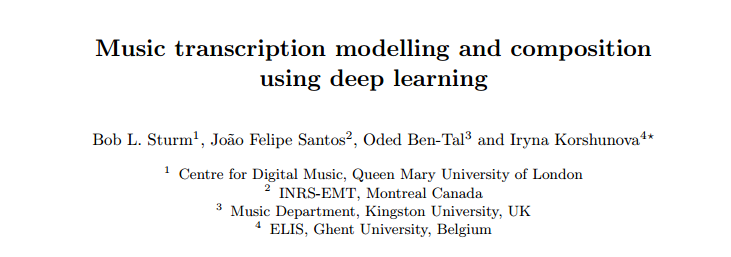
摘要

本周周报的主要内容是对阅读的一篇论文进行了总结，我会从概述，模型选用的数据，如何构建和训练模型，模型的输出结果等方面介绍这个模型。同时在看了一遍melody\_rnn项目每个.py文件的源代码后，我参照代码中的部分注释总结了每个.py文件的主要功能。另外进行了tensorflow的进一步学习，学习了简单的神经网络过程可视化方法，了解了**Optimizer优化器**相关知识。

## 阅读论文总结

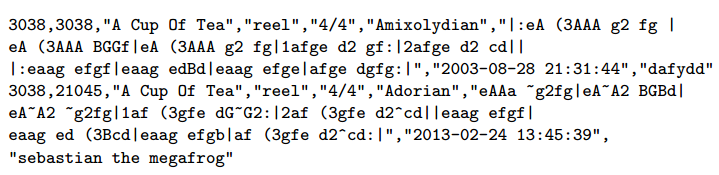


论文概述

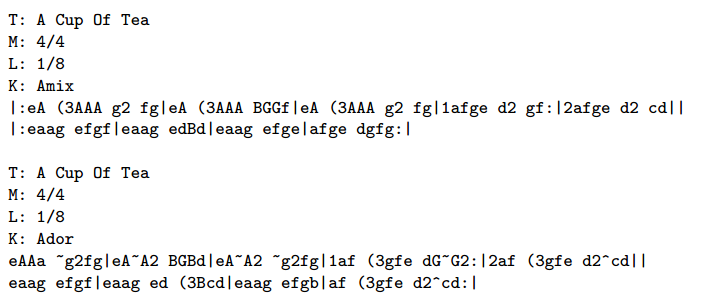
如题，这篇论文主要介绍了如何使用LSTM进行音乐转录建模和合成，使其在音乐组合的特定语境中创造出有用的音乐转录模型。作者建立和训练LSTM网络，并使用它们来生成新的转录。此网络有3个隐藏层，每层包含512个LSTM模块，并使用大约23000个文本词汇(使用ABC记谱法)来训练它们。并采用构建了两种训练模型：一种类型是char-rnn，它在单个字符的词汇表上运行，并在一个连续文本文件上进行训练；另一种类型是folk-rnn，运行在一个词汇的转录token上，并且训练在单个完整的转录上。**每个网络的输出是一个关于其词汇表的概率分布。**

选用数据

对于char-rnn和folk-rnn两个模型采用的数据如下，首先是char-rnn的原始数据（示例两条）：

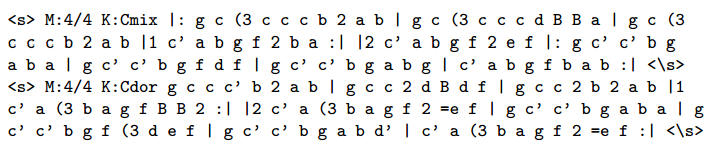


每条数据都是从两个标识符开始，然后是标题、曲调类型、表、键、ABC编码、日期和贡献用户。每个属性的贡献各不相同，其中有些是相当复杂的。比如指定装饰，装饰音，污痕和和弦等。总之原始数据显得有些杂乱无章。为此，作者只保留五个ABC字段(标题、表、键、单元注释长度和抄写)，并分别用空行分隔每个贡献。上述两条示例数据变成：



