的范围更加简单。Adamax的公式如下:

$$\begin{aligned} g_t &= \nabla_{\theta_{t-1}} f(\theta_{t-1}) \\ m &= \frac{\beta_1 * m + (1 - \beta_1) * g_t}{1 - \beta^t} \\ v &= \max(\beta_2 * v, |g_t|) \\ \Delta\theta &= -\frac{\alpha * m}{v + \epsilon} \end{aligned}$$

2. Nadam相当于一个带有Nesterov动量项的Adam。Nadam的公式如下:

$$\begin{aligned}
g_t &= \nabla_{\theta_{t-1}} f(\theta_{t-1}) \\
\hat{g}_t &= \frac{g_t}{1 - \prod_{i=1}^t \beta_{1_i}} \\
m_t &= \beta_{1_t} * m_{t-1} + (1 - \beta_{1_t}) * g_t \\
\hat{m}_t &= \frac{m_t}{1 - \prod_{i=1}^{t+1} \beta_{1_i}} \\
n_t &= \beta_2 * n_{t-1} + (1 - \beta_2) * g_t^2 \\
\hat{n}_t &= \frac{n_t}{1 - \beta_2^t} \\
\bar{m}_t &= (1 - \beta_{1_t}) \hat{g}_t + \beta_{1_{t+1}} \hat{m}_t \\
\Delta \theta_t &= -\frac{\alpha \bar{m}_t}{\sqrt{\hat{n}_t} + \epsilon}
\end{aligned}$$

Nadam对学习率有了更强的约束，同时对梯度的更新也有更直接的影响，一般情况下载使用RMSProp或者Adam的地方，使用Nadam效果会更好。

3. AdaBound是今年ICLR上新提出的一个优化算法，号称有Adam的速度和SGD的性能。公式如下：

$$\begin{aligned}
g_t &= \nabla f_t(\theta_t) \\
m_t &= \beta_{1_t} m_{t-1} + (1 - \beta_{1_t}) g_t \\
v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\
V_t &= \text{diag}(v_t) \\
\hat{\eta}_t &= \text{Clip}\left(\frac{\alpha}{\sqrt{V_t}}, \eta_l(t), \eta_u(t)\right) \\
\eta_t &= \frac{\hat{\eta}_t}{\sqrt{t}} \\
\Delta \theta_t &= \eta_t * m_t
\end{aligned}$$

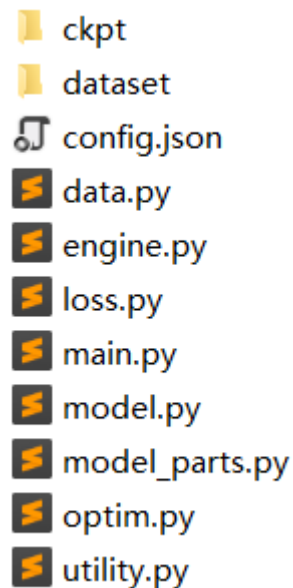
where η is the learning rate, and η_l/η_u is the lower/upper bound of learning rate.

其余详见教材。

这次实验中由于问题本身的复杂性有限，除了SGD和Momentum的优化速度明显慢于Adam类优化器之外，其余基本无明显的区别。

补充材料：

实验的文件目录如下：



1. ckpt中保存有训练好的模型，格式为.npz，使用numpy的savez_compressed函数保存，详见engine.py中的save_model函数。
2. dataset中有两个数据集，一个是tang_mini.npz，为课程github中提供的数据进行预处理后的压缩文件，另一个是tang_big.npz，来自于全唐诗，和上一个数据做了同样的处理，不同之处在于没有去掉标点符号，默认通过模型来生成标点符号，而不是手动添加，且数据规模较大。
3. config.json中用来设置和记录模型的参数。
4. data.py中为预处理文本数据和获取训练使用数据的函数。
5. engine.py中搭建了整个流程的框架，包括：
 - 初始化模型和优化器
 - 获取训练/验证/测试数据
 - 保存模型
 - 加载模型
 - 训练和测试
 - 计算困惑度
 - 生成诗词
6. loss.py中为计算soft max (temperature term) 的loss函数。
7. main.py为主程序，根据config.json中的参数来决定训练/继续训练模型或测试模型、生成诗句。
8. model.py中定义了用来诗词生成的模型，仿照pytorch的风格编写，考虑到遍历，一些细节与pytorch的风格略有不同。
9. model_parts.py中定义了主模型用到的小模型，包括：
 - Embedding
 - LSTM
 - 序列全连接模型
 - 全连接模型
10. optim.py中用numpy实现了用到的优化器，包括：
 - SGD
 - Adam
 - Momentum
 - Nesterov
 - Adagrad

- Adadelta
- RMSProp
- Adamax
- Nadam
- AdaBound

11. utility.py中定义了用到的一些工具函数。

整个实验中没有使用pytorch，全部使用numpy完成，所以对于规模较大的数据，训练会非常慢，所以建议使用tang_mini.npz作为训练数据。