用 Pointer Network 网络模型给数字排序

Linyang He

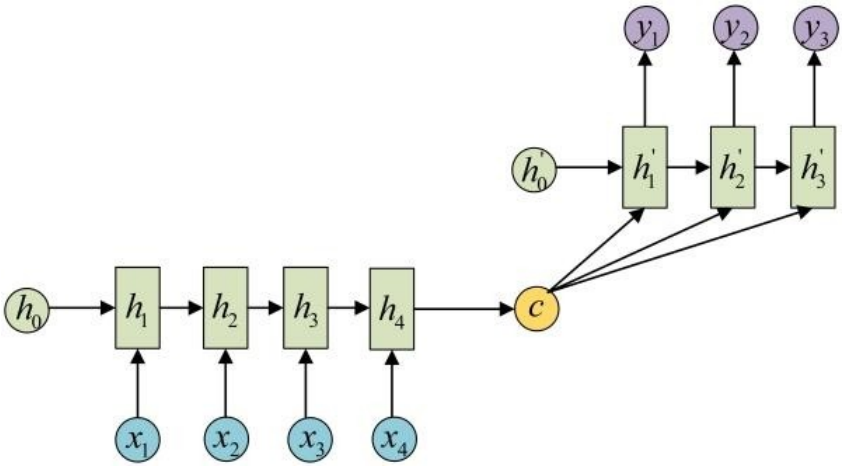
2018 年 12 月 9 日

本文以为数字（包括整数、浮点数）情景为例，重点关注了 Pointer Network 模型作为一个 attention 机制特例的应用。本文先会讲述 attention 模型，再描述了 Pointer Ntework。接着展示在长度分别为 5、10 的整数和长度为 5 的浮点数上排序结果的展示。最后是实验的总结，并提出了一些自己的思考。

1 Background

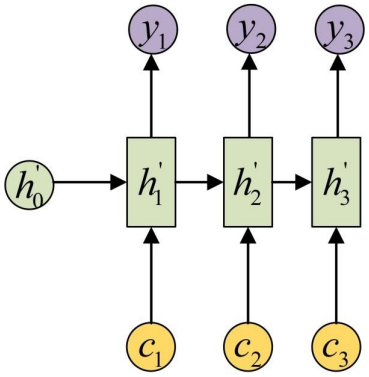
1.1 Attention 机制

基于 Attention 机制（注意力机制）神经网络是最近神经网络研究的一个热点。通常而言，Attention 模型利用在 seq2seq 上。



最基本的 seq2seq 模型包含一个 encoder 和一个 decoder，一般的做法是将一个输入的句子编码成一个固定大小的 state，然后作为 decoder 的初

始状态（当然也可以作为每一时刻的输入），但这样的一个状态对于 decoder 中的所有时刻都是一样的。Attention 即为注意力，人脑在对于的不同部分的注意力是不同的。没有 attention 机制的 encoder-decoder 结构通常把 encoder 的最后一个状态作为 decoder 的输入（可能作为初始化，也可能作为每一时刻的输入），但是 encoder 的 state 毕竟是有限的，存储不了太多的信息，对于 decoder 过程，每一个步骤都和之前的输入都没有关系了，只与这个传入的 state 有关。attention 机制的引入之后，decoder 根据时刻的不同，让每一时刻的输入都有所不同。对于不同的 decoder 阶段，输入的 hidden_state 也发生了变化，而不是仅仅一个 c 作为 decoder 的输入了。



Attention-based Model 其实就是一个相似性的度量，当前的输入与目标状态越相似，那么在当前的输入的权重就会越大，说明当前的输出越依赖于当前的输入。

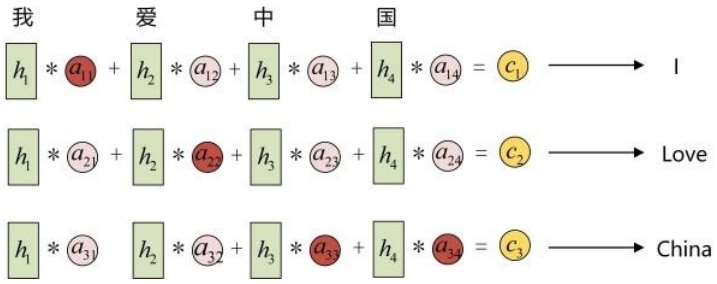
假设我们对 encoder 端的 state 标注为：$(h_1, ..., h_{T_A})$，而 decoder 端的 state 标注为：$(d_1, ..., d_{T_B})$, 那么为了计算在 decoder 端第 t 个时刻的 attention 向量, 我们可以根据如下公式：