这样一来原始数据变成了一个包含13515723个字符的文本文件，共有135种独特的字符，它们每一个都成为char-rnn词汇表的元素模型。

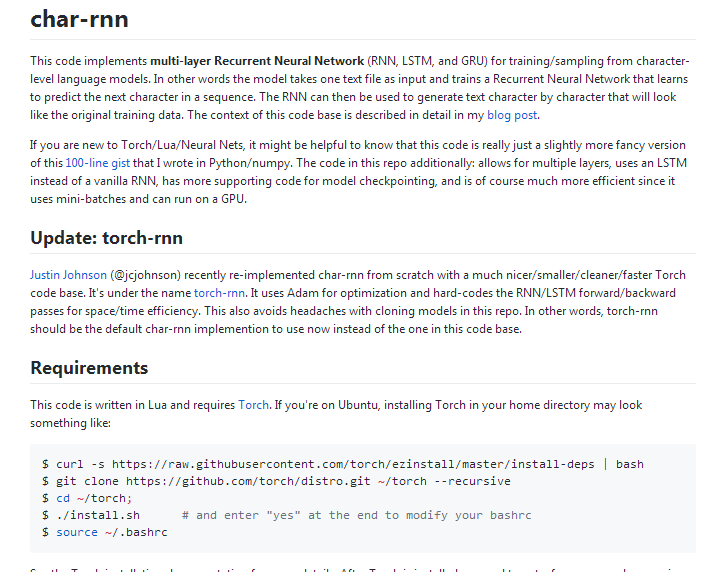
接下来为folk - rnn模型训练创建数据。作者对原始数据进行了删除冗余，清洗等操作之后使得输入的数据变成下面的样子。



至此，folk – rnn模型的数据集包含有4056459个记号，其中2816498个是音高，602673个是持续时间，520290个是度量。这些记号有137种，并且每个记号都成为了模型的词汇表元素。

训练模型

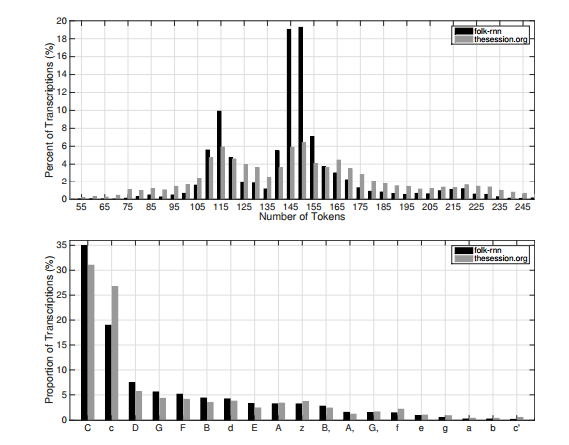
使用“char-rnn”实现构建和训练char-rnn模型（关于charrnn可参考<https://github.com/karpathy/char-rnn>），如图：



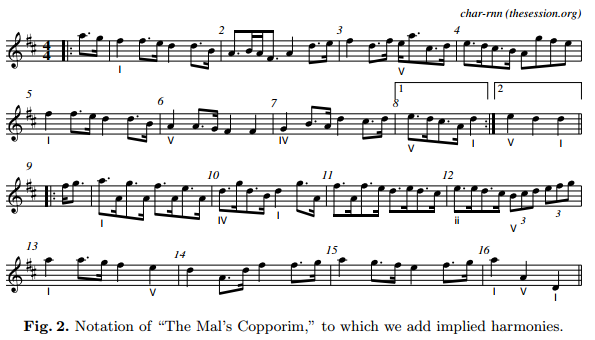
这个使用的是RMSProp算法（此算法属于优化算法的一种，在tensorflow中也有对应的RMSPropOptimizer类），使用的是50个样品的小批量，每一个batch都包含50个字符，并采用一个渐变的裁剪策略来避免梯度爆炸。对于详细的训练过程，诸如初始learning rate的设置，drop rate的设置等等，由于篇幅较长，此处就不粘贴翻译了，可参考原文第7页。

