Report for NLP-Beginner task 4

基于LSTM+CRF的序列标注

叶栩冰

---

1 Task

用LSTM、GRU来训练字符级的语言模型，计算困惑度

1. 参考
    1. 《神经网络与深度学习》 第6、15章
2. 数据集：poetryFromTang.txt
3. 实现要求：Pytorch
4. 知识点：
    1. 语言模型：困惑度等
    2. 文本生成
5. 时间：两周

2 Environment & Data

2.1 Environment

> Python ~= 3.6
>
> IDE : Jetbrains Pycharm
>
> torch ~= 1.10.0
>
> random

2.2 Data

数据为诗句，共163首，链接如下：https://github.com/FudanNLP/nlp-beginner/blob/master/poetryFromTang.txt。但是原数据中含有部分异常数据（含字母等），由于数据量不大，我进行了人工处理。

Example：

> **原数据**：
> 巴山上峡重复重，阳台碧峭十二峰。荆王猎时逢暮雨，
> 夜卧高丘梦神女。轻红流烟湿艳姿，行云飞去明星稀。
> 目极魂断望不见，猿啼三声泪沾衣。

> **Input**：每行最大字符数：12；行数：6
> **Input**：藏头："藏头诗"
>
> **Output**：对应生成的诗句

此外需要注意的是，由于数据量较小，因此本次作业batch仅设置为1。

3 Method & Model

3.1 Embedding

因为本次作业的自然语言类型为中文，没有比较好的预训练模型，因此采用随机初始化。此外，同task2/3，此外，本次作业除了给定的各种序列类别之外，还要另外多加3个类别，分别是: < pad >,< start >,< end >，分别代表padding（即补位，使句子达到同一个长度），句子开头和句子结尾，总共C类标签。但是不同的是，pad由于batch的设置，不参与计算。

3.2 LSTM/GRU

LSTM在这里不再进行介绍，GRU也是较为经典的循环神经网络，因此简要介绍下二者的不同。

GRU（Gate Recurrent Unit）是循环神经网络（Recurrent Neural Network, RNN）的一种。和LSTM（Long-Short Term Memory）一样，也是为了解决长期记忆和反向传播中的梯度等问题而提出来的。

GRU和LSTM在很多情况下实际表现上相差无几，那么为什么我们要使用新人GRU（2014年提出）而不是相对经受了更多考验的LSTM（1997提出）呢。

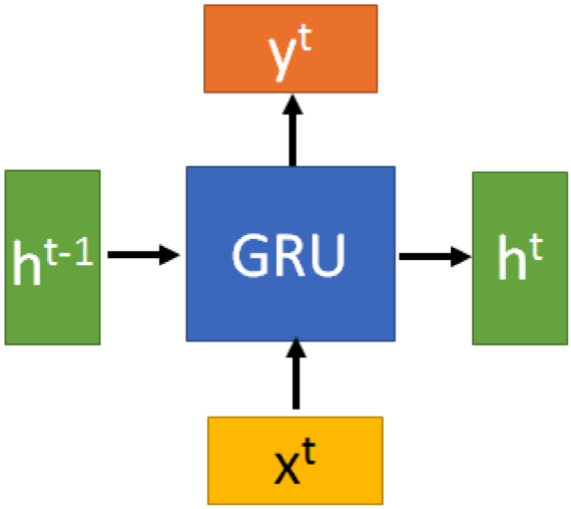
下图引用论文中的一段话来说明GRU的优势所在。即更易于计算。

> We choose to use Gated Recurrent Unit (GRU) (Cho et al., 2014) in our experiment since it performs similarly to LSTM (Hochreiter & Schmidhuber, 1997) but is computationally cheaper.

3.2.1 输入输出结构

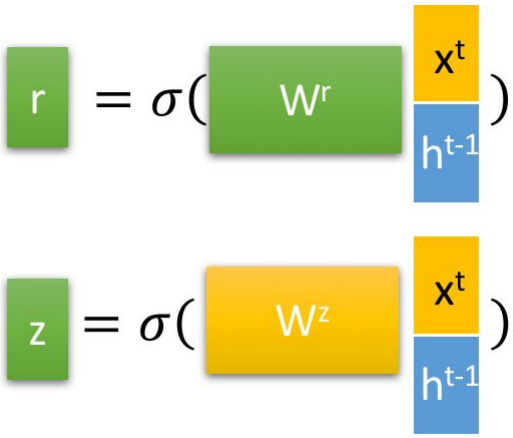
GRU的输入输出结构与普通的RNN是一样的。

有一个当前的输入 x^t，和上一个节点传递下来的隐状态（hidden state） h^{t-1}，这个隐状态包含了之前节点的相关信息。结合 x^t 和 h^{t-1}，GRU会得到当前隐藏节点的输出 y^t 和传递给下一个节点的隐状态 h^t。

