PRML Assignemnt 3 报告

16307130076 赵伟丞

LSTM 原理推导

LSTM 后向传播导数推导

首先，已知 LSTM 的前向传播公式如下：

$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

$\hat{c}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$

$c_t = f_t * c_{t-1} + i_t * \hat{c}_t$

$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

$h_t = o_t * tanh(c_t)$

显然有，

$\frac{\partial h_t}{\partial o_t} = diag[tanh(c_t)]$

$\frac{\partial h_t}{\partial c_t} = diag[o_t * (1 - tanh(c_t)^2)]$

$\frac{\partial c_t}{\partial f_t} = diag[c_{t-1}]$

$\frac{\partial c_t}{\partial i_t} = diag[\hat{c}_t]$

$\frac{\partial c_t}{\partial \hat{c}_t} = diag[i_t]$

$\frac{\partial c_t}{\partial c_{t-1}} = diag[f_t]$

于是有，

$\frac{\partial h_t}{\partial f_t} = o_t * (1 - tanh(c_t)^2) * c_{t-1}$

$\frac{\partial h_t}{\partial i_t} = o_t * (1 - tanh(c_t)^2) * \hat{c}_t$

$\frac{\partial h_t}{\partial \hat{c}_t} = o_t * (1 - tanh(c_t)^2) * i_t$

$\frac{\partial h_t}{\partial c_{t-1}} = o_t * (1 - tanh(c_t)^2) * f_t$

又因为，

$$o_t = \sigma(net_{o,t})$$
$$net_{o,t} = W_o \cdot z + b_o$$

$$f_t = \sigma(net_{f,t})$$
$$net_{f,t} = W_f \cdot z + b_f$$

$$i_t = \sigma(net_{i,t})$$
$$net_{i,t} = W_i \cdot z + b_i$$

$$\hat{c}_t = tanh(net_{\hat{c},t})$$
$$net_{\hat{c},t} = W_c \cdot z + b_c$$

显然，

$$\frac{\partial o_t}{\partial net_{o,t}} = diag[o_t * (1 - o_t)]$$

$$\frac{\partial net_{o,t}}{\partial z} = W_o$$

$$\frac{\partial f_t}{\partial net_{f,t}} = diag[f_t * (1 - f_t)]$$

$$\frac{\partial net_{f,t}}{\partial z} = W_f$$

$$\frac{\partial i_t}{\partial net_{i,j}} = diag[i_t * (1 - i_t)]$$

$$\frac{\partial net_{i,t}}{\partial z} = W_i$$

$$\frac{\partial \hat{c}_t}{\partial net_{\hat{c},t}} = diag[1 - {\hat{c}_t}^2]$$

$$\frac{\partial net_{\hat{c},t}}{\partial z} = W_c$$

由上面一系列式子可得，

$$\frac{\partial h_t}{\partial z} = \frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial x_t} = tanh(c_t) * o_t * (1 - o_t) * W_o$$
$$= o_t * (1 - tanh(c_t)^2) * c_{t-1} * f_t * (1 - f_t) * W_f$$
$$= o_t * (1 - tanh(c_t)^2) * \hat{c}_t * i_t * (1 - i_t) * W_i$$
$$= o_t * (1 - tanh(c_t)^2) * i_t * (1 - {\hat{c}_t}^2) * W_c$$

同样的，

$$\frac{\partial h_t}{\partial W_o} = tanh(c_t) * o_t * (1 - o_t) * z$$
$$\frac{\partial h_t}{\partial b_o} = tanh(c_t) * o_t * (1 - o_t)$$

$$\frac{\partial h_t}{\partial W_f} = o_t * (1 - tanh(c_t)^2) * c_{t-1} * f_t * (1 - f_t) * z$$
$$\frac{\partial h_t}{\partial b_f} = o_t * (1 - tanh(c_t)^2) * c_{t-1} * f_t * (1 - f_t)$$