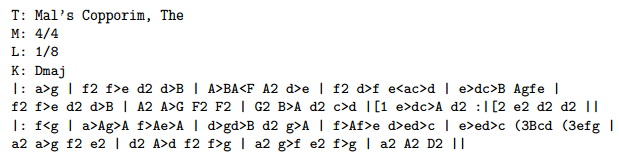
模型输出结果

训练好模型之后，作者使用folk - rnn系统生成了6101个完整的文本。表和模态的比例接近于训练数据集的比例。如图一显示了特定标记长度的转录比例，以及以特定音调结束的比例。最终的音高分布似乎是在两者之间，但不是转录令牌长度。通过结果可以发现(通过观察重复符号的出现)，大约68%的folk - rnn 转录使用度量令牌创建了一个结构AABB，每个部分有8条长；而在训练数据中，54%的文字都有这种结构，这种结构在爱尔兰民间音乐中很常见。



charrnn模型生成了72376个调音标，并自动合成了35,809个。我们用这些结果创造了“无止尽的传统音乐会话”，15个循环通过集合，随机选择7个抄录，每5分钟。。在竖琴上打开这一件还很有趣。下面是我们的charrnn模型的精确输出：





这幅作品展示了一个很常见的传统爱尔兰音乐的结构：一个重复的8小节“曲调”，接着是一个重复8小节的“转向”。。对于结果，作者的后续工作又可以分为三个方面：（1）在种群层面上，比较了训练转录和生成转录的描述性统计；（2）在单独的层面上，研究生成的转录如何反映训练转录中的音乐实践习惯；（3）在应用层次上，使用该系统来生成音乐组合中的创意。

此外，文中还介绍一些其他基于RNN和LSTM的音乐建模和生成的实例（此处选三个简单介绍）：

①Todd

由Todd构建和测试的RNN网络模型由一个含有19个单元的输入层，一个包含8 - 15个单元的隐藏层和一个包含15个单元的输出层组成。每个输入和输出层的一个单元是“注释开始”状态；其他14个单位代表音高，每一个单位从D4到C6。其他四个输入单元识别一个特定的单声道训练旋律，其中有四个，每一个34个音符长。Todd将时间划分开来，使模型的每一个时间步骤都代表一个八音符的持续时间。（这个实例与我上次的报告中提到的melody运作方式类似）。

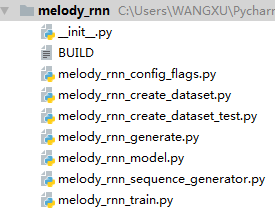
②Mozer

Mozer构建RNN模型并使用**分布式**的音乐编码方法生成旋律。这些系统生成的输出是在note（音符）级别而不是统一的时间步骤级别。每个音高都是根据其基础的频率，半音阶以及在五分之一圈的位置来编码。音符的持续时间用类似的方法进行编码。弦伴奏是根据音高来编码的。一些输入单元表示时间签名、键和下拍。Mozer的RNN模型使用一个带有O(10)单元（O(10)的含义没有搜到..）的隐藏层。训练材料包括人造序列，巴赫的10首旋律(长达190个音符)以及25个欧洲民谣旋律等。Mozer发现这些系统可以成功用于模拟旋律的局部特征，但不能捕捉更长的结构。

③Eck和Schmid - huber

Eck和Schmid – huber首次将LSTM网络应用于音乐建模和生成。与Todd相似，他们采用本地音乐编码方法，13个单位代表13个音高(半音阶)，并使用最小持续时间间隔时间。十六分音符。他们还使用12个输入单元来指定音高。隐藏层由8个LSTM细胞的两个块组成，其中一个块用于旋律，另一个块用于协调。他们将旋律块与和谐方块联系在一起，而不是相反。他们用6分钟的12小节布鲁斯旋律来训练系统，和弦伴奏，每小节8个时间编码。每一次训练都是96次。与Mozer的结果相比，Eck和Schmid - huber发现LSTM网络展现了一种可以长期建模和复制这种风格的能力。