$$s_i^t = v^T tanh(W_1 h_i + W_2 d_t)$$

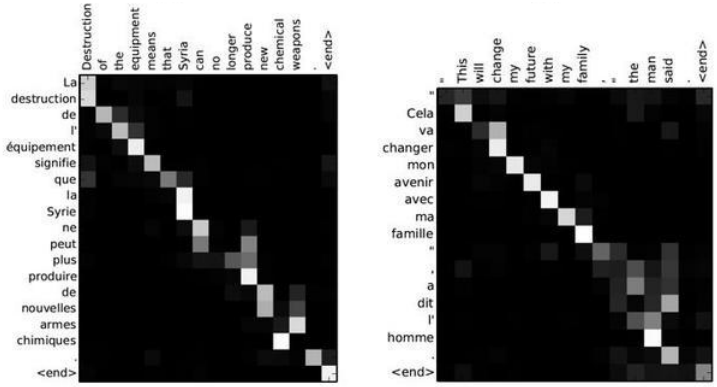
$$a_i^t = softmax(s_i^t)$$

$$c_t = \sum_{i=1}^{T_A} a_i^t h_i$$

这里的 W_1,W_2, v^T 都是通过网络学习得来的参数。训练好这个含有 attention 机制的 seq2seq 网络之后，如果 a_i^T 越高，他们对相应的注意力就应该越高。



如图，以机器翻译（Neural Translation Machine) 为例，attention 值被可视化了，如果越接近白色，则说明 decoder 端另外一门语言的某些词语和 encoder 端源语言的对应的词越被“注意”。这说明了 attention 机制，确确实实起到了特别关注序列中某个对象。



1.2 Pointer Network 与数字排序任务

考虑到普通 attention 机制的 seq2seq 不能处理变长的问题，且如果出现一个词典中不存在的词，普通 attention 也无能为力解决。这时候就需要一个可以处理变长和处理词典中不存在的 token 的网络，Pointer Network 就诞生了。Pointer Network 实际上是一种特殊的注意力机制。我们知道，事实上，注意力机制可以分为两步：一是计算注意力分布 α，二是根据 α 来计算输入信息的加权平均。我们可以只利用注意力机制中的第一步，将注意力分布作为一个软性的指针（pointer）来指出相关信息的位置。**直观而言，是把普通的 attention model 中的加权平均变成了取最大值**。针网络（Pointer Network）[Vinyals et al., 2015] 是一种序列到序列模型，输入是长度为 n 的向量序列 $X = x_1, \cdots, x_n$ ，输出是下标序列 $c_{1:m} = c_1, c_2, \cdots, c_m \; c_i \in [1, n]$, i。

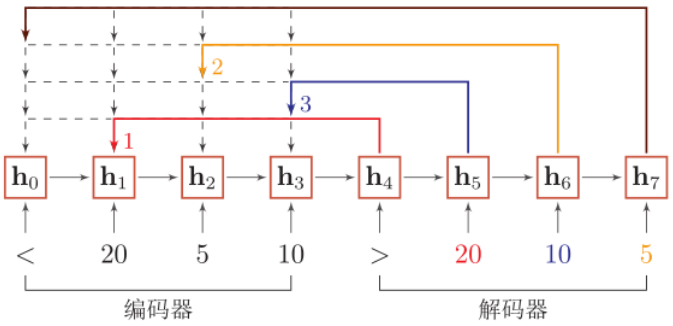
和一般的序列到序列任务不同，这里的输出序列是输入序列的下标（索引）。数字排序任务是指，输入一组乱序的数字，要按照规定的某个顺序进行对下标的排序，比如输入为 20,5,10，输出为 1,3,2。这里的数字可以是整数，也可以是浮点数。本文也正是探讨这样的应用。条件概率可以写成

$$p(c_{1:m}|X1:n) \approx \Pi_{i=1}^{m} p(c_i|X_{c_1}, .., X_{c_{i-1}}, X_{1:n})$$

公式中的条件概率可以看成注意力分布来计算。假设用一个循环神经网络对 $x_{c1}, \cdots, x_{c_{i-1}}, x_{1:n}$ 进行编码得到向量 h_i ，则

$$p(c_i|c_{1:i-1}, x_{1:n}) = softmax(s_i, j)$$

$s_{i,j}$ 其实就是上文普通 attention 机制中的 s_i^t。



如图，我们通过左边的编码器得到了 h3 作为 hidden_state 输入到解码器中。此时，我们在根据 attention 分布指向了概率最大的那个位置。而我们的数据集是已知这个位置信息的，这样通过位置的概率分布和真实的位置信息，通过计算 CrossEntropy 就可以得到 loss 并实现反向传播了。

