基于 LSTM 的语言模型

技术报告

计算机 1905 赵旭

2021 年 11 月

摘要

本次的实践的基础部分主要是实现一个基于 LSTM 的语言模型，且仅需要手动实现循环单元部分即可，不能使用框架提供的标准模块。提高部分需要在自己搭建出的 LSTM 网络基础上，实现双层的 LSTM 网络。

在基础部分，我实现了两种不同的循环单元的设计方法，第一种是助教学长在实践课上 PPT 中所给出的 LSTM 循环单元的模型，第二种是 pytorch 官方网站中给出的 LSTM 循环单元模型。这两种模型原理是相同的，不同是每个门以及记忆更新的计算方法，因此矩阵的维度变换也不同，之后的篇幅中我会阐述我对这两种方法不同处的理解。

在进行上述两种方法实现的过程中，我遇到了一些比较棘手的问题，通过查阅一些相关的资料并且询问了助教学长，最后解决了遇到的这些问题，在这里非常感谢刘新宇学长、吕传昊学长以及穆永誉学长的指导和帮助！在后面的篇幅中我会一一记录我在实验中遇到的问题以及对问题的分析和解决过程。

在提高部分，我基于基础部分的两种不同 LSTM 的循环单元分别实现了两个双层的 LSTM 网络，其实基础部分的循环单元完成后，这个部分就是在循环单元中多加一层相同的公式，并将上一层的输出变成这一层的输入即可，因此在这个部分并没有遇到什么特别的问题，但是最后双层的 LSTM 网络的效果相对于单层的 LSTM 网络并没有提升多少，具体原因还有待商榷。

最后，通过本次课程的学习以及实践过程的锻炼，我从一个一开始对神经网络一无所知小白变成了现在能够理解神经网络的原理并且可以自己编写一个神经网络，我觉得这是我学习该门课程最大的收获，其次对于自然语言处理的一些语言模型的原理我也有了一些深入的了解，能够看懂并且编写部分代码，这也算我在该门课程中一个很大的收获。

1 系统设计

模型一（助教学长在实践课 PPT 中给出），公式如下：

遗忘门：

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

输入门：

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

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

输出门：

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

$$\boldsymbol{h_t = o_t \cdot tanh(c_t)}$$

记忆更新：

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

模型二（pytorch 官网中给出），公式如下：

$$\boldsymbol{i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})}$$

$$\boldsymbol{f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})}$$

$$\boldsymbol{g_t = tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})}$$

$$\boldsymbol{o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})}$$

$$\boldsymbol{c_t = f_t \odot c_{t-1} + i_t \odot g_t}$$

$$\boldsymbol{h_t = o_t \odot tanh(c_t)}$$

在摘要中，我提到过这两种模型在原理上是相同的，不同之处在于每个门以及记忆更新的计算方法，我的理解是，在模型一中，计算遗忘门 f_t、输入门 i_t 以及输出门 o_t 时，需要将当前的输入 x_t 以及上一次的输出 h_{t-1} 拼接成一个矩阵作为处理的输入进行线性变换处理，然后训练参数 W_f 等参数，而在模型二中，将当前的输入 x_t 以及上一次的输出 h_{t-1} 分开进行处理，分别进行线性变换，然后训练 W_{if}，

W_{hf}等参数。其实就相当于将W_f拆成了W_{if}，W_{hf}两个参数进行训练，从这个方面来说，这两种模型最后训练的结果应该差别不大，或者说应该是相似的。

2 实验过程分析

2.1 Version 1

下面是我编写的一个版本的 LSTM 语言模型中的循环单元，该版本运行时，报错显示 Trying to backward through the graph a second time,but the buffers have already been freed。意思是说在进行第二次反向传播的过程中，buffers 的梯度已经被释放了。起初我并不理解这个报错的含义，于是到 CSDN 上去查了一下，然后大致弄明白了这个报错的原因，具体原因可以参考这篇文章：https://blog.csdn.net/SY_qqq/article/details/107384161。由于篇幅限制，这里不详细说明。

通过这篇文章，我分析这里报错可能是因为在进行反向传播的过程中，con 这个参数是依赖于 X[i]的，因此反向传播的过程中需要从 con 传到 X[i]，但是每次循环的时候前一个 con 都会被下一 con 替换掉，导致之前的 con 无法被保留，因此在传播回 X[i]的时候，对应的 con 找不到了，才会出现这种问题。

```
    def forward(self, X):
        X = self.C(X)
        X = X.transpose(0, 1)
        h_t_1 = hidden
        c_t_1 = memo
        h_t_1 = h_t_1.transpose(0,1)
        h_t = torch.zeros(n_step, n_hidden,emb_size)
        con = torch.zeros(n_hidden,2*emb_size)
        for i in range(n_step):
            con = torch.cat((h_t_1[i],X[i]),1)
            ft = self.sigmoid(self.wf(con))
            it = self.sigmoid(self.wi(con))
            cdet = self.tanh(self.wc(con))
            ot = self.sigmoid(self.wo(con))
            c_t = ft*c_t_1 + it*cdet
            h_t[i] = ot*self.tanh(c_t)
```

```
            h_t_1[i] = h_t[i]
            c_t_1 = c_t
        outputs_o = h_t[-1]
        model = self.W(outputs_o)
        return model
```

