赵哲焕 Keras 概述

新闻来源：IR实验室       发布时间：2016/6/29 21:33:27

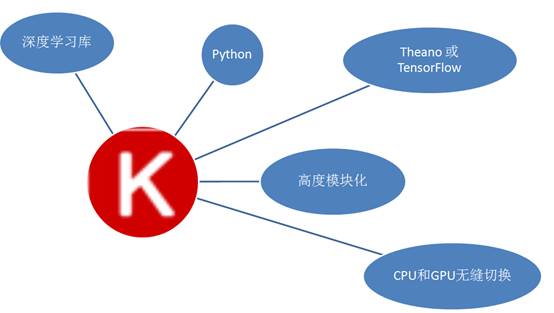
学计算机的可能都有一种强迫症，很多东西其实有很好的工具可以用，但是如果他能自己实现，都会喜欢自己写程序实现它，虽然这样要费很多时间和精力。我也是这样的一个人。

开始学习深度学习的时候，需要用到GPU编程，当时已经有Theano框架了，但是还是想自己写一下CUDA代码，而自己学习能力又有限，所以花了很长时间，用numbapro写了python的GPU代码。

写了几个模型之后，觉得好辛苦，就尝试了一下Theano，觉得也差不了多少，还是需要学习一段时间，虽然比自己写是好多了。前一阵又接触到了Keras深度学习库，用了之后发现自己之前好蠢。用Keras实现深度学习模型，又快代码质量又高。

所以做一下Keras的概述，也相当于自己的一点点总结吧。

首先，Keras 是什么？我们用下面一张图就可以很好的解释这个问题。



Keras是用python实现的深度学习库。它是在Theano或TensorFlow（可以选择）基础上进一步封装得到的库，所以是高度模块化的。用起来非常之方便，就像搭积木一样。而且，它可以在CPU和GPU之间无缝切换。

那现在，我们用一个简单的例子来说明一下Keras是怎么工作的。

现在，我们想用Keras实现如下多层感知机模型。输入层是20个节点，加上两个64个节点的隐藏层，输出层大小是20。隐层的激活函数用的是tanh

在这里把这部分代码根据它的具体功能分成了6块。

第一块，是声明我们用简单的，也是最常用的Sequential方式搭建我们的网络接口。一般的网络结构用Sequential都可以搭建，它就是通过简单的添加一层一层的网络实现的，下面会讲到。

第二块，是构造具体的网络结构。因为我们构建的是简单的多层感知机，所以用最基本的Dense层（普通的全连接层）就可以了。

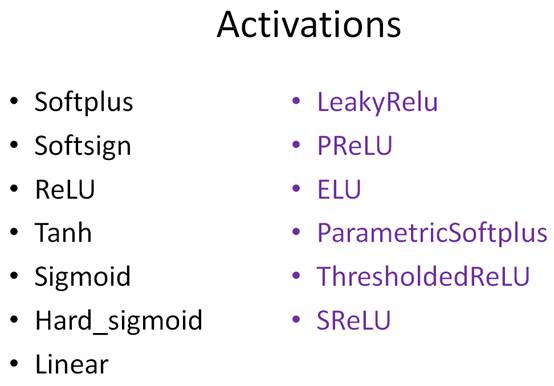
http://ir.dlut.edu.cn/Uploads/ue/image/20160629/6360283283309037218977685.jpg

这是第一个隐含层，用的是Dense 层，这一层的输入维度是20，输出维度是64，这一层参数初始化方式是用均匀分布生成。一般Sequential模型中添加的第一层需要确定输入的维度（input\_dim或者input\_shape）。

