Assignment 3 Report

Part I

1. 求导

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

$$\frac{\partial h_t}{\partial o_t} = \tanh(C_t) \tag{2}$$

$$\frac{\partial h_t}{\partial W_o} = \frac{\partial h_t}{\partial o_t} * \frac{\partial o_t}{\partial W_o} = \frac{\partial h_t}{\partial o_t} * o_t * (1 - o_t) \cdot z^T \tag{3}$$

$$\frac{\partial h_t}{\partial b_o} = \frac{\partial h_t}{\partial o_t} * \frac{\partial o_t}{\partial b_o} = \frac{\partial h_t}{\partial o_t} * o_t * (1 - o_t) \tag{4}$$

$$\frac{\partial h_t}{\partial f_t} = \frac{\partial h_t}{\partial C_t} * \frac{\partial C_t}{\partial f_t} = \frac{\partial h_t}{\partial C_t} * C_{t-1} \tag{5}$$

$$\frac{\partial h_t}{\partial i_t} = \frac{\partial h_t}{\partial C_t} * \frac{\partial C_t}{\partial i_t} = \frac{\partial h_t}{\partial C_t} * \bar{C}_t \tag{6}$$

其中$\frac{\partial h_t}{\partial W_i}$、$\frac{\partial h_t}{\partial b_i}$、$\frac{\partial h_t}{\partial W_f}$、$\frac{\partial h_t}{\partial b_f}$与(3)(4)两式类似求解，不再赘述

$$\frac{\partial h_t}{\partial C_{t-1}} = \frac{\partial h_t}{\partial C_t} * \frac{\partial C_t}{\partial C_{t-1}} = \frac{\partial h_t}{\partial C_t} * f_t \tag{7}$$

$$\frac{\partial h_t}{\partial \bar{C}_t} = \frac{\partial h_t}{\partial C_t} * \frac{\partial C_t}{\partial \bar{C}_t} = \frac{\partial h_t}{\partial C_t} * i_t \tag{8}$$

$$\frac{\partial h_t}{\partial W_C} = \frac{\partial h_t}{\partial \bar{C}_t} * \frac{\partial \bar{C}_t}{\partial W_C} = \frac{\partial h_t}{\partial \bar{C}_t} * \left(1 - \bar{C}_t^{\,2}\right) \cdot z^T \tag{9}$$

$$\frac{\partial h_t}{\partial b_C} = \frac{\partial h_t}{\partial \bar{C}_t} * \frac{\partial \bar{C}_t}{\partial b_C} = \frac{\partial h_t}{\partial \bar{C}_t} * \left(1 - \bar{C}_t^{\,2}\right) \tag{10}$$

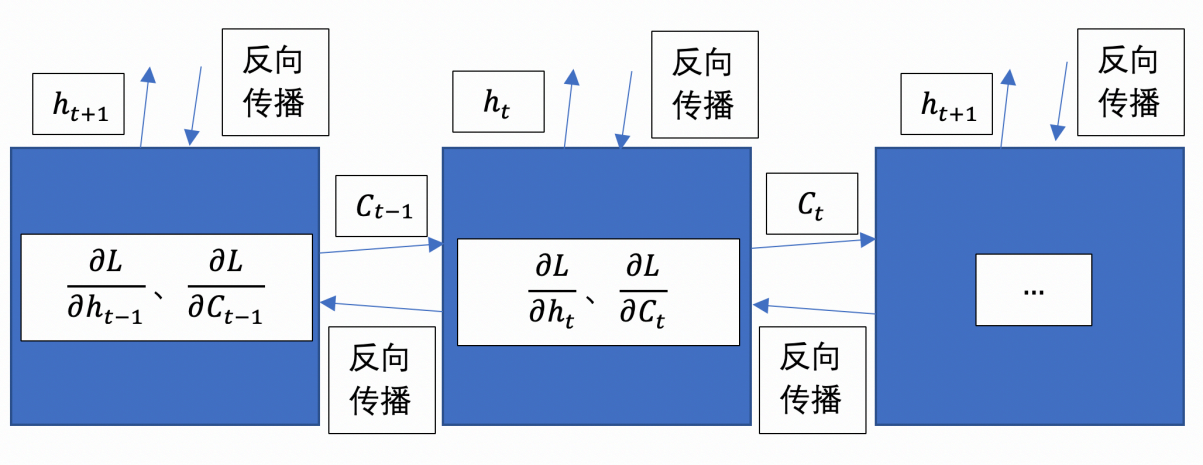
$$\begin{aligned}\left[\frac{\partial h_t}{\partial h_{t-1}}, \frac{\partial h_t}{\partial x_t}\right] = \frac{\partial h_t}{\partial z} = {W_f}^T \cdot \frac{\partial h_t}{\partial f_t} * f_t * (1 - f_t) \\ + {W_i}^T \cdot \frac{\partial h_t}{\partial i_t} * i_t * (1 - i_t) \\ + {W_o}^T \cdot \frac{\partial h_t}{\partial o_t} * o_t * (1 - o_t) \\ + {W_C}^T \cdot \frac{\partial h_t}{\partial \bar{C}_t} * \left(1 - \bar{C}_t^{\,2}\right)\end{aligned} \tag{11}$$

