PRML Assignment 3 Report

Part 1：Differentiate LSTM

(1) Differentiate one step of LSTM with respect to h_t:

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

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

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

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

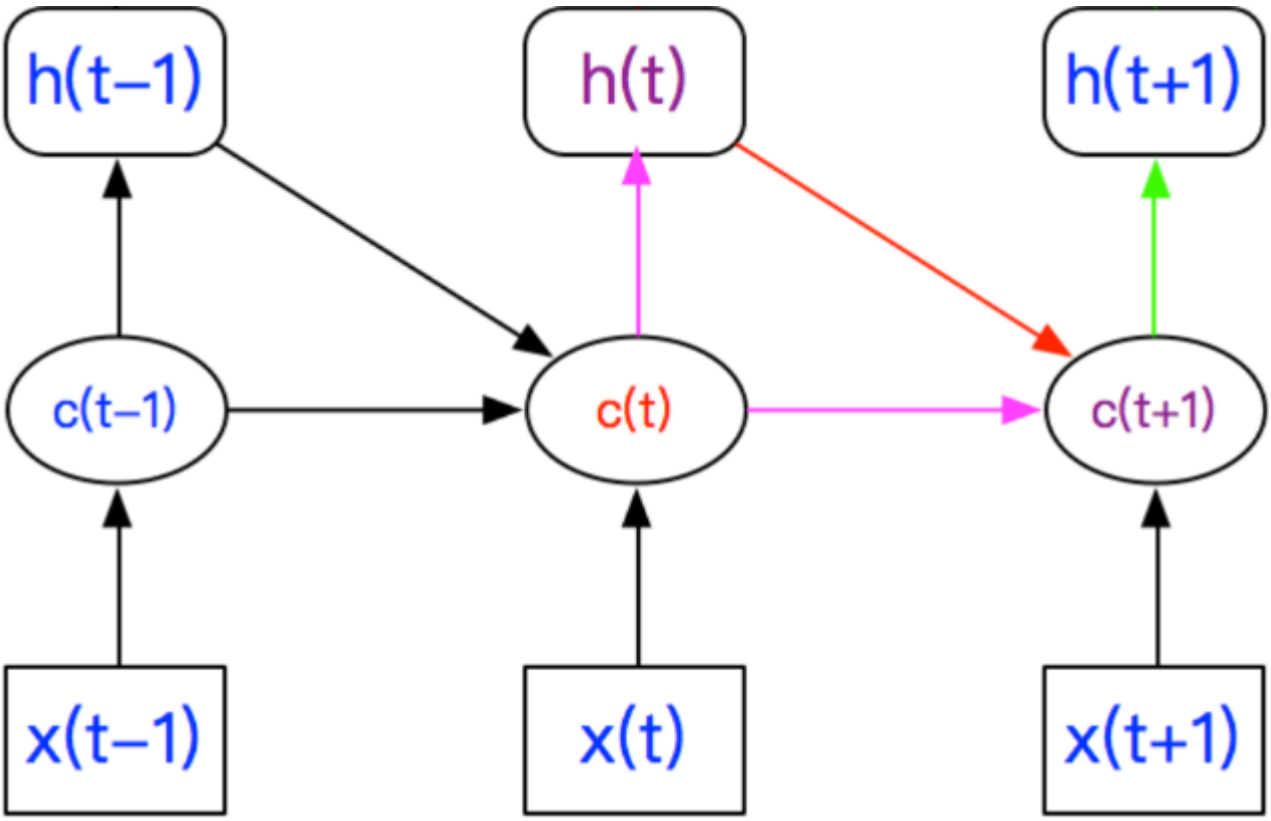
  $= o_t * \frac{\partial tanh(\sigma(W_f \cdot z + b_f) * C_{t-1} + i_t * \bar{C}_t)}{\partial b_f} = o_t * C_{t-1} * f_t * (1 - f_t) * [1 - tanh^2(C_t)]$
- $\frac{\partial h_t}{\partial b_i} = o_t * \frac{\partial tanh(C_t)}{\partial b_i} = o_t * \frac{\partial tanh(f_t * C_{t-1} + i_t * \bar{C}_t)}{\partial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

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

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

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

(2) Back Propagation Through Time:



与CNN网络中的BP算法不同的是，RNN（LSTM）网络中采用BPTT即基于时间的反向传播算法来进行神经网络的反向传播，原因是LSTM处理的是序列数据，并且在BP的过程中需要将整个时间序列上的误差传播回来。

简单地呈现LSTM的结构如上图，可以发现，当前cell的状态会受到前一个cell状态的影响，因此在误差反向传播的过程中，可以返现h(t)的误差不仅仅包含了当前时刻t的误差，也包含了t时刻之后所有时刻的误差。这一点就是BPTT与普通BP算法最大的区别，即每一步的误差都应该利用之后所有步骤误差的总和来表示，这样每个时刻的误差都可以经由h(t)和c(t+1)迭代计算。