http://ir.dlut.edu.cn/Uploads/ue/image/20160629/6360283283355837294433108.jpg

这两还代码分别说明了这一层使用的激活函数是tanh，还有这一层使用Dropout机制。

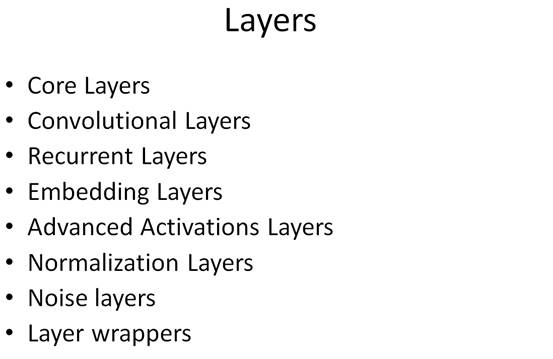
可以使用的激活函数非常丰富，具体如下：



左侧是一些常用的激活函数，而右侧紫色的是经过改进升级的激活函数方法，都可以随便拿过来用。

下面的代码类似，只不过是不用再确定input\_dim这个参数了，只要给出输出节点个数就可以，因为它们会把上一层的输出作为当前层的输入。

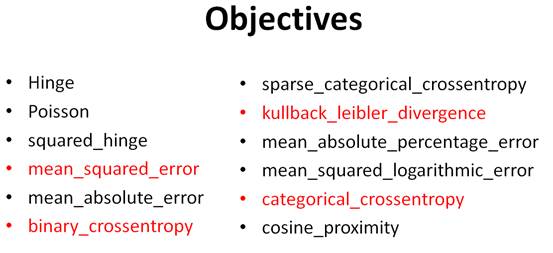
另外，在构建网络结构的时候，以用到的网络层还有很多，下面只是列出来网络层的几大类别，每个类别下面还可以细分好多种，一会儿会接着介绍几种。



第三块，是声明的学习当前网络使用的优化方法，用的是随机梯度下降（SGD）方法。可以使用的优化方法还有很多当前流行的方法，具体如下：



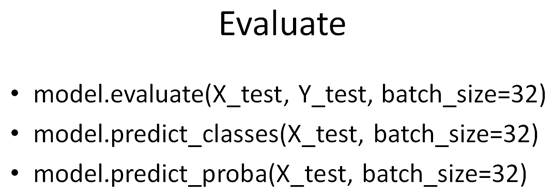
第四块，是对上面的网络结构进行编译的过程，这个过程中需要设置损失函数类型，本文使用的是多类别的交叉熵作为损失函数，Keras提供的损失函数还有很多，具体如下：



其中，用红色标出来的是我们比较常用的损失函数，如，平方差，KL距离，等等

第五块，是拟合的过程，输入数据集，就可以进行模型的学习。

第六块，是用训练好的模型对测试集数据进行测试。可以用evaluate直接计算accuracy值，也可以用predict\_classes或predict\_proba函数分别计算出每个实例的类别或者每个实例对所有类别对应的得分概率值。具体如下：

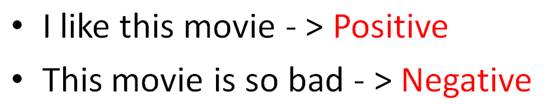


六个模块都讲完了，其实用Keras做事情就是这么简单。下面我们稍微深入的理解一下Keras，对我们常用的一些网络层（如，CNN，RNN，等等）

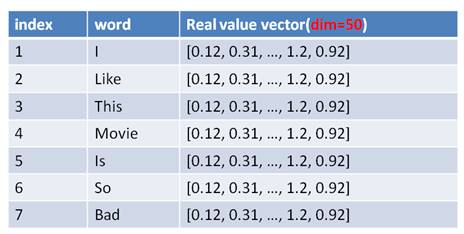
如下面所示，Keras提供了三种维度的CNN也提供了LSTM，GRU等当前流行的RNN系列模型。当然Embedding层是做文本领域必不可少的东西。下面会利用具体的例子，结合下面标红的网络给大家分享一下Keras的建模过程。



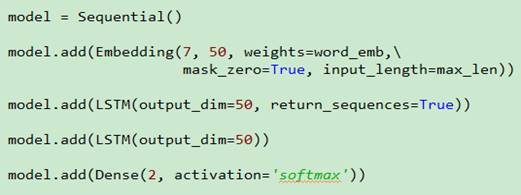
现在我们打算用LSTM做句子的情感分类问题。假设训练集中只有下面两个实例



基于训练集，我们可以利用word2vec等工具训练出对应的词向量。如下表，每个词对应的一个50维的实数向量，同时每个词对应一个整数的索引号。这个索引号在Keras的Embedding层是非常有用的。因为我们查找词向量表的时候，输入的不是一个词，而是词对应的索引值。这个应该是为了技术实现上的方便和效率。