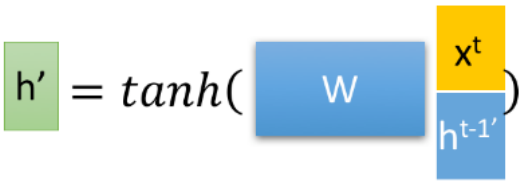


3.2.2 内部结构

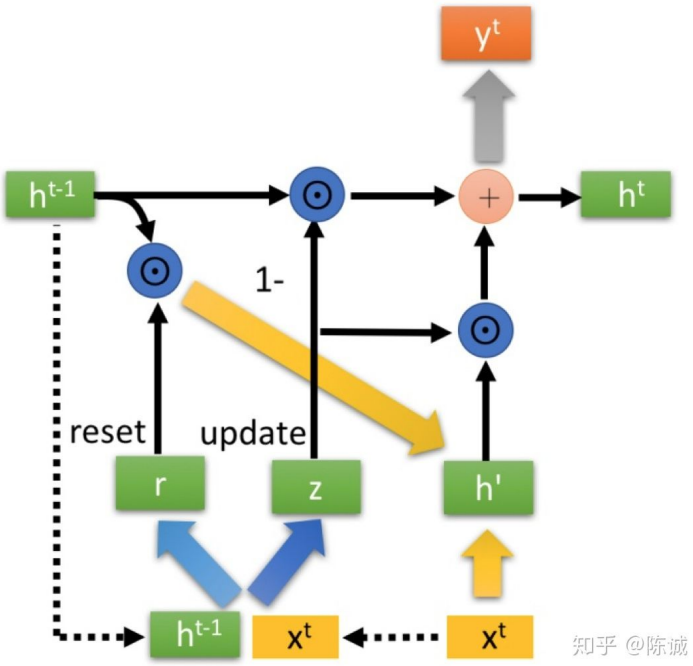
首先，我们先通过上一个传输下来的状态 h^{t-1} 和当前节点的输入 x^t 来获取两个门控状态。如下图2-2所示，其中 r 控制重置的门控（reset gate），z 为控制更新的门控（update gate）。



与LSTM分明的层次结构不同，得到门控信号之后，首先使用重置门控来得到**"重置"**之后的数据 $h^{t-1'} = h^{t-1} \odot r$，再将 $h^{t-1'}$ 与输入 x^t 进行拼接，再通过一个tanh激活函数来将数据放缩到**-1~1**的范围内。即得到如下图2-3所示的 h' 。



这里的 h' 主要是包含了当前输入的 x^t 数据。有针对性地对 h' 添加到当前的隐藏状态，相当于"记忆了当前时刻的状态"。类似于LSTM的选择记忆阶段。



3.2.3 更新记忆

最后就是GRU最关键的一个步骤，我们可以称之为"更新记忆"阶段。在这个阶段，我们同时进行了遗忘了记忆两个步骤。我们使用了先前得到的更新门控 z （update gate）。更新表达式为：

$$h^t = (1-z) \odot h^{t-1} + z \odot h'$$

首先再次强调一下，门控信号（这里的 z ）的范围为0~1。门控信号越接近1，代表"记忆"下来的数据越多；而越接近0则代表"遗忘"的越多。GRU很聪明的一点就在于，我们使用了同一个门控 z 就同时可以进行遗忘和选择记忆（LSTM则要使用多个门控）。

- $(1-z) \odot h^{t-1}$ ：表示对原本隐藏状态的选择性"遗忘"。这里的 $1-z$ 可以想象成遗忘门（forget gate），忘记 h^{t-1} 维度中一些不重要的信息。
- $z \odot h'$ ： 表示对包含当前节点信息的 h' 进行选择性"记忆"。与上面类似，这里的 $(1-z)$ 同理会忘记 h' 维度中的一些不重要的信息。或者，这里我们更应当看做是对 h' 维度中的某些信息进行选择。
- $h^t = (1-z) \odot h^{t-1} + z \odot h'$ ：结合上述，这一步的操作就是忘记传递下来的 h^{t-1} 中的某些维度信息，并加入当前节点输入的某些维度信息。

可以看到，这里的遗忘 z 和选择 $(1-z)$ 是联动的。也就是说，对于传递进来的维度信息，我们会进行选择性遗忘，则遗忘了多少权重 （z），我们就会使用包含当前输入的 h' 中所对应的权重进行弥补 $(1-z)$ 。以保持一种"恒定"状态。

3.2.4 LSTM & GRU

GRU是在2014年提出来的，而LSTM是1997年。他们的提出都是为了解决相似的问题，那么GRU难免会参考LSTM的内部结构。 r (reset gate)实际上与他的名字有点不符。我们仅仅使用它来获得了 h' 。那么这里的 h' 实际上可以看成对应于LSTM中的hidden state；上一个节点传下来的 h^{t-1} 则对应于LSTM中的cell state。1-z对应的则是LSTM中的 z^f forget gate，那么 z我们似乎就可以看成是选择门 z^i 了。

