# 基于LSTM神经网络架构利用MIDI短音乐段作为输入产生音乐

本周完成了该篇论文的翻译，并查阅了论文中所提到的RNN的相关知识。

## 摘要

现如今越来越多的人开始研究自动编曲的方式，但是极少有在研究过程中使用神经网络作为架构来实现自动编曲。

本文研究的是基于长短期记忆网神经网络架构（LSTM），通过输入一段音乐后计算机预测获得一段较为贴近原有音调模式产生新的音乐。训练该神经网络的数据集为74首MIDI格式Led Zeppelin的歌曲。所有的MIDI文件数据都会转换到二维数组内，假设这个二维数组为a[ i ][ j ]，那么这个二维数组的i、j值分别是MIDI文件的音高和MIDI tick[[1]](#footnote-0)。数组的数据通过顺序进行选择最终用于神经网络的训练，目前如何选择数据大致有四种方式，其中一种方式是移除歌曲内无声的部分，通过对最终生成音乐的结构进行分析发现，这四种选择数据的方式都是有效的。通过让参加者听通过LSTM神经网络架构生成的音乐小样并将这些小样以听的时候的愉悦度作为标准进行分级，最终发现移除歌曲内无声的部分作为训练神经网络的数据生成的歌曲是最为成功的。神经网络难以学会如何在音乐方式段落之间进行过渡，在未来的研究中，通过尝试增加数据集的大小等方法来解决这个问题。

## 介绍

作曲自古就是人类的一大爱好，它的历史可以追溯到公元前200年古希腊的Seikilos epitaph，先存历史最悠久的编曲比赛。近代人们开始寻找自动化作曲的实现方式，最近关于自动作曲的研究发表于1960年。作曲的产生是出于人们对于听音乐进行娱乐和通过音乐调整子的情绪的需求。作曲是项耗时耗力的工作，如果能让这项工作变得自动化那么一定会很有价值。

目前有许多自动化作曲的尝试，包括通过语言处理工具利用音乐的语法模式来进行作曲，或者是通过离散数学分析不同程度优秀的音乐。最近在许多科学领域，深度学习取得了成功，基于深度学习的神经网络实现自动作曲是必要的。处于这样的想法，本文将探究如何利用深度学习来实现自动生成音乐。

### 1.1先前的研究

对于利用深度学习实现音乐的自动生成的研究在此之前是非常少的。

Chen和Miikkulainen希望能够找到一个神经网络可以用于查找音乐的结构。为了能够对目标有最好的预测结果，他们使用进化算法来完成这项任务。他们所编写的音调和节奏的进化算法能够完成依照正确的音乐结构创作音乐。但是产生的音乐过于简单，而且系统不能处理多种乐器的情况。

Eck and Schmidhuber研究了当时流行的神经网络架构的问题所在，LSTM神经网络在其他研究领域获得了成功，他们认为其在音乐生成也是可行的。之后他们利用LSTM神经网络输入蓝调的代表性和弦成功生成了蓝调。其重点放在了网络如何让生成的音乐拥有正确的结构和节奏。

Franklin验证了神经网络中过去的数据对于音乐生成的重要性。通过RNN不仅想要完成想要编写特定类型的音乐，而是想要取代复制、创作音乐。通过LSTM，又由于音乐是可再版的，通过重配和声，新的音乐就能被创作出来。通过替代和学习和弦（符合音乐结构）能够创造出新的音乐。

Sak ，Senior 和 Beaufays 三人一起探寻了RNN在语言识别的运用。RNN在语音识别方面做的非常成功。这项成就为神经网络直接处理音乐文件提供了技术支持，人们对于音频的处理不再只局限于基于文频的音乐数据。

Johnston基于LSTM循环网络用许多的音乐来训练神经网络（ABC符号文本形式），并且接收之前所有的元素（音乐文件内）作为下一特征预测的输入。通过循环这样的方式，新的歌曲能够被创作出来，创作出来的每一个元素都将返回到RNN中去。通过对不同结构的音乐输入后测试发现，音乐都可以被产生出来。但是这样的方式只能成功产生一些简单的音乐，随着输入的音乐变得越来越复杂，生成音乐的音调开始变得变那么准确，同时利用ABC符号来代替原有的音频文件也耗时过长，这并不算成功。

这五个基于神经网络简单实现音乐生成的研究为LSTM网络的建立提供了坚实的基础。但是，此次研究也是基于输入音乐的片段来产生音乐，它也不会使用基于语法模式而是用MIDI文件作为替代。与之最为类似的研究仅仅研究单声部的音乐，但是本文中使用的系统能够支持复调的音乐。此次研究也可以作为最近大火的深度机器学习领域的一个优秀的例子，为之后再深度学习方面的研究提供了不错的基础。

### 1.2 研究范围

此次研究的目的是编写一个基于数秒的旋律就能够自动生成音乐的程序。这个程序基于神经网络，并且该网络已经使用其他音乐文件进行过训练。根据输入的旋律，借助神经网络在其他音乐文件中学习到的权值，该程序能够预测这段音乐之后音高最有怎样的可能进行变化。将预测的旋律附加到原有输入的旋律中去，通过重复这样的步骤，音乐就生成了。

值得注意的是，音乐是被编码的，而系统编码时候，大多数被编码的音频文件都会在描述音乐音高的时候加入特殊的时间戳。

虽说有一个足够大的数据集，该网络能够学会如何在系统里工作，但是如果将它转换到一个系统，那么会变得更加容易被训练，也能够大大提高训练的精度、降低训练所需的时间。

如先前所言，许多产生音乐的方式都基于研究中所使用被编码的音乐所定义音乐编曲语法。绝多大多数的研究文献综述资料中也是这样做的，而本文中所使用的编码是MIDI格式，虽然说这种编码的表示学习起来比较困难。但是一种算法如果能够将文件内容转换到系统里去，那么神经网络将会更加容易被训练它产生的结果也会更加准确。

MIDI文件格式将会在2.2进行详细地描述，通过这种编码方式最为重要的是计算机能够知道音乐什么时候在播放，知道音乐每一Tick编码后的音高。这个算法能够将音乐文件转换到一个系统（特别是一个数组），将时间的表示方式从相对表示转换为绝对表示，每一个时间段转换为一个Tick。

本文中使用的神经网络是一个长短期神经网络（LSTM），是递归神经网络的一种特殊形式。在2.3将会被详细地解释。LSTM神经网络允许音乐文件顺序结构的输入输出，也可以只任何计算节点之间。由于音乐一般是顺序的，一段确定的顺序的音高会紧跟着其他的音高，因此做到这一点非常重要，即使是序列模式下这样的情况也存在（例如歌曲转换处或者是副歌部分）。

此篇论文关于数据集的内容在2.1。在解释完MIDI编码系统和神经网络的实现后，对于这个系统的一些变化的部分在2.4，然后紧接着在2.5主要讲的是如何在进程中预测新的音乐片段。后期处理在2.6，对研究结果的评测在3，关于此次研究的讨论和总结分别在4,5。

## 2 方法

### 2.1 数据集

对于此次研究，此次研究所使用的数据集为Led Zeppelin的74首曲子。每一首曲子都被编码为1型 MIDI文件（具体将在下一节进行讨论），文件获得于zeppelinmidi.com，网站上对于如何下载该文件做出了介绍。