## melody\_rnn项目中每个类实现的功能



**melody\_rnn\_config\_flags.py**

为Melody RNN模型配置提供类、默认值和实用工具，其中定义了构建模型和训练模型要用到的标签以及参数；

**melody\_rnn\_create\_dataset.py**

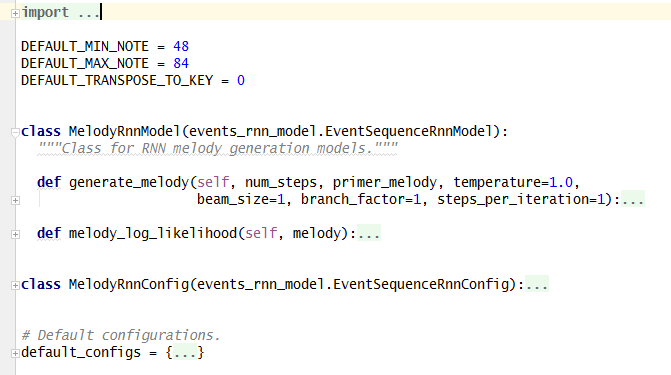
从NoteSequence原型中提取旋律并保存它们到TensorFlow的SequenceExample原型中，用于对Melody RNN模型的输入，将单音旋律转换为特定于模型的编码；

**melody\_rnn\_generate.py**

从Melody RNN模型的一个训练的检查点生成旋律，其中包含获取模型所使用的训练目录或检查点路径函数，从bundle\_file读取GeneratorBundle对象函数，以及生成旋律并保存为MIDI文件的函数等。

**melody\_rnn\_model.py**

完整运行一个Melody RNN模型，包含设置运行的默认配置、从引物旋律中产生旋律、存储一个MelodyRnn的配置、评估模型运行时旋律的log likelihood等功能函数。（如下图）