综上为 h_t在对其他元素的导数，在代码实现中所重复出现的项可以复用，故仅将所有未知部分展开，重复项未带入。

2. BPTT

关于时间求导，需要将 LSTM 按时间展开，可以获得每个时间点 h_t关于 loss 的求导值。

t 时刻，LSTM 还需要知道 $\frac{\partial L}{\partial h_t}$ 和 $\frac{\partial L}{\partial C_t}$ 即可求出 t 时刻所有变量的导数值，如下图所示

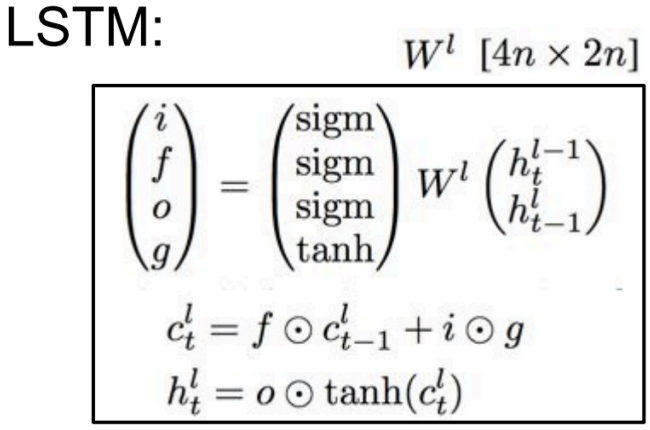


所以 BPTT 的基本行为是，按照时间展开，并从最后一个时间对应的 LSTM cell 开始对各个变量进行求导，并沿时间往更早时间进行传播，直到求出所有变量的导数。

Part II

0. LSTM 的实现

进一步观察 LSTM 的公式可以发现，公式中有很多行为一致的计算，所以将 4 个 W 和 4 个 bias 进行拼接，以达到更少次数的矩阵乘法，从而获得更高效率的代码。具体行为见下图。



(图片来源 http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf)

根据该图中简化后的公式，以及第一问中的具体求解，可以获得 pytorch 实现的 LSTM(见 model.py)和 numpy 实现的 LSTM(见 MyLSTM_np.py)

1. 初始化

1.1. 为何不出初始化为零

1.1.1. 初始化为零会导致前向传播的大量数值为零，同样也会导致反向传播，如果过多的数据被初始化为 0 的话，会导致模型中的参数无法进行学习，自始至终数值都保持零。

1.1.2. 如果某一部分的参数被设置为零时，可能参数的数据也能通过反向传播获得数值（成为非零值）。但是因为参数具有对称性， 导致反向传播得到的数值同样具有对称性（某些数据的更新是相同或者相似的），导致模型的训练会花费更大的时间甚至没法训练到最优。

