**翻译：从单词到音标**

**1. 项目简介**

目标：给定一个单词的拼写，给出这个单词的音标。

核心算法：sequence to sequence。

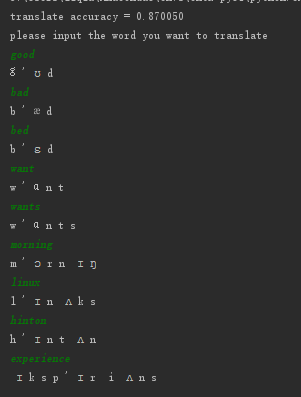
操作系统：windows 10

编程语言：python

深度学习框架：Microsoft CNTK

应用前景：有很多专有名词，不能在词典中找到对应的音标。比如DAPPA（美国国防部高级研究计划局），这时你就可以求助于我的翻译软件，它可以告诉你DAPPA的音标应该是d ' æ p ʌ。这样你就避免了念D~A~P~P~A~的窘境。

效果演示：



**2. Sequence to Sequence 算法**

Sequence to sequence 是谷歌在2014年提出的一个基于RNN的算法，可以建立从一个序列到另一个序列的映射，这两个序列都是可变长度序列。其核心思想在于使用两个RNN（LSTM），一个作为encoder，将序列A编码到一个定长向量，一个作为decoder，将定长向量解码成目的序列。

该算法建立序列x1, x2, …, xt 到序列y1, y2, …, yt’的映射，这两者的条件概率用下式得到



其中v是x1, x2, …, xt编码成的向量。有了条件概率，我们就可以有序列x1, x2, … xt 得到与之最匹配的y1, y2, …, yt’

算法细节：

1. 将序列X(x1, x2, …, xt)逆序输入到第一个LSTM中，将LSTM的最后一个输出E作为序列X的编码。然后让E作为第二个LSTM的初始状态。

2. 训练时，将 y1, y2,…, yt’-1依次输入到第二个LSTM中，要求LSTM的输出序列是y2, y3, y4,…,yt’。yt’必须是一个特定的终止符。y1必须是一个特定的起始符。

3. 测试时，将起始符作为LSTM的第一个输入，然后将LSTM的输出作为LSTM的下一个输入，直到LSTM输出终止符。LSTM的输出即为翻译结果。

**3. Sequence to Sequence 算法的实现**

1）框架选取

Microsoft CNTK是一个类似于Google Tensorflow的基于计算图模型的深度学习框架，用户使用python和给定的API，描述一个计算图，框架对用户描述的计算图自动求微分，通过随机梯度下降算法训练。

2）LSTM

我使用的是双层，带有Stabilizer([Self-stabilized deep neural network (P. Ghahremani, J. Droppo](http://ieeexplore.ieee.org/document/7472719/))的LSTM。多层LSTM相较于单层LSTM拥有更强大的表征能力。Stabilizer则可以使神经网络对学习率不敏感。一般的神经网络对于反向传播的学习率高度敏感，设置大了会导致训练不稳定，得到较差的训练结果，设置小了又会造成收敛过慢，学习率还应该随着训练的进行动态改变，使调参过程非常耗时耗力。而Stabilizer可以允许你使用一个全局不变的，没有仔细调节过的学习率，并达到和仔细调节后的学习率相同甚至更好的结果。

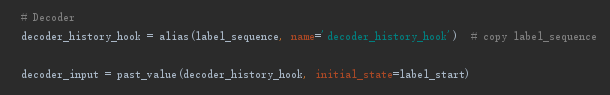
3）字符编码与解码

我将训练数据中的字母与音标编码为one hot值。解码神经网络输出时，只需要将输出向量的最大值置为1，其余置为0，然后解码这个 one hot值，即可知道神经网络的输出是哪个字符。

4）Sequence to Sequence 算法的实现难点

Sequence to Sequence算法实现的主要难点是解码时训练和测试的输入数据不同，训练时，输入数据是一个已知的序列，即目标序列；测试时，输入数据是一个未知的序列，需要动态产生，且长度无限。我需要描述一个计算图，可以使用同一个神经网络，处理两种不同的情况。