有了训练好的词向量，就可以构建我们想要的模型了。



在上面的代码中先讲一下Embedding层。

  其中7代表的是词向量表的大小（词汇量）。

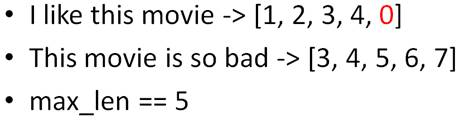
  50代表的是每个词对应的词向量的维度。

  weights=[word\_emb] 参数用来初始化词向量表，如果没有对这个参数赋值，说明词向量是随机初始化的。（代码中拉掉了一对中括号）

  mask\_zero=True是为了解决句子的不同长度带了的可并行的问题。这里我们可以统一一下句子的长度，对于较短的句子，可以用0来填充。这样，mask\_zero=True之后，对于0值就会忽略计算，但是因为它对齐了句子的长度，所以也可以完成并行计算。

  Input\_length=max\_len，确定的是输入的长度，可以看出需要输入固定长度的词对应的索引的整数序列。

回到问题本身，可以得到如下输入数据：



因为最长的句子为5个词，所以输入可以固定成5。两个句子的词对应转换成索引值，然后长度不到max\_len就用0来填充。

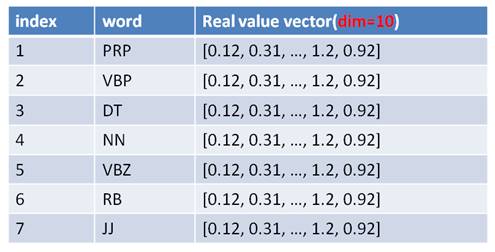
在上面的两个LSTM层中，如果需要叠加多层LSTM，那上一层的LSTM必须设置，return\_sequence=True。因为LSTM的输入是个序列数据，如果没有设置return\_sequence=True，返回的值是序列中最后一个元素对应的输出。

       最后一层是输出层，因为需要判断正负两种极性，所以输出节点为2。

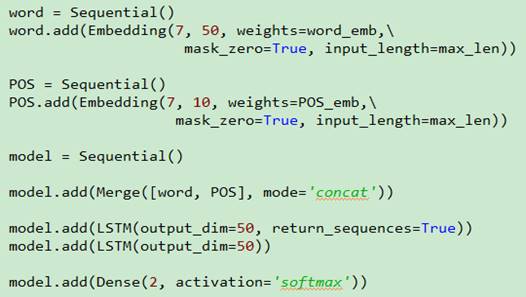
       我们平时还经常需要扩展一些特征，这是可以通过Merge层完成。假设如下所示，我们打算把每个词的词性信息也作为输入。



这时，我们同样通过对每个词性赋予一个实数向量（一般是随机初始化），如下表所示，跟词向量表很像。

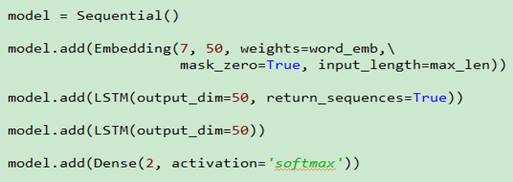


那现在的代码就可以写成如下形式：

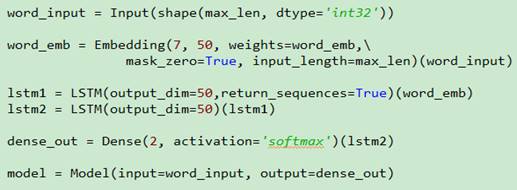


可以看出，Sequential可以嵌套在另一个Sequential里面，word和POS分别是两种输入形式，然后通过Merge把两种输入形式进行合并，之后输入给LSTM中。