在 deep learning 书中有提到对称性相关的内容

“Perhaps the only property known with complete certainty is that the initial parameters need to “break symmetry” between different units. If two hidden units with the same activation function are connected to the same inputs, then these units must have different initial parameters. If they have the same initial parameters, then a deterministic learning algorithm applied to a deterministic cost and model will constantly update both of these units in the same way.”[1]

1.1.3. 为了直观的感受上面两段话，我将代码初始化方式进行了修改，做了三组实验。（因为超参设置不为重点，故不在此提及）

测试一（embedding 和 LSTM 均设置为零）

可以看出训练五十个 step 之后，参数数值均全为零。

```
embeddings.weight
Parameter containing:
tensor([[0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]], requires_grad=True)
weight_ih
Parameter containing:
tensor([[0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]], requires_grad=True)
epoch 0 step 50 loss 4.1293625831604
```

测试二（embedding 设置为零，LSTM 使用 xavier uniform 初始化）

观察到 embedding weight 中每一行的数值绝对值几乎相同，在 embedding 空间中张成的特征向量较为对称。

```
embeddings.weight
Parameter containing:
tensor([[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        [ 0.0015,  0.0023,  0.0043,  ..., -0.0007, -0.0023, -0.0011],
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        ...,
        [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
        [-0.0285,  0.0324, -0.0341,  ...,  0.0241, -0.0319,  0.0335],
        [-0.0286,  0.0296, -0.0291,  ...,  0.0292, -0.0288,  0.0304]],
       requires_grad=True)
weight_ih
Parameter containing:
tensor([[-0.0126,  0.0395, -0.0691,  ..., -0.0688, -0.0032, -0.0623],
        [ 0.0496,  0.0482,  0.0024,  ...,  0.0293,  0.0653,  0.0117],
        [ 0.0487, -0.0026, -0.0939,  ...,  0.0250, -0.0454,  0.0394],
        ...,
        [-0.0463, -0.0291, -0.0004,  ...,  0.0586,  0.0334,  0.1052],
        [-0.0329, -0.0157, -0.0560,  ..., -0.0444, -0.0522, -0.0899],
        [ 0.0569,  0.0236,  0.0483,  ...,  0.0090,  0.0926, -0.0342]],
       requires_grad=True)
epoch 0 step 50 loss 3.6402790546417236
```

测试三（embedding 使用默认随机初始化，LSTM 设置为零）

可以观察到参数 weight_ih(lstm 中与 X 相乘的矩阵)中每一行的数据相差都不大，也体现出 zero 初始化导致的对称性

```
embeddings.weight
Parameter containing:
tensor([[ 1.3688,  1.1269,  1.1907,  ..., -1.1830, -1.8229, -0.1366],
        [-0.3534,  2.4159,  0.9774,  ..., -0.1710,  0.2309,  1.1607],
        [ 0.3266, -1.5997, -0.6339,  ..., -1.1567, -0.4575,  0.3388],
        ...,
        [-0.9662, -0.8951,  0.0644,  ...,  1.8365,  0.9758, -0.4084],
        [ 0.7470,  0.4163,  0.1973,  ...,  0.4356,  2.5062, -1.1960],
        [-1.7391,  1.0641, -1.8501,  ...,  1.1508,  0.2279,  0.2796]],
       requires_grad=True)
weight_ih
Parameter containing:
tensor([[-0.0133, -0.0132, -0.0136,  ..., -0.0189, -0.0185, -0.0158],
        [ 0.0145,  0.0144,  0.0147,  ...,  0.0203,  0.0204,  0.0174],
        [-0.0135, -0.0134, -0.0138,  ..., -0.0206, -0.0200, -0.0167],
        ...,
        [ 0.0143,  0.0166,  0.0148,  ...,  0.0252,  0.0254,  0.0226],
        [ 0.0224,  0.0222,  0.0209,  ...,  0.0150,  0.0142,  0.0144],
        [ 0.0154,  0.0155,  0.0156,  ...,  0.0214,  0.0179,  0.0104]],
       requires_grad=True)
epoch 0 step 50 loss 2.61483097076416
```