以输入序列s_1,s_2,.....,s_n为例，输入序列即对应着输入参数x(t)。首先，经过前向传播到每一层，就能够得到对应的由h(t)构成的预测序列；接下来对误差进行反向传播，不妨设t+1时刻即对应于s_{t+1}与输入x(t+1)的误差为p，那么当传播到x(t)时，需要经由h(t)和c(t+1)迭代计算，并将p包含到该时刻的误差中。以此方式不断迭代，则每一时刻的误差都为之后所有时刻的误差总和，从而体现出LSTM结构在时间层面上的依赖性，通过以上的方式即可以完成对于序列s_1,s_2,.....,s_n的BPTT反向传播。

Part 2：Autograd Training of LSTM

（1）Initialization：

在神经网络的训练过程中，参数学习是基于梯度下降法进行优化的，而梯度下降法在开始训练时需要给每个参数赋一个初始值，而这个初始值的选取十分关键。

Q1：why should not just initialize them to zero?

对于**全零初始化**：如果将梯度下降的所有参数都初始化为0，通过网络的训练，我们会发现全零初始化方法在前向传播过程中会使得神经元的激活值均为0，从而在反向传播过程中，根据公式不同维度的参数会得到相同的更新。因此，不管进行多少轮的正向传播与反向传播，每一层中的每一个节点学习到的东西都是相同的。而神经网络希望不同节点学习到不同的参数，并且学到不同的特征，显然全零初始化不能够达到网络训练的目的，会影响训练模型的效果或者导致模型无法收敛。

Q2：A way to initialize them properly

从Q1可以看到，全零初始化方法训练得到的结果带有对称性，而神经网络需要的就是破坏这一对称性，即使得不同的隐藏层神经元能够学习到不同的东西。因此有以下几类参数初始化的方式：随机初始化、标准初始化、Xavier初始化、He初始化，并且利用数据和参数的均值都为0，输入和输出数据的方差一致的标准来衡量初始化的优劣。

对于**随机初始化**，即将神经网络参数初始化为随机值，很容易得到网络输出数据分布的方差会随着输入神经元个数而发生改变。以Sigmoid激活函数为例，Sigmoid函数中间斜率大两侧小，且两侧近乎平行于X轴，如果随机初始化的参数值落在两边（即落在无梯度的范围内），则会导致梯度下降过程中梯度的变化很慢，从而减缓了模型训练的进程。

对于**标准初始化**，初始化方式为$standard\ uniform: W_{i,j} \sim U(-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}})$，通过对方差乘以一个系数确保每层神经元的输出具有相同的方差，提高训练收敛的速度。以上的方法保证了激活函数的输入值的均值为0，方差为常量1/3，和网络的层数和神经元的数量无关。对于Sigmoid激活函数，虽然能确保自变量处于有梯度的范围内，但由于其输出大于0，违反了衡量标准中均值为0的假设。因此，该初始化方法更适用于tanh激活函数。

对于**Xavier初始化**，该初始化要求神经元的权重的方差$Var(\omega) = \frac{2}{n_{in}+n_{out}}$，并且以如下方式进行初始化$xavier\ uniform: W_{i,j} \sim U(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}})$，但该方法有一定的限制，其推导过程假设激活函数在零点附近接近线性函数，且激活值关于0对称，而Sigmoid与Relu不满足这些假设。

对于**He初始化**，初始化方式为$he\ uniform: W_{i,j} \sim U(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}})$，该方法考虑了Relu对于输出数据分布的影响，使得输入与输出数据的方差一致，适用于Relu激活函数。