对于这些文件所要研究的是歌曲的音轨，因为歌曲的音轨能够很好地体现歌曲的情况。这样能让机器学习具有更强的表达功能，也使得计算机能更好地学习和制作音乐。处于对产生结果准确性的考虑，选择单一的乐队作为数据集的输入。

结束对机器学习的训练后，选择四个不一样的文件来进行歌曲的预测。第一个文件是手工创建只包含四个单音的曲子。第二个文件是Darren Tate vs. Jono Grant一首简单结构多重复的trace曲—— Let the Light Shine In (Arty Remix)。第三个文件稍微复杂些，是一首爵士乐，Washington & Young的Stella by Starlight。最后一个文件是数据集中所包含的音乐——Stairway to Heaven。

### 2.2 MIDI Tick 数组

### 2.2.1 MIDI格式

MIDI将所要演奏的乐曲信息用[字节](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82" \t "https://baike.baidu.com/item/MIDI/_blank)进行描述。譬如在某一时刻，使用什么乐器，以什么音符开始，以什么音调结束，加以什么伴奏等等，MIDI文件本身并不包含[波形](https://baike.baidu.com/item/%E6%B3%A2%E5%BD%A2" \t "https://baike.baidu.com/item/MIDI/_blank)数据。但是这样的格式是没有办法直接输入到机器学习中去，因此需要编写一个新的算法来将这个编码的乐曲转化为二维数组。一个轴所包含的是音乐的每Tick的的信息（用绝对代替相关），另外一个轴包所绘制的是音高。二维数组的值是音高的大小。

MIDI文件存储了多种events，其中包含着关于歌曲信息的Meta events（例如：歌名），非冗余events有NoteEvents, ControlChangeEvents, PitchWheelEvents, SysExEvents, ProgramChangeEvents 还有 EndOfTrackEvents。

NoteEvents可以控制特定音高的速度，还能够控制音高停止的事件。

其他Events控制发出音乐发声的方式（例如让音乐听起来像是其他类型乐器）。但是由于此次所使用来训练神经网络的音乐只有一种乐器，因此其他events所产生的影响应该被忽略。但是为了存储正确的时间结构，Tick值应该被存储下来。

MIDI有三种类型用来处理并行的轨道。一条轨道是多个events的集合，在1型MIDI文件中，每一种乐器都占有一条轨道。0型MIDI文件中不同的乐器都在一条轨道上，然后使用non-NoteEvents来进行控制。2型MIDI文件因为需要使用一个系统来向文件内添加音乐轨道，现在已经很少使用了。但是由于数据集中的文件都是1型MIDI，所以并用在意是不是单种乐器。因为1型MIDI文件是单音乐轨道，所以可以把他认为是单种乐器。解码系统也能使用0型MIDI文件（不在数据集中），但是会创建比1型MIDI文件更为杂乱的数组。

### 2.2.2 编码

因为MIDI文件能够播放128个不同的音高，数组的列所对应的是向量，向量所对应的元素代表着相应的速度。MIDI文件内每一Tick都含有属于当前Tick的向量，这些向量创建大小为128的矩阵，算法如下（Python-esque的伪代码）

total\_ticks = 0

musical\_events = []

track = MIDI\_file.getBestTrack

for event in track:

if isMusicalEvent(event):

total\_ticks += event.num\_ticks

musical\_event.append(event)

grid = matrix(128, total\_ticks)

current\_vector = matrix(128, 1)

position\_in\_grid = 0

for event in musical\_events:

if isNonNoteEvent(event):

position\_in\_grid += event.num\_ticks

else:

if event.num\_ticks != 0:

for i in event.num\_ticks:

grid[:,position\_in\_grid]=current\_vector

position\_in\_grid++

if isNoteOffEvent(event):

current\_vector[event.pitch] = 0

if isNoteOnEvent(event):

current\_vector[event.pitch] = event.velocity

Python-Midi2[[2]](#footnote-1)的工程包可用于访问MIDI文件的内容。

程序（伪代码getBestTrack）所选择的轨道轨道为第一条轨道，包含NoteEvents。与磁鼓磁道相一致，选取的轨道中Events为最多。在本篇论文中，所使用的音轨是数据集中含有NoteEvents的第一条轨道。

由于创建的多个向量需要存储包含128个音高所对应的速度，而且由于events包含每一个音高的相应信息，并且向量之间相互联系，tick的数量存储在event中意味着要将连续的per-tick向量存储在矩阵中去。

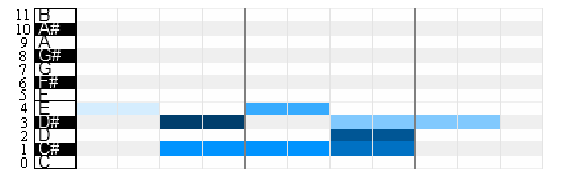


图1:MIDI文件的可视化。深蓝色的单元表示音调的速度（体积）。128个音高中只有12个被表示了，每一小节表示1tick。

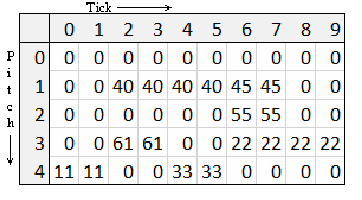


图2:将MIDI event转换为二维数组的实例。为了展示的便利，仅展示最初的五个音高（而非 128）。其他剩余的都是零。

当event中的tick不相对于为零的时候添加向量，这就意味着稍后该时间会被触发（相对于前一时间），向量会复制这个持续时间的值。只要向量位置被处理，那么向量就能为下一Event变化。速度为零意味着之前播放过的声音不会再播放。这一般由NoteOffEvent控制，但是速度为零的NoteOnEvent也能够控制这样的情况。伪代码中的最后一行代码将指定音调的速度输入到 to-be-placed vector中去，图1和图2为例。

该算法将MIDI文件编码为二维数组。将该数组如何用于训练LSTM网络，到第2.3.2进行详细解释。

### 2.2.3 解码

该二维数组也可以解码回MIDI文件。并将预测tick时间内的矩阵转换为MIDI文件。预测过程在2.5节进行具体描述。

解码该数组的算法如下（Python的伪代码）：

track = newMIDITrack()

previous\_vector = grid[0] # first vector

for note\_index in previous\_vector:

if note\_velocity != 0:

track.append(NoteOnEvent(0, previousvector[note\_index],note\_index))

tickoffset = 0

for vector in grid:

if previousvector == vector:

tickoffset ++

else:

for note\_index in previous\_vector:

if previousvector[note\_index] == vector[note\_index]:

Continue

if previousvector[note\_index] != 0 and vector[note\_index] == 0:

track.append(NoteOffEvent(tickoffset, note\_index))

else:

track.append(NoteOnEvent(0, vector[noteindex], note\_index))

tickoffset = 0

tickoffset ++

previousvector = vector

该译码算法从数组中第一个向量的内容开始，只有并触发所有相关的事件（对于所有非零的元素）。然后利用tickoffset对所有的向量进行迭代和比较。当有连续的向量是相同的时候，增加tickoffset的值。否则对内容进行比较。任何不相等的元素都会触发另一个事件。