1.2. 如何初始化

初始化方法其实还是一个较为活跃的研究领域，许多论文做出了许多假设推测得出一些初始化方法，且在实际运用过程中较为有效，下面进行了简单的分析并用实验说明做出对比。

对于公式，设定 n_{in} 个输入和 n_{out} 个输出的全连接层

1.2.1. Standard normal

Standard normal 使得神经元的输出具有相同的方差，以提高训练收敛速度。

$$W_{i,j} \sim N(0, \frac{1}{\sqrt{\mathrm{n}_{in}}})$$

1.2.2. Xavier normal

考虑$\sum_{i=1}^{n} x_i w_i$的方差，假设 $E(X) = E(W) = 0$ 以及 weights 与输入 X 几乎无关。可以得到$var[\sum_{i=1}^{n} x_i w_i] = Var[x](n\, Var[W])$[2]

意向性的我们希望 n Var[W] 为 1。

$$W_{i,j} \sim N(0, \sqrt{\frac{2}{\mathrm{n}_{in} + \mathrm{n}_{out}}})$$

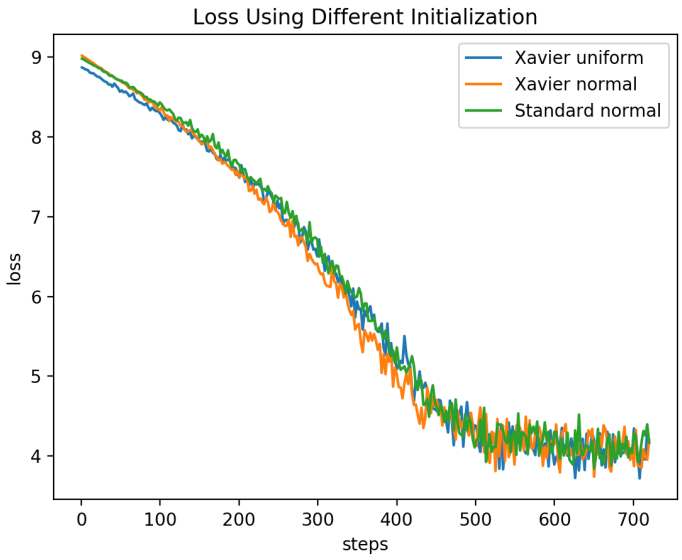
1.2.3. Xavier uniform

与 Xavier normal 类似，只是将 normal 改为 uniform

$$W_{i,j} \sim U(-\sqrt{\frac{6}{\mathrm{n}_{in}+\mathrm{n}_{out}}}, \sqrt{\frac{6}{\mathrm{n}_{in}+\mathrm{n}_{out}}})$$

1.2.4. 实验(使用 SGD 进行优化)

可以看出三个方式都因为对初始化方差进行了考虑，都在一定程度上缓和了梯度爆炸和梯度消解问题。



2. 诗歌生成

2.1. 数据集

本实验使用全唐诗[4]数据集，并将原数据集中繁体字简化为简体字，并为了训练的方便，将所有的诗句都设置为 125 字以内。如果小于 125 字，则在诗歌前方进行 padding 直到长度为 125。在 dataUtil.py 中可以找到获取数据的函数，其中利用 fastNLP 生成了 vocabulary。

2.2. 参数设置

模型见文件 model.py，训练和生成代码见 main.py。

以下为训练时使用的参数。

```
use_gpu = True
batch_size = 128
lr = 1e-3
epoch = 50
# gen_every = 25
gen_every = 50
max_gen_len = 200
save_every = 5
pp_every = 100
tolerance = 10
```