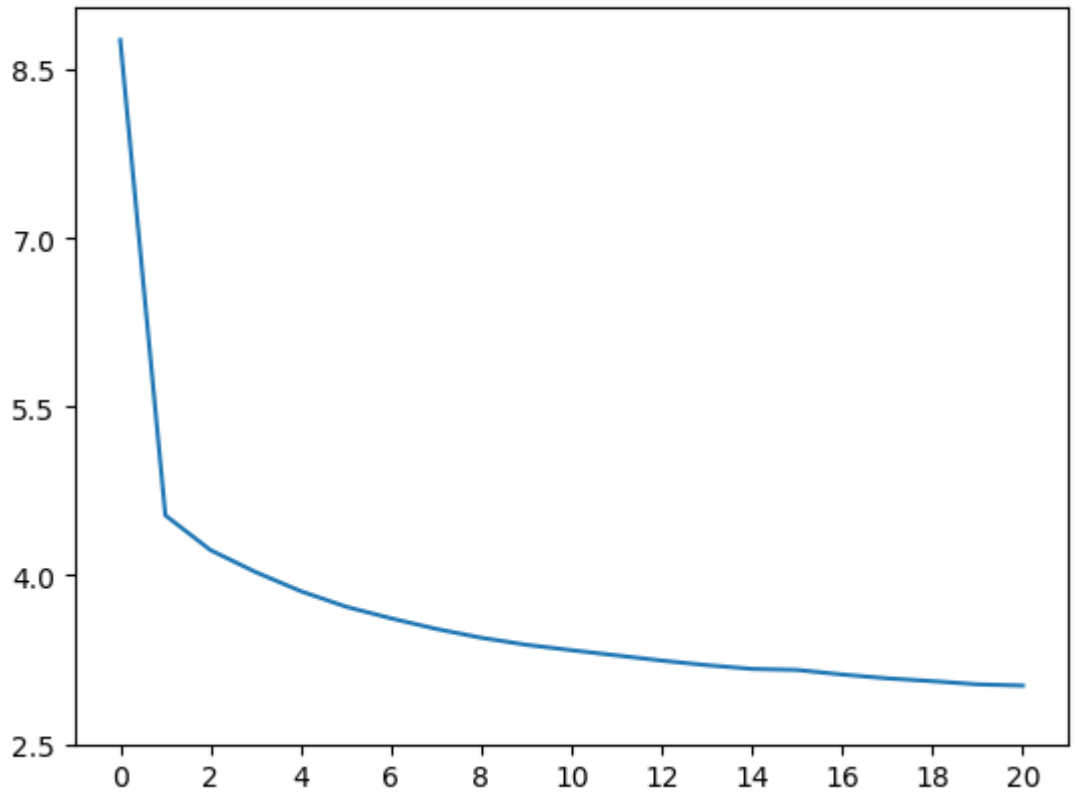
那么**针对于LSTM网络**，同时采用了Sigmoid与tanh作为激活函数，对于tanh函数即可以使用标准初始化来进行参数的初始化；而对于Sigmoid函数，我们需要做的就是确保自变量处于有梯度的范围（不落在相对平的两侧），因此可以利用随机初始化参数辅以乘以矫正因子1/sqrt(n)来校准方差，即通过torch.rand(x.dim,y.dim) * 1/sqrt(n : the number of input)来初始化，这样就保证了网络的所有神经元都储于有梯度的范围内，加快梯度下降过程中梯度的变化速度也即模型的收敛速度。

（2）Generating Tang Poem：

Q1：Perplexity

采用全唐诗作为LSTM模型训练集，并且利用Adam优化器，部分参数设置：learning rate=1e-3，embedding_dim=512，hidden_dim=512，batch_size = 128（具体见tang_torch/config.py）。 经过一定数量的Epoch的训练，**通过交叉熵计算得到的loss从8.76下降到2.92左右**，并趋于稳定，模型逐渐收敛，loss曲线如下图所示；并且通过输出可以看到，**在测试集上的困惑度Perplexity约为18.75**，近似为e^{loss}量级。

loss曲线：



Q2：Generate poems（数据集：全唐诗）

1、日：

日月照东海，此时谁与同。一身不可见，万事何由迁。

2、红：

红粒装成器，黄金络作囊。中年一半在，緼白一茎皮。

头白吠天竺，心胸自中细。一寸不可见，一生心不足。

3、山：

山河通高楼，山水连云屏。俯见禹迹巅，仰视天长地。

4、夜：

夜半秋风起，西风动吹笳。君王不可见，吹断不胜怀。

陇上秋阴满，河边战迹来。碛云犹起碛，沙碛正喧鼙。

5、湖：

湖上春光好，春来淑气多。蜜蜂裁网髻，莲子拂金梭。

拂地红莲叶，迎风竹叶黄。花时双脸薄，风暖五侯娇。

6、海：

海内昔年狎血消，忠臣不得死人间。自从万里无人识，何况青山有故乡。

青山之下饶烟雨，白马千年万里田。长安城北多春草，莫道西陵江水东。

7、月：

月出何门来，不见山中草。今年不见君，独有江南别。

Q3：The implement by using numpy

完全采用Numpy进行LSTM模型的实现，与Torch的方法差别在于反向传播的部分，需要自行编写模型训练中反向传播更新模型参数的过程（即根据Part1中求出的各参数梯度以及BPTT的反向传播方法进行实现），具体实现见**tang_numpy/model.py**。