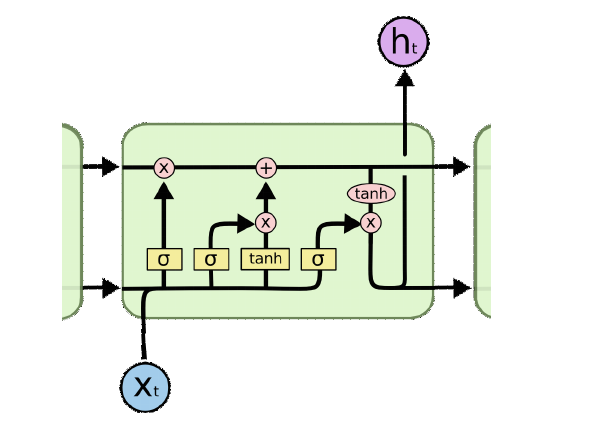
如果前一向量的音高不为零而且当前向量为零，那么所使用的是 NoteOffEvent，否则所使用的将NoteOnEvent来进行控制。在一个向量内如果发生了变化为了保持events之间的相对关系，tickoffset将会被重置。

### 2.3 LSTM网络

### 2.3.1 背景

RNN神经网络是一种节点定向连接成环的[人工神经网络](https://baike.so.com/doc/1591389-1682151.html" \t "https://baike.so.com/doc/_blank)。这种网络的内部状态可以展示动态时序行为，对暂存和动态时序的文件进行编码转换。处理单元之间既有内部的反馈连接又有[前馈](https://baike.so.com/doc/2282021-2414071.html" \t "https://baike.so.com/doc/_blank)连接，该神经网络通过合并之前步骤所处理的信息和当前输入的信息作为节点来完成的。

但是RNN出现了一个问题，这样的网络结构会出现长期依赖问题「11」，学习相对较早以前产生的关系往往具有较低的权重，随着时间的推移重量减少，但是这个问题最终通过LSTM了。



图三：一个普通可视化的LSTM模型。LSTM 的关键就是细胞状态，水平线在图上方贯穿运行。它接收来自第二状态，还有底部穿过的线。两个状态都接收来自前一模型的输入，但是第二状态还接收额外的输入（xt），产生一个暂时的输出（ht）图片来自一个2015年的blog[[3]](#footnote-2)。

与RNN类似，LSTM由神经网络链所构成。与一般的RNN不同的，LSTM有着相对简单的架构（如一个模块包含一层执行节点之间的计算），LSTM模块围绕着细胞状态构建，细胞状态是模块中两个管理模块中其中一个。LSTM的细胞状态包含着RNN的记忆。它是通过转发前一个模块的输出进行创建，并且在转发到下一个模块的单元状态之前接收来自第二状态的信息。第二状态接收输入（常规神经网络的输入）并且通过特定的计算来重新确定新输入的哪一部分将输入到细胞状态中去（存储器），并确定哪个部分将作为瞬时输出进行返回。LSTM具体计算的实现大概会有所差异，但是其标准框架大致如下：

从细胞状态中丢弃什么信息。这个决定通过一个称为忘记门层完成。可以通过计算来确定细胞状态中哪些东西是应该被遗忘，以便新的信息能够替代旧的信息（例如可以通过计算方差来确定）。在一个标准的LSTM网络，忘记门层能够被如下计算：

ft = σ(Wf · [ht , xt ] + bf )

σ指的是Sigmoid函数，f决定什么是要被遗忘的。由于使用Sigmoid函数因此，这个值介于0到1，而且这个范围对应着遗忘的程度：越小的值，遗忘程度越高（从细胞状态中删除）。W指的是神经网络节点相应权值。x和h是输入，x对应新输入，h是前一模块数据转发，B为一个常数。

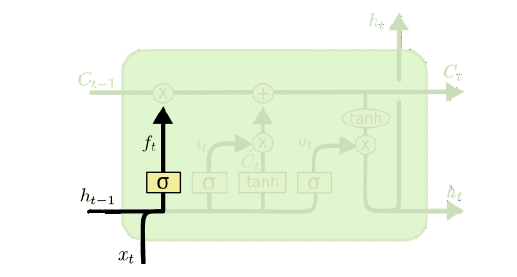


图4：忘记门层

下一步是确定什么样的新信息被存放在细胞状态中（从输入），通过输入门层来决定什么值将要更新替代原有的值。这些组合的结果，加（数学加法）到新的细胞状态。由于这个算法不去乘以非常小的数（太小的数字会导致梯度消失），而是选择加的做法能够解决不同类型的（递归）神经网络中梯度消失的问题。

it = σ(Wi · [ht−1, xt ] + bi)

C˜ t = tanh(WC · [ht−1, xt ] + bC)

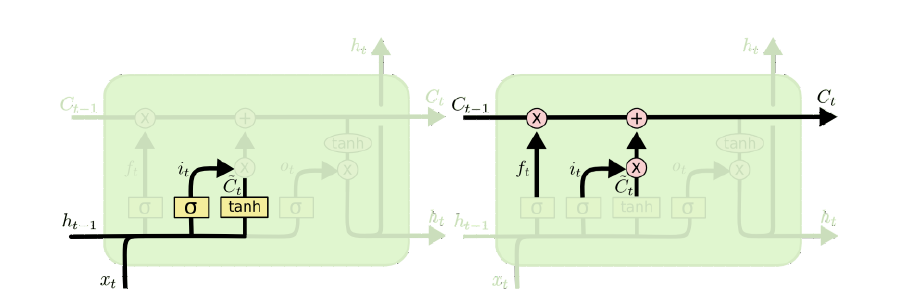


图5：输入门层

这决定了哪部分被更新了。i是取代之前所遗忘的输入信息，C是新的细胞状态。 tanh的取值范围是-1到1，神经网络建立在这样一个范围。使用这些值，如何更新细胞状态如下：

Ct = ft ∗ Ct−1 + it ∗ C˜ t

f先与之前细胞状态相乘，然后添加到新的细胞状态i ∗ C˜。

确定输出什么值，将通过输出门层来决定。这个输出将会基于我们的细胞状态，但是也是一个过滤后的版本。首先，我们运行一个 sigmoid 层来确定细胞状态的哪个部分将输出出去。接着，我们把细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。一个标准的LSTM神经网络中，输出层如下进行计算：

ot = σ(Wo[ht−1, xt ] + bo)

ht = ot ∗ tanh(Ct)

o代表着所要提交的输出，通过细胞状态的相乘创建出一个状态传递到网络的输出节点。状态发送到网络的输出节点，并被发送到下一个模块（该进程再次运行）。

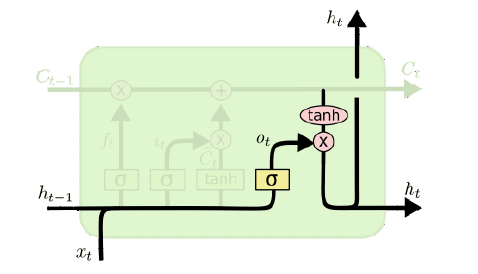


图6：输出门层

虽然在不同的LSTM神经网络实现中这些门层的顺序可能会有所不同，但是忘记门层一般在输入门层之前，否则会出现神经网络忘记了刚刚学到的。最后通过输出门，模块最终输出它在周期中学到的东西。

上述描述了以在论文中所使用的神经网络的架构解释了标准的LSTM网络。所使用的框架python为脚本语言的Keras[[4]](#footnote-3) 和 Theano[13]。

### 2.3.2 安装启用

该网络如图7所示，由三层组成。将输入值输入到由512个节点的LSTM神经网络中去。紧接着，dropout和正则化来减少过拟合的情况，在次之后还有一层有512个节点相对应。稠密层将作为输出的节点数降低到128个（有128个可能的音高）。该程序架构是由实验确定的。通过线性激活的方式可以将均方误差来训练减小损失，RMSProp优化[12]可用于加速神经网络的训练。