2.3. 生成结果

“日”：

日夕朱鸪秀，分明始陈吟。
主人思挟袖，端折逐风威。
雾为阊岑谷，天盘险外诛。
宿临光照白，松草叶黄埃。

“红”：

红庭万树绿沈愁，舍客晚吟秋天霞。
譬如擩风怨嵩谷，稍疑竹树花花木。

“山”

山里北峯号旧宅，竹林何事无人处。
东南更爱不择重，通筑五侯城外积。

“夜”

夜得秋光明月明，畾含偏婉画屏风。
坐来薄势自相讶，垠碧无踪可依迟。

“湖”

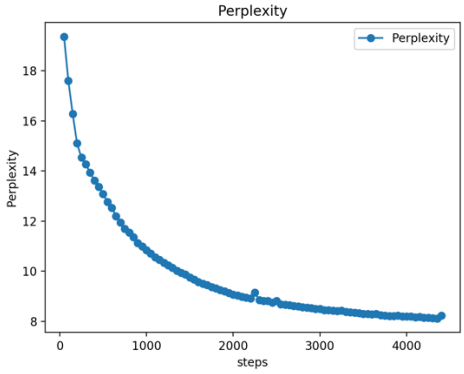
湖上忆黄埃，王孙何所钦。
他人今老去，遇夕宿郊官。
买鹰神原上，白夏借巢蛇。
龙蛇昏巢炯，一旦塞南师。

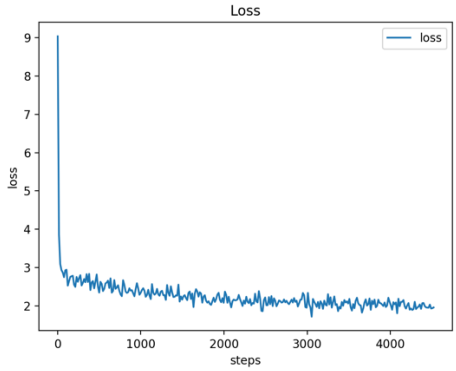
“海”

海郭楚云起，钟声遥夜深。
落天开光细，芳草晓云低。
欲动轩庭出，霜风飐飐台。
水崖连席水，分袂带蕃声。

“月”
月空见西宫，出谷东净流。
萧萧万里余，浯山一片除。
一室秋天涯，巴垒松溪雪。
青林下苔壁，花开开月面。

2.4. Perplexity 和 Loss





2.5. Numpy 版本实现

在 MyLSTM_np.py 中，实现了代码 numpy 版本的 LSTMCell 的前向求值和后向传播，以及 LSTM 的前向求值和后向传播。并实现了 test 函数，用于 loss 和 gradient 的比较。以下为部分比较结果。

```
[[[ 5.09387360e-04 -5.71846752e-04 -8.86510110e-04  3.67121303e-05
   -4.39971500e-05  7.81986258e-04 -7.09567542e-04 -5.93971080e-04]]]
[[ 2.42422401e-04 -2.60072777e-04 -3.32500010e-04  1.47274934e-05
  -2.04968635e-05  3.19051656e-04 -2.88645978e-04 -2.98276155e-04
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  -5.83808688e-03  3.35411100e-03  3.75523408e-03 -2.36023820e-04
   3.47262367e-04 -5.75437744e-03  3.46210747e-03  4.07794817e-03
```

3. 优化探究

3.1. SGD

$$\theta = \theta - \eta \cdot \nabla_\theta \mathcal{J}(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

最简单但是也较为直接和常用的优化方法，其中将 batch 中的多个样本进行联合，让降低梯度的随机性，从而一定程度上增加更新方向的准确性。

3.2. SGD with momentum

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

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

SGD 中可以看出，对于参数的优化完全依赖于一个 batch 的输入，随机性仍然较大，因此模拟物理中动量的概念。引入动量项之后，多个 batch 之间的随机性可以互相抵消，让前期的更新方向更加准确，速度更快一些。