2.2 Version 2

对第一个版本的报错进行分析后，我对自己的代码进行了修正，代码如下。该版本虽然解决了第一个版本的问题，说明我的分析是对的，却也引发了新的问题，报错显示：one of the variables needed for gradient computation has been modified by an inplace operation。意思是说梯度计算所需的第一个变量已经被 inplace 操作修改了。看到这个报错后我觉得和第一个问题很像，因为第一个问题是在进行反向传播的时候，某一个需要的梯度没了，而这个是被修改了。因此我接着第一问解决问题的思路去做，我尝试对 c_t 也进行对 con 同样的处理，也尝试将所有门的输出均进行多次保存处理，但是结果均不对，我百思不得其解，无奈之下向助教学长们寻求帮助，穆永誉学长给我看了一篇 CSDN 的博文，并且帮助我改正了代码中的错误。博文链接 https://blog.csdn.net/ncc1995/article/details/99542594。非常感谢穆永誉学长的帮助。看了这篇博文并对比学习了学长帮我改正的代码和 version2 的代码后，我也明白了自己所写循环单元的问题所在，具体分析见 2.3 Final version。

```
    def forward(self, X):
        X = self.C(X)
        X = X.transpose(0, 1)
        h_t_1 = hidden
        c_t_1 = memo
        h_t_1 = h_t_1.transpose(0,1)
        h_t = torch.zeros(n_step, n_hidden,emb_size)
        con = torch.zeros(n_step,n_hidden,2*emb_size)
        for i in range(n_step):
            con[i] = torch.cat((h_t_1[i],X[i]),1)
            ft = self.sigmoid(self.wf(con[i]))
            it = self.sigmoid(self.wi(con[i]))
            cdet = self.tanh(self.wc(con[i]))
```

```
            ot = self.sigmoid(self.wo(con[i]))
            c_t = ft*c_t_1 + it*cdet
            h_t[i] = ot*self.tanh(c_t)
            h_t_1[i] = h_t[i]
            c_t_1 = c_t
        outputs_o = h_t[-1]
        model = self.W(outputs_o)
        return model
```

2.3 Final version

代码如下。通过对比 version 2 中的代码，我终于知道自己的错误在哪了，虽然我一开始对这个错误发生的原理的理解是正确的，但是在分析上出了问题，从 version 1 报错开始我就认为是在反向传播中 con 传不到 X[i]所以报错，但实际上是 con 没有传播到 h_t_1[i]中才报错的，而后面各个门的输出计算又需要依赖于 con[i]进行计算，con 有 n_step 个，各个门的输出每次循环却只有一个，因此在反向传播的过程中导致每个门的输出不知道对应哪个 con，因此报错说梯度计算所需的变量被 inplace 操作修改了。因此其实只要将除了 X[i]外的其他的参数均改成单个参数，不需要多余的中间结果的保存，结果就对了。

通过自己实现 LSTM 语言模型中循环单元，我对于该语言模型循环单元中的矩阵维度的变换理解的更加清楚了，不仅如此，通过对代码报错的分析，我对于神经网络的反向传播的过程也有了更加深入的了解，收获颇多！

```
    def forward(self, X):
        X = self.C(X)
        X = X.transpose(0, 1)
        h_t_1 = torch.zeros(n_hidden,emb_size)
        c_t_1 = torch.zeros(n_hidden,emb_size)
        for i in range(n_step):
            con = torch.cat((h_t_1,X[i]),1)
            ft = self.sigmoid(self.wf(con))
            it = self.sigmoid(self.wi(con))
            cdet = self.tanh(self.wc(con))
            ot = self.sigmoid(self.wo(con))
            c_t = ft*c_t_1 + it*cdet
            h_t = ot*self.tanh(c_t)
            h_t_1 = h_t
```

```
            c_t_1 = c_t
        outputs_o = h_t
        model = self.W(outputs_o)
        return model
```