图7：神经网络。第一个有512个节点的LSTM层被输入了一个由MIDI文件转换为二维数组。Dropout和正则化将会处理另一个512个节点的LSTM层。Dense层之后将这些节点与一个128\*1的向量相链接作为输出。

通过MIDI文件来训练神经网络。如前一节所述，将MIDI文件转化成一个矩阵，然后将矩阵中的内容复制到输入向量序列，每个向量都有相对应的标记向量。序列的长度需要足够大，以便输入的能够包含不同的音高（理想状态下所有旋律的变化），但是也不能太大，否则可能会无法找到输入和标签之间的关系（导致在损失函数中出现过高的损失）。标签向量比矩阵中最后一个序列向量还要长一点。

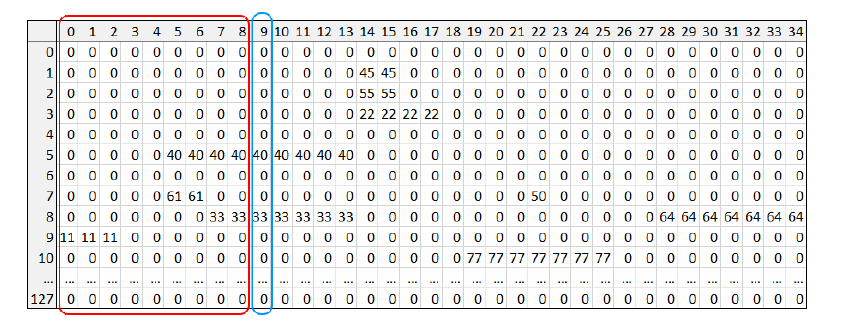


图8：输入的选择和由MIDI矩阵转化而成的标签向量。红色部分是被选择为输入的向量序列（长度为9的序列）。蓝色部分是被选择的标签向量。

虽然序列的大小通过实验已经被确定了，有效的数量为12，因为（如文章早先所说）MIDI格式中的Tick与MIDI分辨率中每小节中四分音符的数量一致，而且12能够用4和6所除，这就允许不同节拍来组成曲子。然而数据集中的音乐都可以用4来组成，这就是所使用的序列的大小都是4的原因。

选择了输入和标签向量后，将所选择列的指定步长向右移动，然后选择下一个输入向量序列和相应的标记向量。这些序列和向量将添加到两个列表中去。只要数据集中的数据被处理完毕，向量序列和标签向量将会分批地进入到神经网络中去。由于硬件的限制，分批处理是必要的，因为同时处理太多的序列CPU的内存很快就会溢出。该神经网络每批次将会训练50次，训练和预测将会在分布式ASCI超级计算机4中进行（DAS-4）[[5]](#footnote-4)。

### 2.4 变形

四种不同LSTM的变形将会处理先前在论文中所说的分批处理，从而评估LSTM中细微的不同是如何影响音乐生成的。

* 常规/标准的选择批次：在编码过的音乐数组中连续地进行选择。每批次/序列由64个向量构成。这意味着被神经网络所学习的向量数为64（每个序列由一个标签向量）。
* 去除为空的向量：有时候音乐中音轨出现无声的部分是很正常的（包括来自数据集中的音轨）。在一个编码过的数组中，将会以为空的向量进行表示。在音乐无声的时候这样学习可能会变得更加有效，否则由于向量序列和相应的标签向量相似可能会导致。因此在训练神经网络的时候，将为空的向量从编码数组中删除对于训练神经网络会变得有效。
* 较大的序列大小：在实现中所选择批次/序列的大小有不同的选择。（主要由于循环预测的问题，具体将在第三节进行描述）。序列大小为96（中）和160（大）在在LSTM变形中使用，通过这样的方式试图降低过拟合并且能够学习更大的音乐结构。
* 随机批次选择：在处理选择批次的过程中，选择从编码后的矩阵中随机抽样1000个，而非使用步长一致连续批次的向量。

### 2.5 预测

完成了神经网络的训练之后，预测程序就能进行允许。基于输入歌曲的一小部分，以相同的一Tick的数组格式用来训练神经网络。利用相同的序列长度来确定段的大小。通过输入的矩阵和学习的权重对于下一Tick预测一个新的向量。下一小节神经网络觉得最有可能的曲子是由向量的值所确定的（由训练时的遗忘函数所决定）。完成预测一个向量之后，对应于输入段的数组中的每个向量向左推一个位置（列），并删除第一个向量。数组中预测产生的向量成为了数组中最后的向量，重复这样的步骤，预测另一个向量再调整数组，最后获得输入歌曲之后预测部分的音乐。

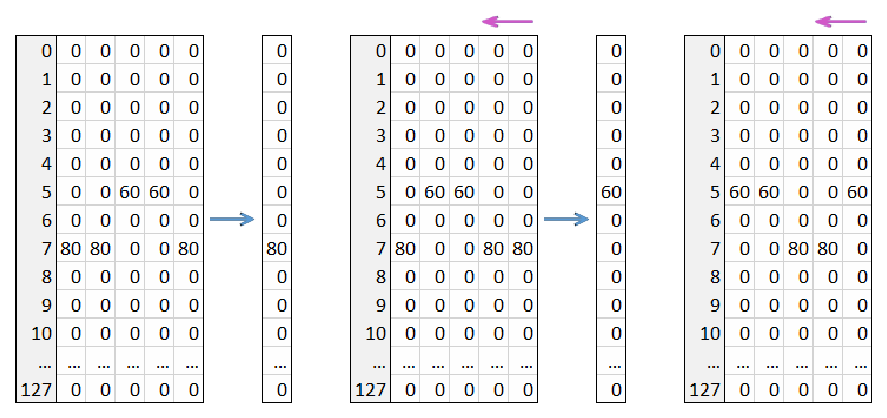


图9:预测之后的向量。该向量会向左进行移动，预测产生的向量添加到数组中去。预测产生另一个新的向量然后改变为另一个数组。序列长度为5。

在预测过程中，预测产生的向量都会被单独存放在一个单独的列表中，这样这个数组就能转换成为MIDI文件。由于线性神经网络的性质，向量内部的值可能需要被裁剪为不低于0或者高于127的值。

### 2.6 后期处理

为了让产生的音乐听起来更加流畅，预测完成后一些后期处理是必要的，将区别较为微小的音高用连续的音高所代替可以完成这样的工作。通过查看每一个预测数组中向量的速度，将数组内出现的最高速度音高查找出来并存储。

下一步，将每一个向量的音高的速度值与出现的速度最高的音高进行比较，如果二者之间的速度相差超过10%，将该音高相关的向量设置为0。反之，将其设置为100。在最终生成的文件中所有的速度值将会变为MIDI文件常见的100。（此次论文内数据集中的大部分文件速度为100）。

使用这种比例比较可以捕捉必要的音高，但是在预测中使用这样的办法并不可行，但是仍然能获得一些重要的旋律（例如：背景音）。使用常数比较阈值并不能够获得这样的结果而且会使高速度值变为单一的长音。（由于过于微小的差异会被认为是微不足道，因此变得平滑）。