2 Experiment

笔者在分析了提供的样例代码之后，认为样例代码的数据处理操作并不符合一般的流程规范。比如，为何在一个 epoch 里面才调用生成数据，这样不就意味着，每一个 epoch 都只有一个 batch 么，这相当于在当前迭代中遍历整个训练集，这样的操作就失去了 mini-batch 梯度下降的可能性，代码的泛化能力较差，更何况我们知道通常情况下，效果最好的就是 mini batch。为了解决这个问题，让 mini-batch 算法能够真正被使用，笔者并未采用样例代码的框架。

2.1 Dataset

可以看到，样例代码中已经提供了生成数据的函数，只需要调用即可。不同的是，笔者选择在训练前就构造好数据集。

2.2 Model

pointer work 基本上分为了三个部分。一个 encoder，一个 decoder，和一个用来 attention 的部分。

2.2.1 Encoder

这部分并不受其他部分影响。于是我们可以用一个普通的 LSTM 或者 GRU 网络作为 encoder 即可。注意 LSTM 和 GRU 的输出长度不同。LSTM 有两个 hidden_state 门控输出。

2.2.2 Decoder

因为我们知道 decoder 端受到 encoder 端每个时序上的输入影响，因此，我们使用 LSTMCell 或者 GRUCell 在 for 循环中构造网络，而不是直接用一个 LSTM 或者 GRU 网络。

2.2.3 Attention

Attention 关注的是在 1.2 中公式的实现。我们可以用线性层来实现。其实这相当于在当前时刻的 decoder 的 hidden_state 和全面全部时刻 encoder 的 hidden_state 上面又添加了个全连接层，并通过 softmax 归一化之后，得到了概率分布。这个概率分布可以和标注好的 Y 进行 crossentropy 计算。

2.3 Training

定义超参数如下，

超参数	值	解释
EPOCH	400	训练多少代
Learning_rate	0.001	学习率
total_size	10000	数据集大小
batch_size	250	batch 大小为 250，一个 epoch 有 40 个 batch
input_size	1	输入数据的维度
weight_size	256	attention 中用得到的全连接层的节点个数
hidden_size	512	RNN 中的隐藏层的结点个数

模型采用 AdaM 优化，crossentropy 作为 loss function。

2.4 Test

模型训练好后，另取数据进行测试。

3 Result

本次实验中，我们训练了三个模型，情况如下，其中 trraning_loss 是最后一次迭代的值。正整数排序：

Task	**Int, 5 bit**		**Int, 10 bit**	
Model	LSTM	GRU	LSTM	GRU
Train Loss	0.00136	0.00020	0.01336	0.00007
Train Acc	99.74%	99.94%	98.48%	99.36%
Test Acc	96.70%	**97.20%**	**91.30%**	75.50%

浮点数排序：

Task	**Float, 5 bit**		**Float, 10 bit**	
Model	LSTM	GRU	LSTM	GRU
Train Loss	0.00078	0.00118	0.00872	0.00753
Train Acc	99.60%	99.66%	99.51%	99.81%
Test Acc	87.10%	**91.80%**	53.20%	**64.40%**

结果分析：

1. 我们发现，对于正整数而言，随着要求排序的序列长度变长，GRU 的稳定性下滑很严重。这更加说明了 LSTM 确实可以解决长短依赖问题。

2. 在排序序列比较短的时候，GRU 的表现更优异。考虑到 GRU 的门控相较 LSTM 要少一个，在序列较短的时候，处理的效能会更高。

3. 对于浮点数排序，当序列长的时候，GRU 和 LSTM 的表现都比较糟糕。说明 pointer network 算法本身更适合处理离散型的数据。

4. 总体而言，在不知道数据类型的时候，短序列用 GRU，长序列用 LSTM 是不错的选择。

4 Others

对数字排序生成这个任务的一些思考：

1. 我们发现，在训练和测试的时候，实际采用的都是同样长度的数字序列。但是我们知道 seq2seq model 是可以处理变长问题的，如果测试的时候采用的是更长的数字序列，结果又会如何呢。

2. 实际上，普通的 attention model 已经可以对定长度的整数序列排序很好的 work，但是如果是浮点数，我们会发现，不可能让测试集中的每一个数据都出在训练集中，这种情况下，普通 attention 模型就失效了。从这种角度而言，pointer network 有点像一个复制粘贴的网络，根据需要把输入中的元素挑选出来按照一定顺序排列。而这也是为什么 pointer network 适合用在文本摘要的任务上。把一段文字的一些词语挑选出来拼起来就可以成为其摘要。