与LSTM相比，GRU内部少了一个"门控"，参数比LSTM少，但是却也能够达到与LSTM相当的功能。考虑到硬件的计算能力和时间成本，因而很多时候我们也就会选择更加"实用"的GRU。

3.3 困惑度

困惑度（perplexity）的基本思想是：给测试集的句子赋予较高概率值的语言模型较好,当语言模型训练完之后，测试集中的句子都是正常的句子，那么训练好的模型就是在测试集上的概率越高越好，公式如下：

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N}}$$

由公式可知，句子概率越大，语言模型越好，迷惑度越小。

下面是一些 ngram 模型经 训练文本后在测试集上的困惑度值：

- Results from Goodman ("A bit of progress in language modeling"), where $|\mathcal{V}| = 50,000$
- A trigram model: $p(x_1 \dots x_n) = \prod_{i=1}^{n} q(x_i | x_{i-2}, x_{i-1})$.
  Perplexity $= 74$
- A bigram model: $p(x_1 \dots x_n) = \prod_{i=1}^{n} q(x_i | x_{i-1})$.
  Perplexity $= 137$
- A unigram model: $p(x_1 \dots x_n) = \prod_{i=1}^{n} q(x_i)$.
  Perplexity $= 955$

可以看到，之前学习的 trigram 模型经训练后，困惑度由955跌减至74，这是十分可观的结果。

3.4 Model

```
self.dropout = nn.Dropout(drop_out)
        _x = nn.init.xavier_normal_(torch.Tensor(len_words, len_feature))
        self.embedding = nn.Embedding(num_embeddings=len_words,
embedding_dim=len_feature, _weight=_x)
        # 只有这一层不同
        if strategy == 'lstm':
            self.gate = nn.LSTM(input_size=len_feature, hidden_size=len_hidden,
batch_first=True)
        elif strategy == 'gru':
            self.gate = nn.GRU(input_size=len_feature, hidden_size=len_hidden,
batch_first=True)
        else:
            raise Exception("Unknown Strategy!")
        self.fc = nn.Linear(len_hidden, len_words)
```

4 Train & Result

4.1 Train

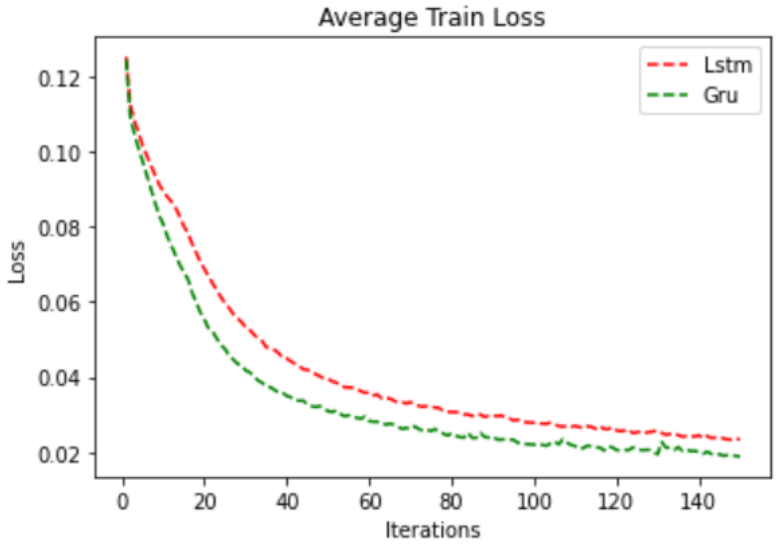
具体训练参数如下：

- loss : Cross Entropy Loss
- random embedding
- lr = 0.001

- epochs = 150
- l_h, l_f：50
- batch size = 1

4.2 Result

无法通过准确率来评估本任务，从训练的loss来看GRU 比 LSTM 稍好一些。



4.3 Analysis

1. Embedding

   我没有找到适宜中文数据的词袋模型，因此选取了随机初始化。这导致当想要生成的藏头词不包含在词袋内会出现OOV问题。

2. 预测字 to 预测词

   我对此问题的处理是这样的：

   - 随机初始化一个向量
   - 输入到 LSTM / GRU 中，得到新的向量，这个向量长度与输入相同
   - 输入到全连接层，得到新的向量，长度为C，代表下一个字在C种字符的得分
   - 对该向量取最高分，对应第i个索引（index），就是下一个字
   - 如果下一个字是 “句号” 或者 “< end >”，给该诗句画上句号，开始重复以上步骤生成下一句。

3. LSTM & GRU

   两种模型训练出的差异在Method部分我对此进行了对比与分析。总之与LSTM相比，GRU内部少了一个”门控“，参数比LSTM少，但是却也能够达到与LSTM相当的功能。考虑到硬件的计算能力和时间成本，因而很多时候我们也就会选择更加”实用“的GRU。