除了这种修匀法以外，将所有过低速度值完全移除掉，这些值所对应的音调在音乐实际编写过程出现的可能性并不高，如果将这些值留在数组中，那么它们会变成柔和的背景噪音。通过使用每一个音调出现的最高速度值，最容易能够找出该音高是否包含有意义的音乐。如果出现的最高速度低于5，则将数组中每个向量的速度值设置为0。

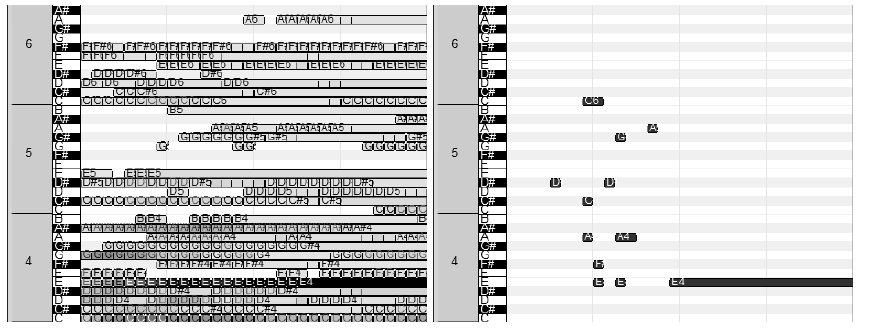


图10：预测产生的音乐进行后期处理的前后

当更多的训练和更大的数据集变得没有意义的时候，后期处理的数量也许会减少。

### 3 结果

在本文中对于音乐评价较为困难的，由于对于音乐的评价是较为主观的。然而，预测产生的音乐能够通过常见的音乐结构进行推断，这首先进行处理：通过研究四种批次选择产生的结果进行分析。

之后为了能更加客观地分析主观的音乐，通过调查的形式进行实证调查来进行评价。

### 3.1 常规/标准的选择批次

通过标准的方式选择批次利用example.mid作为输入，生成了结构化的音乐（见图11）。使用jono.mid和stella.mid作为输入，会生成以一段混乱的噪音开始的音乐（见图12）。然而这两个文件并不能产生除无声部分以外的会被后期处理过滤掉的极低速度值。使用stair.mid能生成音乐刚开始的288ticks像天国的阶梯一样，之后就变成混乱的杂音了。虽然利用常规/标准的选择批次的手段打破了这一点，产生的音乐具有一定的结构，但是它并不和原有的音乐相一致。

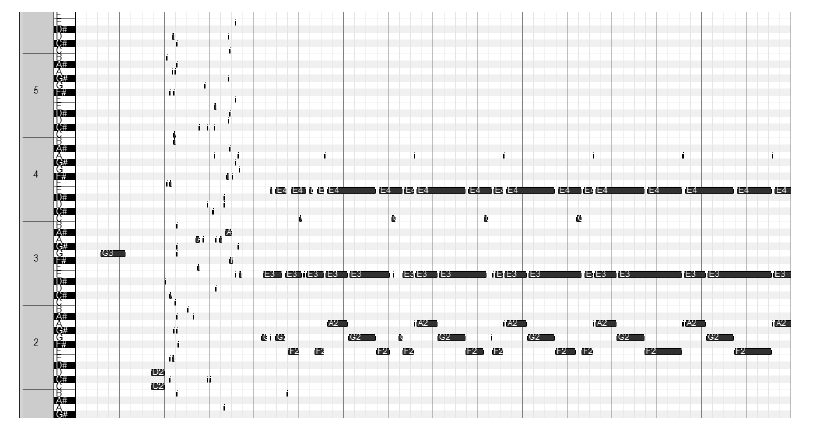
值得注意的是在下列图片中，输入的音乐文件可能不会和附录中音乐的内容完全一致，这是因为后期处理的过程中，输入的某些部分可能会被过滤掉。但是这并不意味着它在生成过程中没有被完全使用，因为后期处理是在进行预测过程之后进行的。

图11：以example.mid作为输入使用常规方式选择批次进行预测。灰色垂直线的包含32 ticks。输入的音乐是前64 ticks。在135 ticks进入到旋律中前，生成的音乐（64 tick开始）开始出现杂音。

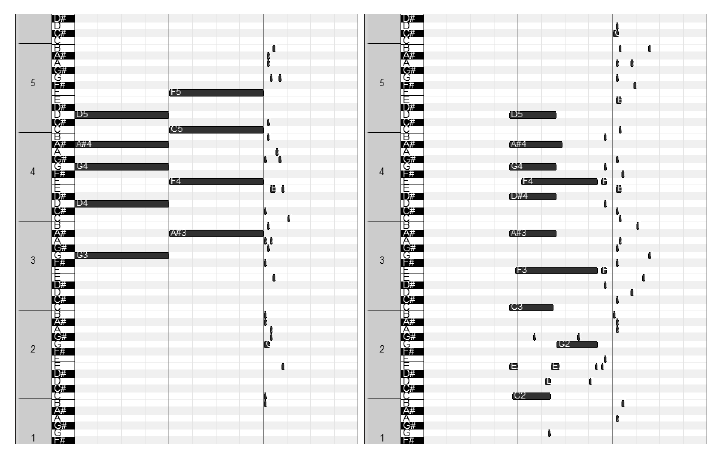


图12：使用jono.mid和stella.mid作为输入使用常规方式选择批次。两段音乐都包含有一小段杂音并且没有能继续进行生成有效的音乐。

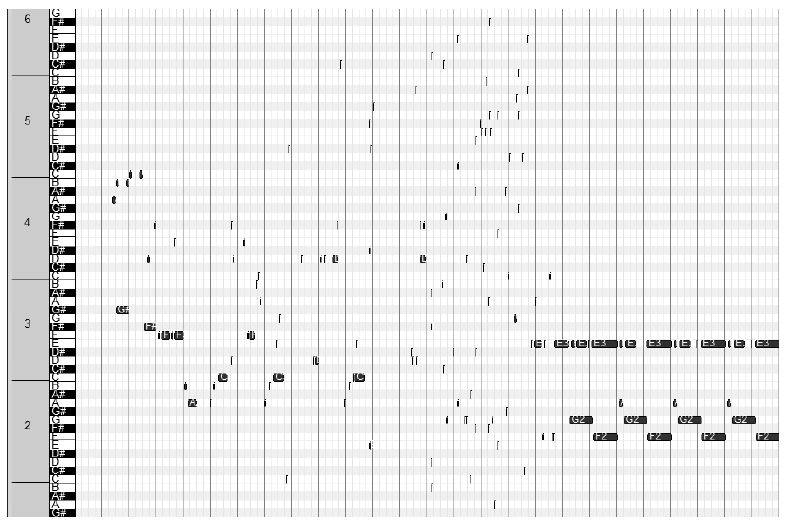


图13：使用stair.mid作为输入利用常规的方式选择批次进行音乐预测。为了能够展示更多的歌曲将图片进行了缩放。在输入文件开始后64 ticks，生成的旋律的288 ticks与天堂的阶梯类似，其中有一个充满噪音的部分，其次是一段重复的旋律而非远程歌曲的部分。

### 3.2 去除空向量

此种LSTM神经网络的变形采用的是在训练期间去除输入文件中的空向量，使用序列长度为64的常规选择批次方式。当神经网络无法产生任何东西它并不会以沉默作为结束，它会让结果显示得像是正确的。预测的歌曲比之前显示的歌曲方差会更大一些，下图将会被压缩以显示更多ticks。