我采用的计算图的输入有两个，一个是输入序列X，一个是训练时要用的标准输出序列Y（测试时也要提供Y，不过那是Y是个只提供长度信息的无意义序列）。第二个LSTM的输入序列为Z。Z有可能是y1,y2,y3,…,yt’-1（训练时），也有可能是y1加上第二个LSTM的输出序列（测试时，此时Z的长度为无限长）。在计算图中嵌入终止条件的判断逻辑是一件不容易的事情。为了方便，将Z裁减到t’-1。这样在训练时，Z没有被裁减，测试时，当输出序列长为t’-1时第二个LSTM停止输出。测试时，拿到输出序列后，在输出序列中找到终止符，并删去终止符后面的内容，才是真正的翻译内容，因此测试时提供的Y应该比较长，确保有意义的输出不被裁减。



这一段代码表示在训练网络中，将y1,y2,…,yt’-1（label\_sequence）作为解码器的输入。



这一段代码表示，克隆训练网络，并用网络输出替代被克隆网络中的decoder\_history\_hook(即label\_sequence)，将克隆的网络作为测试网络。

**4. 数据集**

1）数据集简介

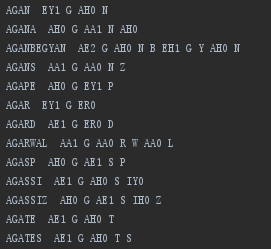
我使用的训练数据集是CMU Pronouncing Dictionary。这个数据集是一个开源的美音发音词典。

这个数据集为了避免编码问题，使用了ARPAbet symbol set 来表示音标。就是用英文字符的组合来编码本为罗马字符的音标，部分编码如下所示

'AO': 'ɔ', 'AA': 'ɑ','IY': 'i','UW': 'u','EH': 'ɛ','IH': 'ɪ',

这个数据集用0,1,2来表示重读与否。0是不重读，1是主重读，2是从重读。在我的翻译结果中，'表示主重读，. 表示从重读。

数据集中部分内容如下图所示，每行第一个是单词，其余为其音标。



2）数据集中信息偏向的影响

事实上，美国不同地区的发音，以及对单词的音标是不一样的。在CMU Pronouncing Dictionary，ə常常用ʌ来替代。受数据集影响，我的翻译程序中也常常用ʌ替代ə

比如对于linux的翻译中，必应词典给出的音标是l ' ɪ n ə k s，而我的翻译程序给出的音标是l ' ɪ n ʌ k s

**5. 翻译失败的情况与分析**

1）翻译失败的典型案例

Afternoon

翻译：' æ f t ɝ n . u n

标准答案：.æ f t ɝ n ' u n

错误：重音符号标错。

Mismatch

翻译：m ' ɪ s m ʌ tʃ

标准答案：（1）m ' ɪ s m ʌ tʃ， （2）m ɪ s m 'ʌ tʃ

Resume

翻译：r i s ' u m

标准答案：’rezu.mei

2）案例1分析

Afternoon的重音应该在前，这是因为名词的重音通常在前，动词重音通常在后。Afternoon对于我训练的神经网络来说只是一串字符序列，它不知道这是个名词，故不能利用名词的重音在前的这一信息。

3）案例2分析

案例二中，mismatch既可以做名词，也可以做动词，故有两个发音，一个重音在前， 一个重音在后。而我的神经网络只能给出一个输出，故不可能在任何时候都给出mismatch正确的发音。

4）案例3分析

案例三中，神经网络输出了一个英语初学者在读resume时常犯的错误。一定程度上，这也不能算是神经网络的能力不足，它确实给出了resume这个字符序列最可能的发音。可惜resume是个外来词汇，发音和拼写不一致。

5）总结

这个从单词到音标的翻译程序达到了87%的准确率。它不能达到100%的准确率的原因是，一个单词的发音不止和其拼写有关，还与其词性，词源等其他属性有关，更糟糕的是，一个单词还可以有多种发音，这是只能输出一种发音的翻译难以处理的。

我的程序在这个项目中遇到的问题，也是NLP中常见的两个问题，多义和外部信息问题。

**6. 代码**

代码文件一共有3个，nn.py, prepare.py和sequence2sequence.py。

nn.py是Microsoft CNTK中自带的python文件，其中定义了我用到的带有Stabilizer的LSTM

prepare.py是我写的预处理程序，里面实现了对CMU Pronouncing Dictionary数据集的预处理函数，并定义了ARPAbet symbol到音标的映射。

Sequence2sequence.py是主程序，里面有sequence to sequence算法的实现，模型的训练，测试，使用。

在实现sequence to sequence算法时，借鉴了Microsoft CNTK Example: Sequence to Sequence 项目中对于sequence to sequence算法的实现。