2.4 Doubles-layer-model

完成单层的 LSTM 语言模型后，双层的 LSTM 相对来说就简单很多了，只需要将上一层的输出作为下一层的输入即可。这里展示模型二的双层 LSTM 网络循环单元，代码如下。

```
    def forward(self, X):
        X = self.C(X)
        h_t_1 = torch.zeros(n_hidden,emb_size)
        c_t_1 = torch.zeros(n_hidden,emb_size)
        h_t_1_two = torch.zeros(n_hidden,emb_size)
        c_t_1_two = torch.zeros(n_hidden,emb_size)
        X = X.transpose(0, 1)
        for x_t in X:
            it = self.sigmoid(self.wii(x_t)+self.whi(h_t_1))
            ft = self.sigmoid(self.wif(x_t)+self.whf(h_t_1))
            gt = self.tanh(self.wig(x_t)+self.whg(h_t_1))
            ot = self.sigmoid(self.wio(x_t)+self.who(h_t_1))
            ct = ft*c_t_1 + it*gt
            ht = ot*self.tanh(ct)
            h_t_1 = ht
            c_t_1 = ct
            x_t_two = h_t_1
            it_two =
self.sigmoid(self.wii_two(x_t_two)+self.whi_two(h_t_1_two))
            ft_two =
self.sigmoid(self.wif_two(x_t_two)+self.whf_two(h_t_1_two))
            gt_two =
self.tanh(self.wig_two(x_t_two)+self.whg_two(h_t_1_two))
            ot_two =
self.sigmoid(self.wio_two(x_t_two)+self.who_two(h_t_1_two))
            ct_two = ft_two*c_t_1_two + it*gt_two
            ht_two = ot_two*self.tanh(ct_two)
            h_t_1_two = ht_two
            c_t_1_two = ct_two
        outputs = ht_two
        model = self.W(outputs) + self.b
        return model
```

3 实验结果记录

3.1 模型一 单层神经网络

实验结果记录如下：

Times	The value of loss	The value of ppl
1	5.667827	289.405
2	5.656630	286.183
3	5.647472	283.574
4	5.660667	287.340
5	5.669947	290.019
6	5.649603	284.179
7	5.670213	290.096
8	5.657224	286.353
9	5.660880	287.401
10	5.637375	280.725
Average	5.657784	286.528

3.2 模型二 单层神经网络

实验结果记录如下：

Times	The value of loss	The value of ppl
1	5.644543	282.744
2	5.645837	283.110
3	5.641981	282.021
4	5.644246	282.660
5	5.651772	284.796
6	5.650905	284.549
7	5.658775	286.797
8	5.649966	284.282
9	5.642108	282.057
10	5.649712	284.210
Average	5.647985	283.723

3.3 模型一 双层神经网络

实验结果记录如下：

Times	The value of loss	The value of ppl
1	5.785598	325.577
2	5.798185	329.701
3	5.777577	322.976
4	5.835456	342.221
5	5.802486	331.122
6	5.811144	334.001
7	5.795996	328.980
8	5.785608	325.580
9	5.805892	332.251
10	5.810545	333.801
Average	5.800847	330.621

3.4 模型二 双层神经网络

实验结果记录如下：

Times	The value of loss	The value of ppl
1	5.747085	313.276
2	5.756645	316.285
3	5.791027	327.349
4	5.747090	313.278
5	5.758680	316.930
6	5.764996	318.938
7	5.772001	321.18
8	5.759750	317.269
9	5.773184	321.560
10	5.779521	323.604
Average	5.764998	318.967

4 实验结果分析

The value of loss	模型一	模型二
单层 LSTM 网络	5.657784	5.647985
双层 LSTM 网络	5.800847	5.764998

The value of ppl	模型一	模型二
单层 LSTM 网络	286.528	283.723
双层 LSTM 网络	330.621	318.967

通过以上两个表格中数据的对比，双层 LSTM 网络的效果并没有单层的效果好，但是两种模型的效果大致相同。

5 实验总结

通过该门课程的学习，我有了很多的收获，也学到了很多东西，包括单词的表示，如独热表示，分布表示以及 TF-IDF 模型等，到单词嵌入如 Word2Vec 模型、Skip-gram 模型等，再到学习了前馈神经网络、基于循环神经网络的序列表示以及 LSTM 模型等。通过本次编写 LSTM 语言模型中的循环单元的锻炼，我在实践的动手能力上也有了很大的进步，如能够自己编写一个简单的神经网络，对于 LSTM 语言模型的循环单元中各个参数矩阵的维度变化等也有了更清晰的理解。同时在这个实验过程中，通过解决自己遇到的问题，我对于神经网络中反向传播的过程也有了很清晰的认识和理解。不仅如此在该门课程中，我也认识了热心、善良的助教学长们，很耐心地给了我很多的指导和帮助。总之，在本门课程的学习和实践过程中，我收获颇多。

6 致谢

首先衷心感谢肖桐老师和马安香老师的在课程中的答疑解惑，在我课程学习中遇到的困难和疑惑给予了许多解答和意见，同时衷心感谢刘新宇学长、吕传昊学长以及穆永誉学长在我实验所遇到的困难中给予的悉心指导和帮助。还要感谢在自然语言处理课程中共同学习的同学们，大家在课程的学习中互相学习，相互帮助，共同度过了一段难忘的时光！最后，感谢评阅报告的老师和学长学姐们的辛苦工作！