令人惊讶的是，该网络通过这种方式比使用常规批次选择能够更好地从一个模式过渡到另一个模式。但是利用定期批次选择发现在一些相同的片段同样会出现杂音。

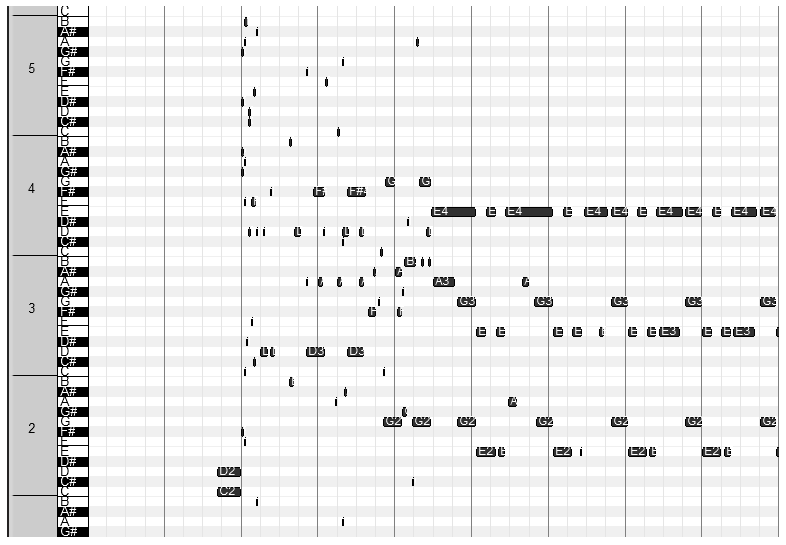


图14：使用example.mid作为输入使用零向量消除的方法预测产生的音乐。输出后，大概有一段32 ticks的杂音，接着是大约48 ticks 的旋律，之后是一个不同的旋律，然后重复这样的状况。

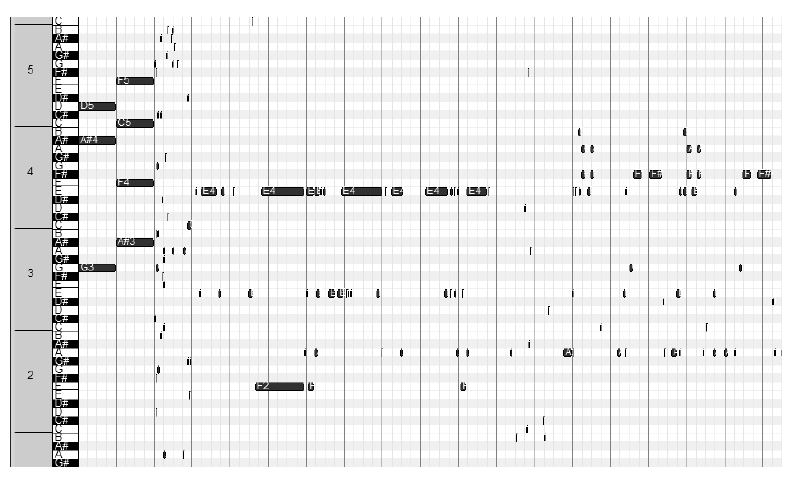


图15：使用jono.mid作为输入使用零向量消除的方法预测产生的音乐。在杂音之后，有一段变化较小的长周期长音，然后是一个短间隔，紧接着是一段较为悦耳、结构重复的音乐。

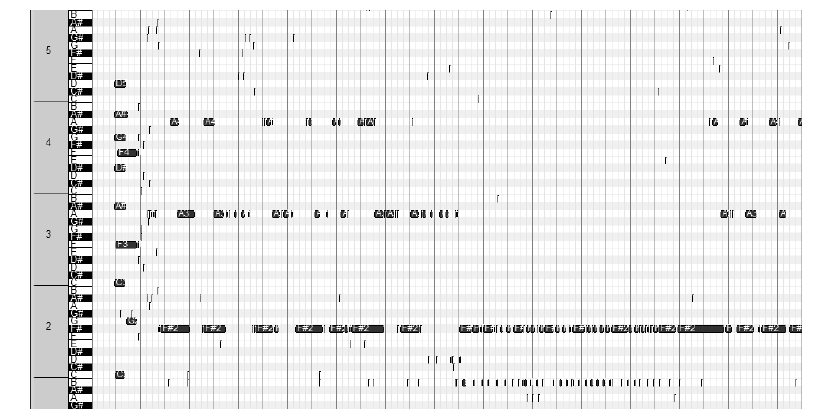


图16：使用stella.mid作为输入使用零向量消除的方法预测产生的音乐。大量地产生音乐，但是这些音乐的结构重复，充满了极端值。

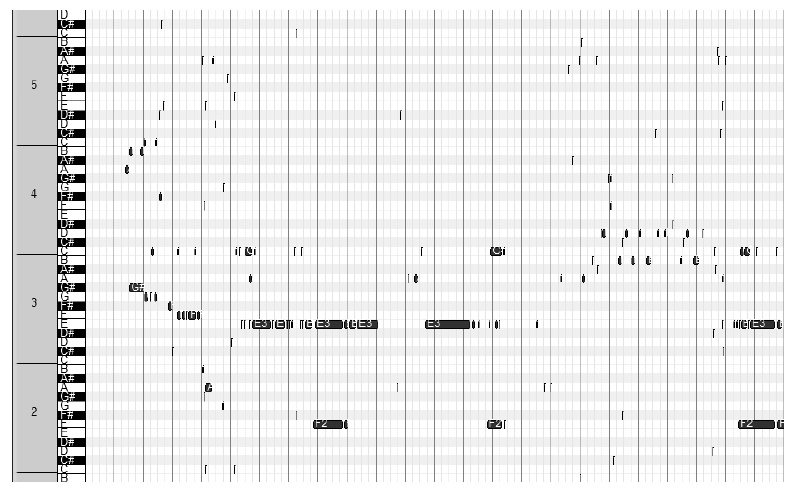


图17：使用stair.mid作为输入使用零向量消除的方法预测产生的音乐。在开始的96 ticks预测产生的音乐听起来像天堂的阶梯。接下来是结构简单的一段：在转换为简单结构之前，有160 ticks的较为悦耳的片段。然而并不是原创歌曲的一部分。

### 3.3 较大的序列大小

较大的序列大小能让计算机网络处理更为长的音乐结构，而且处理转换的效果比常规的64 ticks要更加优秀。下图的输入的序列大小相对来说是较大的，三段用灰色线进行分离，相对应的是一个中等批次的大小96 ticks和五段160 ticks的大批次。但是数据显示，预测效果并没有提高。中等大小在停止或只存在一个持续的单音之前能够预测100 ticks的。序列大小较大，神经网络会产生一小段杂音或者什么都不产生。