好，如果你觉得模型还不够复杂，我想要更复杂的模型，那么可以用泛型接口模型来实现。



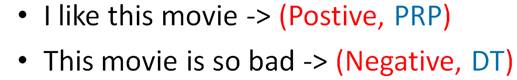
这个是上面提到过的代码块



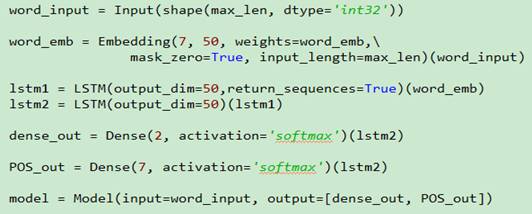
这个是对应的泛型编程思想实现的。我们可以看出，泛型接口方法其实就是把每一层看做是一个函数，函数的输入是上一层的函数的输出，这样一层套一层。那么什么样的复杂模型需要泛型接口方法去实现呢？就是有多个输出的时候。前面扩展特征属于多个输入，当然也可以用泛型接口方法实现，也可以用我们用的Sequential嵌套的方式实现。而当你需要多个输出的时候，就只能用泛型方法了。

那什么时候需要多个输出呢？首先给你们讲一个故事。据说，一个母亲不是到时报社编辑还是什么，正在对一张撕碎了的报纸进行还原。但是因为文字都差不多，一时半会很难还原回去。当她累的休息片刻的时候，她的小女儿却用几分钟就帮她拼回去了。她非常惊讶，所以问女儿是怎么做到的。她女儿说：“这个很简单啊，因为文字的反面是一个大大的人的照片，我只要看着那张照片去还原这张报纸，反面的人拼好了，那正面的文字当然也就拼好了啊”。说这个故事是为了说明，我们要达到某个目的的时候，也可以从另一个角度或者维度去思考这个问题，或者把这个维度的分析也添加进来，一起分析，问题就会变得简单的多。

那再次回到我们的情感分类问题上，我们的目的是对每个句子划分是积极还是消极，那对这个问题有没有能找到那个“背面的人”呢？很不好意思，我还没有找到。但是为了继续讲泛型接口方法的使用，我假设一个句子的第一个词的词性就是那个“背后的人”，这样这个模型需要同时输出两种结果，喜欢在拟合第二个输出的过程，对第一个输出的拟合有促进作用



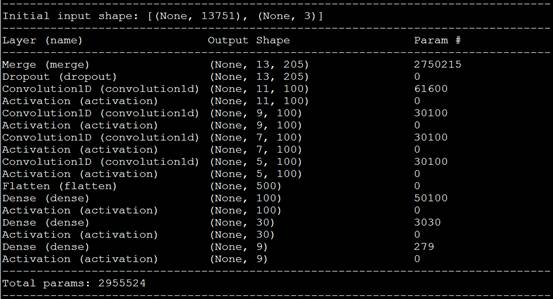
那代码就可以写成下面的形式。



POS\_out就是新添加的输出，lstm2同时做了dense\_out和POS\_out的输入。

Keras的核心功能基本就介绍差不多了。下面再介绍几个简单实用而且容易被忽视的小功能。

Model的summary函数可以打印出你设计的详细网络结构。有时真正的结构不是你想要的，但是你还不知道，所以需要用这个函数打印出来，看看网络结构是不是你想要的。它也会把所有的参数个数计算出来，告诉你模型的复杂程度。下面就是一个例子。其中None这个维度代表mini\_batch的大小。



每一层都有trainable参数，默认是True，如果设成False这一层参数在拟合的过程中不会得到更新。当我们需要固定词向量的时候，就可以通过设置这个参数实现。

对于LSTM/GRU这两个模型，有一个consum\_less参数，分别可以设成gpu，cpu，mem。

  consume\_less=gpu，把所有的参数矩阵合并成一个，高并发。所以在gpu上跑的时候可以用这个参数

  consume\_less=cpu，把大的矩阵相乘分成少量但是大的矩阵相乘，所以速度快但是耗内存。

  consume\_less=mem，把大的矩阵相乘分成多个但是小的矩阵相乘，所以速度慢但是省内存。