3.3. Nexterov

引入动量项后的 g_t 不应该是加入 m_t 项之前的 gradient，而应该是之后的。因此将公式更新为

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

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

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

nesterov 项在梯度更新时做一个校正，避免前进太快，同时提高灵敏度。

以上都是人工直接硬性设置学习率，一下几种为自适应学习率。

3.4. Adagrad

Adagrad 对学习率进行了一个约束。

$$n_t = n_{t-1} + {g_t}^2$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} * g_t$$

可以看出公式中$\sqrt{n_t + \epsilon}$对梯度和学习率拥有类似于正则化的功能，g_t 较小的时候能够放大梯度，而后期 g_t 较大的时候能够减小梯度。

3.5. Adadelta

从 Adagrad 可以看出，后期会因为分母项不断增大，而导致更新速度过慢的情况出现，Adadelta 对其进行了优化。

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

$$\Delta\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} * g_t$$

除了引入了类似于动量的优化之外，Adadelta 还减去了对全局学习率的依赖

$$E(g^2)_t = \rho * E(g^2)_{t-1} + (1-\rho) * {g_t}^2$$

$$\Delta\theta_t = -\frac{\sqrt{\sum_{r=1}^{t-1} \Delta\theta_r}}{\sqrt{E(g^2)_t + \epsilon}}$$

3.6. Adam

Adam(Adaptive Moment Estimation)本质上是带有动量项的 RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。[3]

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

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

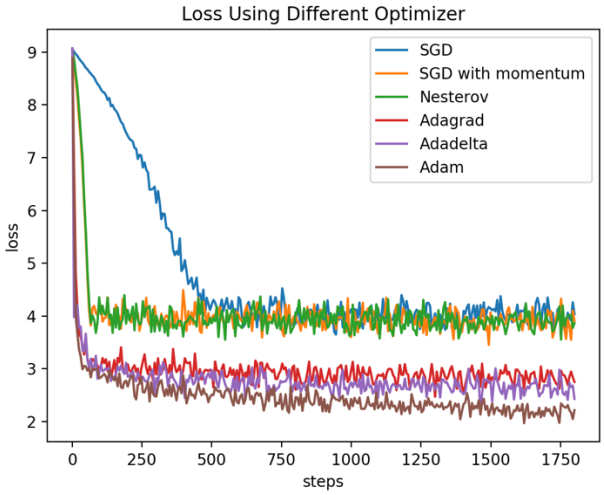
$$\widehat{m_t} = \frac{m_t}{1-\mu^t}$$

$$\widehat{n_t} = \frac{n_t}{1-\nu^t}$$

$$\Delta\theta_t = -\frac{\widehat{m_t}}{\sqrt{\widehat{n_t}+\epsilon}} * \eta$$

3.7. 对比实验

下图为利用不同优化方法获得的 loss。可以看出 loss 的下降与上述分析基本一致，对于该问题，使用 Adam 的效果最优。



4. Supplementar

4.1. PP 值计算讨论

在计算 pp 值时，在代码中直接使用了 pytorch 中提供的 CrossEntropyLoss，该函数在计算交叉熵时会根据 batch size 进行平均，由 Jensen 容易得值，该种方式计算得到的“pp 值”会小于要求中的每条输入进行计算所获得的 pp 值。但是相对大小没有改变，因此仍然满足 early ending 的需求。

4.2. fastNLP

因为时间仓促没有深入探究 fastnlp，但是在使用 vocabulary 的时候发现一个较为不便捷的地方，函数左值进行简单的优化可能比较便于灵活地使用。

*想象中的 fastNLP

```
vocab = Vocabulary(min_freq=10, padding="</s>").add_word_lst(wordlist).build_vocab()
```

*实际中的 fastNLP

```
vocab = Vocabulary(min_freq=10, padding="</s>")
vocab.add_word_lst(wordlist)
vocab.build_vocab()
```

Reference

[1] https://machinelearningmastery.com/why-initialize-a-neural-network-with-random-weights/
[2] http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf
[3] https://zhuanlan.zhihu.com/p/22252270
[4] https://github.com/chinese-poetry/chinese-poetry