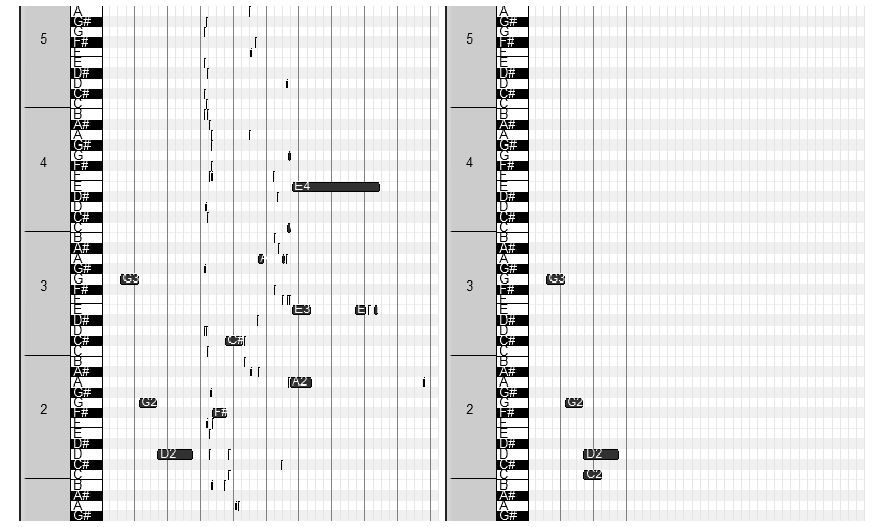


图18：用example.mid作为输入使用较大序列大小的方式。左边为序列大小为96（中等），右边序列大小为160（大）。一些音乐通过中等的序列大小产生，但是没有使用非常大的序列大小。

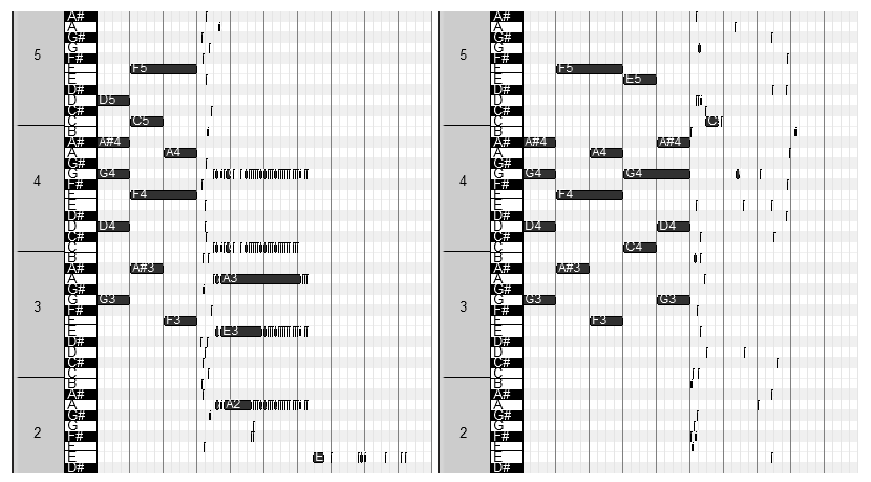


图19：用jono.mid作为输入使用较大序列大小的方式。左边为序列大小为96（中等），右边序列大小为160（大）。中等大小序列产生了一些带结构的音乐（音高时有时无）。在停止产生音乐之前，大型序列一般会产生噪音。

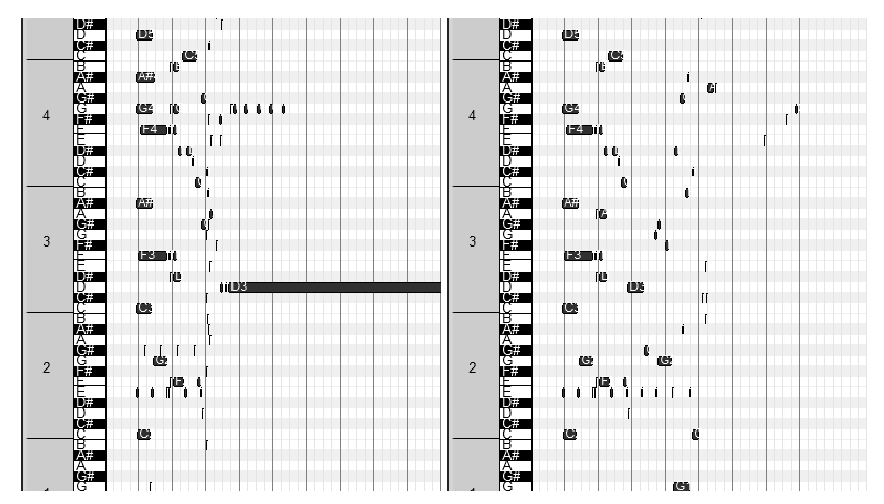


图20：用stella.mid作为输入使用较大序列大小的方式。左边为序列大小为96（中等），右边序列大小为160（大）。小序列产生音乐，中等大小序列产生一个持续的单音，大型序列产生一些零星的音调。

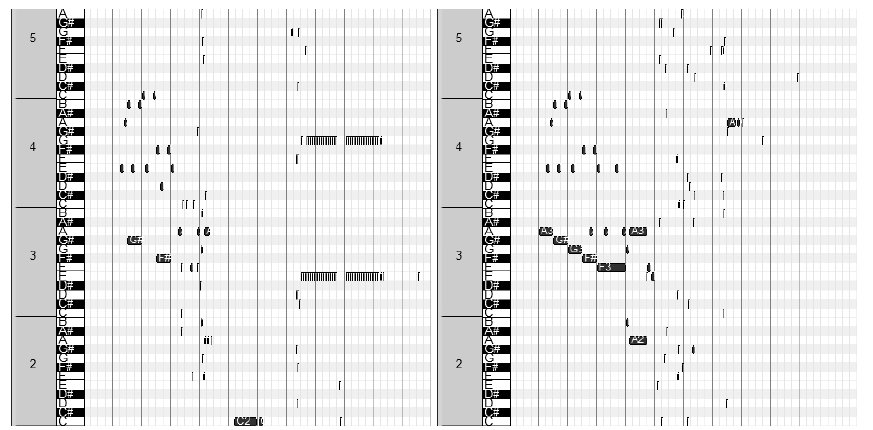


图21：用stair.mid作为输入使用较大序列大小的方式。左边为序列大小为96（中等），右边序列大小为160（大）。在中等序列中，只有刚开始的时候有一点像原创歌曲。大序列只会产生噪声。

### 3.4 随机批次选择

这个批次选择的变形尝试了1000个随机批次还有3000个。由于神经网络没有办法预测这两种输入的输入歌曲，只有三个结果在图片中显示。都使用批次大小很大的，即输入大小为160 ticks，对应着用灰色垂直线分割为5个。