**melody\_rnn\_sequence\_generator.py**

构建一个序列生成器接口，包含创建一个MelodyRnn序列生成器，构建数字转换启动序列等。

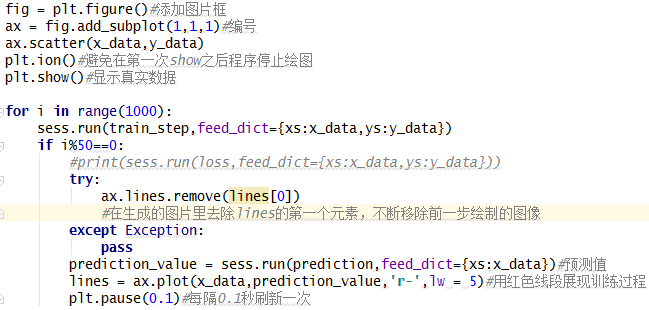
**melody\_rnn\_train.py**

用于训练一个Melody RNN模型。

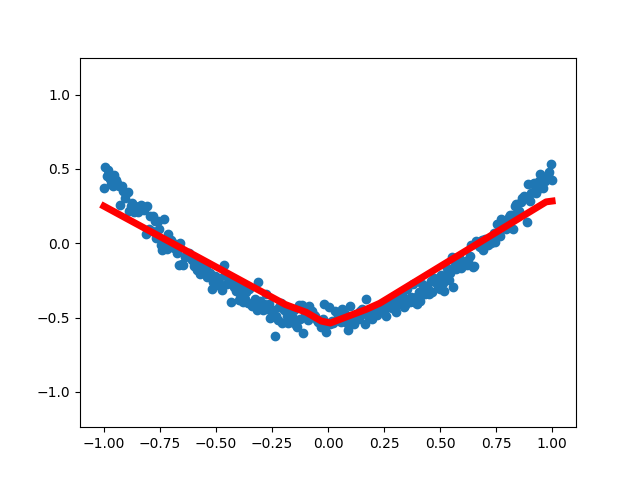
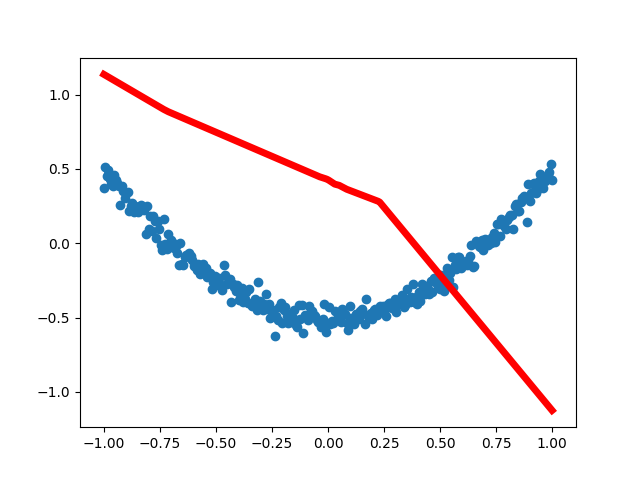
### Tensorflow相关学习

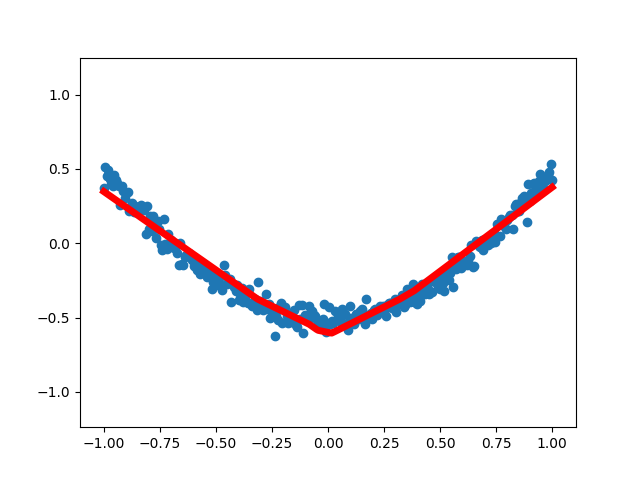
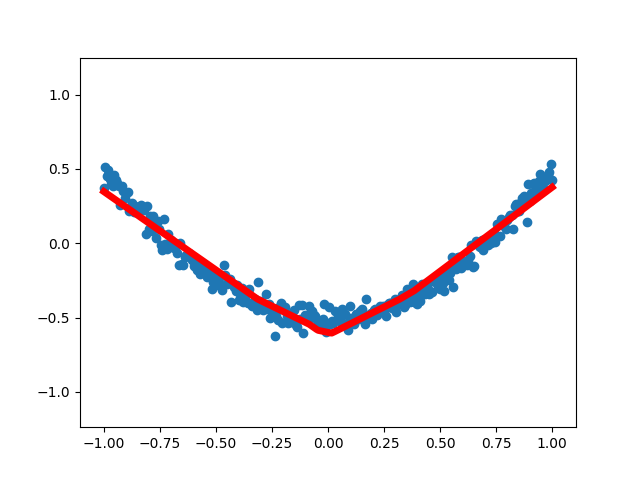
**①简单神经网络训练过程可视化**