$$\frac{\partial h_t}{\partial W_i} = o_t * (1 - tanh(c_t)^2) * \hat{c_t} * i_t * (1 - i_t) * z$$
$$\frac{\partial h_t}{\partial b_i} = o_t * (1 - tanh(c_t)^2) * \hat{c_t} * i_t * (1 - i_t)$$

$$\frac{\partial h_t}{\partial W_c} = o_t * (1 - tanh(c_t)^2) * i_t * (1 - \hat{c_t}^2) * z$$
$$\frac{\partial h_t}{\partial b_c} = o_t * (1 - tanh(c_t)^2) * i_t * (1 - \hat{c_t}^2)$$

至此，所有导数推导完毕。

时序上的导数传播

时序上的传播，就是依照前向传播的反向顺序，利用$\frac{\partial h_t}{\partial h_{t-1}}$进行时序上的反向传播，然后在每个计算单元中计算相应的导数，即可实现在序列中沿时间顺序的反向传播。

基于Numpy的LSTM实现

位于LSTM_np.py文件中，检测方法为：使用同tensor（array）输入是进行一步迭代后计算出的梯度一致。

基于 LSTM 的唐诗生成模型

参数的初始化

参数初始化显然不能全 0，这会导致模型参数对称的问题，导致模型无法被恰当地训练。我个人在代码中使用了 PyTorch 提供的参数初始化中的 uniform_来初始化各个参数，其从(-\frac{1}{\sqrt{hidden_size}},\frac{1}{\sqrt{hidden_size}})均匀分布中取值来初始化模型参数（此部分参考了 PyTorch 自带的 LSTM 的实现。

模型实现

额外数据集

本人在训练模型时使用了全唐诗来进行训练。包括训练集中 34552 首诗和测试集中 8638 首诗。

数据预处理

对于不足 seq_len 长度的序列，在其前面补充<pad>（之所以在前面填充是因为在后面填充会出现输出全为<pad>的情形。对于超过 seq_len 长度的序列，以 seq_step 为步长截断为长度为 seq_len 的序列。字典则通过 fastNLP 生成。

各个超参

- 词典大小：6253
- batch size：128
- 学习率：1e-3
- 序列长度：200
- 序列步长：100
- input size：128
- hidden size：128

生成结果

以下诗句的划分为人工操作。

- 日清忆道年城别，上入春无翠花高。山万风云此去水，我分生新下南夜。
- 红日春高月，时青出云东。天白从来君，白常时江云。长风千中前，燕家行满金。百道流青水，一烟黄旧来。
- 山秋月上春，已寒烟旧中。无间高心来，初云海五寒。
- 夜心无为如玉白，我有一月此台青。水前大山时天相，开白清天长风上。君云天人四不闻，波相一春上千金。之风是心与天青，一别何日此岂客。
- 湖十高有日，山月南云春。清天万上今，旧北无朝心。自下高月此，九始问行多。如是君云长，得思时不多。
- 海何南高山，青石小人山。山上入东台，风开朝野双。天新不山河，酒上君青石。龙门千一别，晓日山万西。
- 月上春青多，行天何不出。三五我何马，长中天下白。云白常千君，春风无水闻。高风春草有，此台花未平。

困惑度计算

由于困惑度本身可以解释为交叉熵的指数形式，因此本人选择在测试集上计算交叉熵（在每个 batch 上分别计算，不求平均），然后在取 e 指数之后取平均值，得到困惑度。然后基于测试集上的困惑度是否连续两次上升决定是否进行 early stop。在最后使用的模型中（经过20个epoch的训练），在测试集上困惑度平均值为 5.92.

优化器的选择

本人尝试使用了 PyTorch 已经实现的优化器中较为常用的两种，Adam 和 SGD with momentum。损失函数使用交叉熵作为损失函数。其中 SGD with momentum 中 momentum 取 0.9。这两种优化中 Adam 表现更好，主要是收敛速度更快（少了两个 epoch)，而且最后的 loss 也相对较小。