（3）Optimization：

在机器学习中，一般通过采用优化方法来试图寻找模型的最优解，大致上分为梯度下降法、动量优化法、自适应学习率优化算法。

梯度下降法是最基本的一类优化器：

首先是**标准梯度下降法GD**，使用以下公式进行参数更新$W_{t+1} = W_t - \eta_t \Delta J(W_t)$，从表达式中看出，模型参数的更新调整与代价函数关于模型参数的梯度有关，即沿着梯度的方向不断更新模型参数，从而最小化代价函数。该方法训练速度较慢，而且容易陷入局部最优解即鞍点（导致模型参数不再更新）。

其次是**批量梯度下降法BGD**，简单来讲就是每次批量选取n个训练样本，根据这全部n个样本进行更新，则使用BGD表达式为：$W_{t+1} = W_t - \eta_t \Delta J_i(W_t, X^{(i)}, Y^{(i)})$，相比于GD而言每次权值的调整发生在批量样本输入之后，而不是输入一个样本就更新一次参数，大大加快了训练速度。

最后是**随机梯度下降法SGD**，对比于BGD，假设从一批训练样本n中随机选取一个样本i_s，同样利用以上的表达式进行参数更新，可以看到虽然引入了随机性和噪声，但期望仍等于正确的梯度下降。该方法每次从大型数据集中选取一定数量数据点，进行一次模型参数更新，相比于GD的每输入一个样本更新一次参数，收敛速度更快。

动量优化法是在梯度下降法的基础上进行改变，从而加速梯度下降：

使用动量的随机梯度下降**SGD with Momentum**，即引入一个积攒历史梯度信息动量来加速SGD，从训练集中取一个大小为n的样本，则采用了Momentum优化以后的表达式为：$W_{t+1} = W_t - v_t$

$v_t = \alpha v_{t-1} + \eta_t \Delta J(W_t, X^{(i_s)}, Y^{(i_s)})$，可以理解为当前权值的改变会受到上一次权值改变的影响，相当于带上了惯性而加快收敛速度。动量法主要解决随机梯度中引入噪声，以及SGD收敛过程中和正确梯度相比来回摆动较大的问题。

牛顿加速梯度**NAG**算法，是Momentum动量算法的变种，参数更新的表达式为：$W_{t+1} = W_t - v_t$

$v_t = \alpha v_{t-1} + \eta_t \Delta J(W_t - \alpha v_{t-1})$。Nesterov动量梯度的计算在模型参数施加当前速度之后，因此可以理解为向标准动量中添加了一个校正因子，从而可以提前直到下一个位置的梯度，然后使用当前位置来更新参数。但是NAG仅对于BGD有收敛率上的提升，对于SGD的作用不大。

自适应学习率优化算法针对于机器学习模型的学习率，采取策略更新学习率，提高训练速度：

AdaGrad算法，独立地适应所有模型参数的学习率，具有代价函数最大梯度的参数相应地有个快速下降的学习率，而具有小梯度的参数在学习率上有相对较小的下降。AdaGrad算法优化策略可以表示为：
$W_{t+1} = W_t - \frac{\eta_0}{\sqrt{\Sigma_{t'=1}^{t}(g_{t',i}) + \epsilon}}$，假定一个多分类问题，从表达式看出，对于出现比较多的类别数据，该算法会给予越来越小的学习率吧，而对于比较少的类别数据，会给予较大的学习率，因此适用于数据稀疏或者分布不平衡的数据集。AdaGrad的主要优势在于不需要人为的调节学习率，但随着迭代次数增多，学习率会越来越小，最终趋近于0。

AdaDelta算法，对于上一个算法需要指定全局学习率，而该算法利用前$t-1$次模型参数每次的更新步长为参照，进行学习率的更新，并且不需要设置一个默认的全局学习率。在模型训练的前期和中期，AdaDelta的表现很好，加速训练的效果好；而在模型训练的后期，模型会反复地在局部最小值附近抖动。

Adam算法，Adam中动量直接并入了梯度一阶矩的估计，并且Adam包含了偏置修正，修正从原点初始化的一阶矩和二阶矩估计，具体的实现策略可以表示为：$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{v_t}} + \epsilon}\hat{m}_t; m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t;$

$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2; \hat{m}_t = \frac{m_t}{1-\beta_1^t}; \hat{v}_t = \frac{v_t}{1-\beta_2^t}$。相比于其他的自适应算法，对于目标函数没有平稳要求，即loss_function可以到随着时间变化，并且能较好的处理噪音样本和稀疏梯度，总体来说Adam对于收敛速度有很好的加速作用。

通过对于该LSTM模型采用不同的优化器，可以发现采用Adam时（自定义初始化学习率1e-3），相对来说模型的收敛速度较快，并且模型效果较好，因此在本次Assignment中选择采用Adam算法进行优化。

附录：文件目录

1、tang_torch LSTM模型实现（Pytorch）

2、tang_numpy LSTM模型实现（Numpy）

3、report.pdf assignment3报告