上周学到了如何构建一个简单完整的神经网络，下面在原程序代码中添加如下内容，使神经网络的训练过程能够较为清晰的展现出来：（使用matplotlib包）



运行之后可以显示出一个动态的过程，下面展示部分过程：

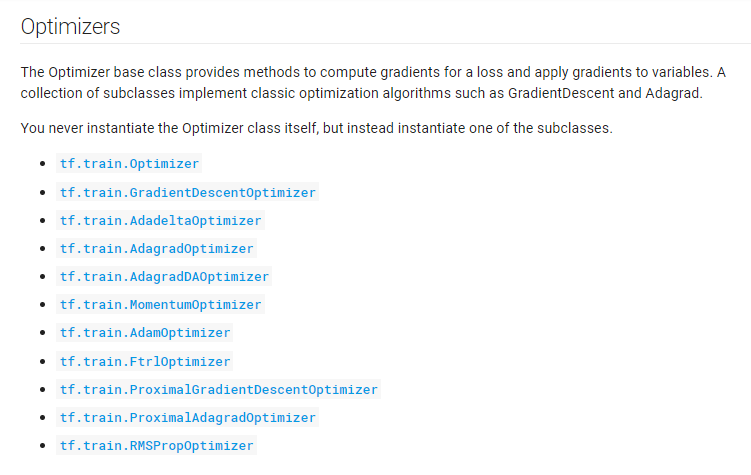




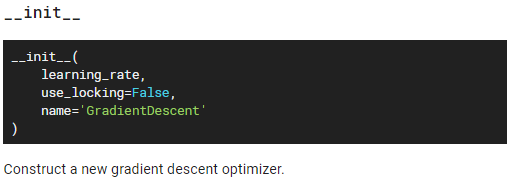
可以看到，训练出的结果曲线虽然没有完全吻合真实的数据曲线，但也越来越贴近。

**②Optimizer优化器简介（用于加速神经网络训练）**

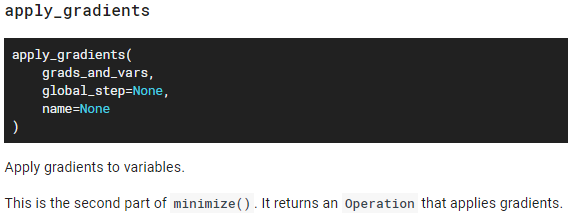
在tensorflow官网搜索optimizer，我们可以看到很多种优化器，它们的分类基本都是基于对learning rate（学习效率）的改变来区分的；一般初学使用GradientDescentOptimizer，更高级一点的可以尝试MomentumOptimizer等；



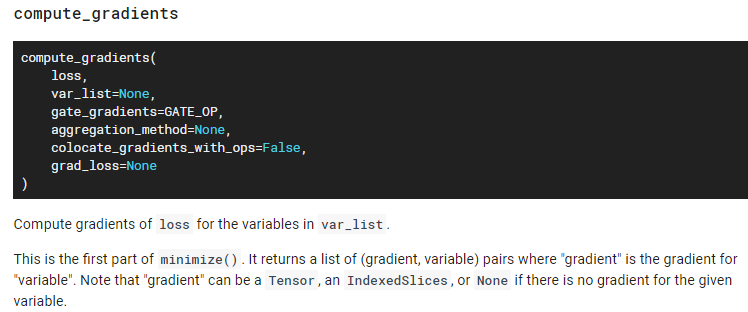
在这个部分我选了GradientDescentOptimizer即实现梯度下降算法的优化器类详细看了一下里面的主要函数和用到的参数：



首先，结合init的参数介绍可以发现，这个构造函数关键的参数只有一个learning rate，后面的use\_locking参数用于实现对更新操作的控制，name参数可选，默认名称是GradientDescent；



和函数名称一样，这个函数是用来把梯度“应用”到变量上面，即按照梯度下降的方式加到上面，属于minimize()函数的第二个步骤。其中grads\_and\_vars是由compute\_gradients()函数返回的(gradient, variable)列表，global\_step可选，用于在变量更新后增加一个变量。



Compute作为关键函数，属于minimize()的第一个部分，用来计算var\_list中变量梯度的loss值。它返回一个(gradient, variable)的列表，其中gradient是variable的梯度。而且gradient可以属于tensor也可以是一个IndexedSlices或者None。