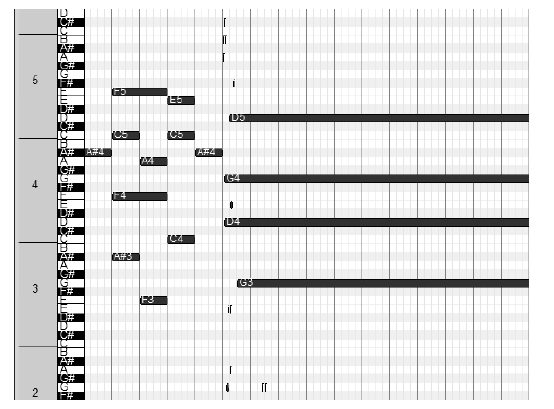


图22:用jono.mid作为输入使用大小为1000的随机选取批次。音乐预测卡在同一音高并不断地继续预测。

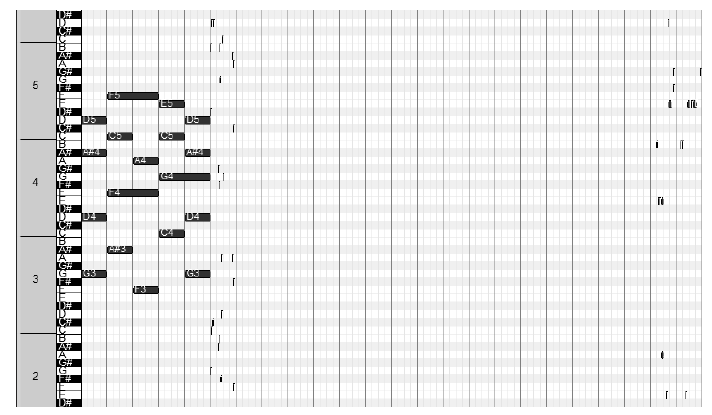


图23：用jono.mid作为输入使用大小为3000的随机选取批次。有趣的是，神经网络预测长时间变为无声，但最终会过渡到非静音。

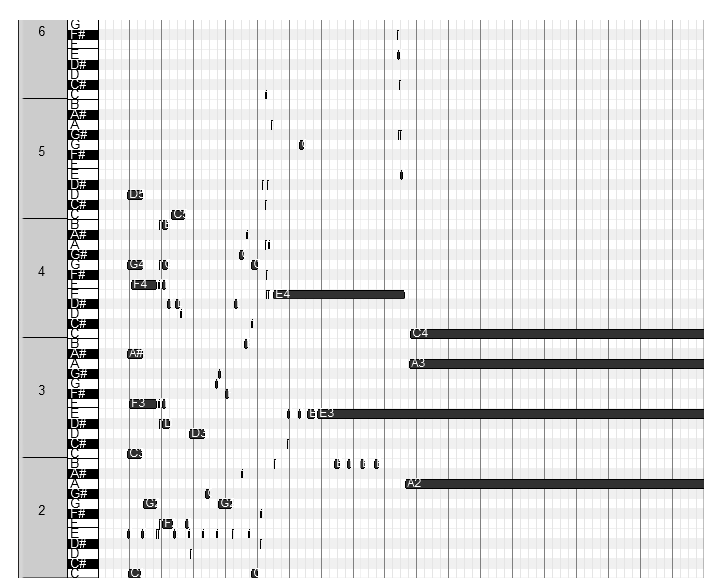


图24：用strlla.mid作为输入用大小为1000的随机选择批次。在172 ticks开始的长持续音调结束时，神经网络预测了如何完美过渡到下一连续的音调。

### 3.5 评价

通过定期批次选择，使用jono.mid和stella.mid无法产生远超过杂音的小片段。这有可能是由于用于训练神经网络的曲目包括长时间无声的部分，这可能会导致神经网络的预测由于内存不够大，无法转换为非无声状态下，最终停留在一个持续性的无声状态下。

对于产生杂音的原因大概是由于LSTM神经网络没能发现其短时记忆输入中ticks值和它通过数据集进行训练所学习到的音乐在不在调上（如：在example.mid中，几乎没有随机的音调的情况）。由于对于神经网络来说它并不能够正确地认知音调，他不能成功地找到一个相似的音高而非返回数个不相似的音高。这样的情况会持续到神经网络能够识别一个模式，然后对一个结构进行学习。这样的解释是基于定期批量选择相关的事实，神经网络在对于继续编写天堂的阶梯并没有遇到困难（随着产生音乐的继续，神经网络越来越难产生原创歌曲），这说明与之相关最重要的是输入。通过定期批次选择，神经网络在产生一段杂音之后，最终会产生一个结构重复的音乐。这意味着神经网络正确地学习了音乐的结构，而非在数据集中找到了对应的歌曲，但是较短过于重复的音乐结构是会让人感觉单调。理想状况下，神经网络能够产生结构不同的旋律。发生这样的情况是可能是由于训练过程中神经网络学习到的某种结构正好符合指定的序列大小（批次）：如果在数据集中出现连续出现一个听起来比较悦耳的音乐结构，而且其中一个事件和序列大小相一致，那么在预测结构的结尾时，神经网络可能会觉得这是第一次出现然后不断地重复。由于序列的大小决定了神经网络的内存，所以神经网络没有办法记住它已经对于这样的情况已经预测过多次了。而且使用序列大小较大的序列又会因此产生大段的杂音而无法产生音乐，因此很难用这样的结果说明较大的序列大小有助于减少结构的重复并且能够有效地对音乐进行过渡。

一般来说，较大的序列大小，尤其是序列大小为160 ticks，神经网络对于产生音乐更加困难（利用example.mid作为输入并没有产生任何文件）。通过增大序列的大小也许能够找到正确的音调。也有可能是因为示例文件不够长，不能组成完整的160 ticks的输入段所以必须要向文件附加零向量，而由于输入中含有太多零向量的原因所产生的音乐是无声的，但是这样的解释并不能解释其他文件出现的问题。

利用随机批次选择，杂音的周期比其他选择方式变形所产生的小得多。但是，偶尔会出现神经网络能够预测音乐但是却会停留在同一音高上的情况，这可能是因为使用随机选择批次，学习如何进行音乐过渡会比较困难。如果一首神经网络产生的歌曲有着悦耳的过渡结构，这有可能是由于在训练过程中随机发现了这种结构的变化，其他选择方式也能出现这样的情况。当预测音乐出现经常性的失败时，对于这样的情况很难解释。

许多预测产生的音乐（所有选择方式的LSTM神经网络变形）都存在极端值。虽然可以通过后期处理的方式使用不同的阈值来过滤其中一些阈值。值得注意的是，这些极端值的音高与预测产生歌曲的主要模式相近。没有音高比主模式中的最高音高高于20个音高。

这并不意味着神经网络已经开始学习如何对音调进行处理，但是它至少学习到在符合数据集中模式的歌曲中是不会出现极限音高的。

从结果来看，去除空向量能够产生较好的预测结果。虽然去除空向量能够让音乐结构看起来正确但是歌曲是否优秀还是取决于人们的主观判断。

### 3.6 调查

十位参与者被要求听9首歌曲样本：

1.定期选择，example.mid

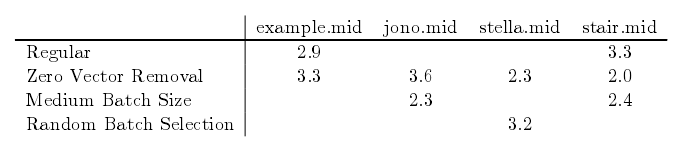
2.定期选择，stair.mid

3.去除空向量，example.mid

4.去除空向量，jono.mid

1. 去除空向量，stella.mid
2. 去除空向量，stair.mid
3. 中等序列大小的批次选择，jono.mid
4. 中等序列大小的批次选择Stair.mid
5. 中等序列大小的批次选择stella.mid

对于每一个样本，参与者被要求对于听到的音乐所产生的愉悦进行评级，从1到5，5为最愉悦的，每个样本的平均成绩：



去除空向量所得的样本有着最高的平均成绩，而中等序列大小批次选择产生的两首音乐平均得分最低。使用去除空向量预测方法最为悦耳，这与预测产生音乐的结构检验得出结果一致。

这之类似，神经网络利用大序列大小的选择方式未能产生大量的音乐，而且它通常产生的音乐都是杂音。

### 4 讨论

在LSTM神经网络中实现了表示MIDI文件的格式，但是只能处理一个通道（不同的MIDI通道通常代表不同的乐器）。为了能够处理多个通道的数据，该算法需要重新创建一个更大立体单元，或者一个使用二维单元的值然后构建一个系统将不同的值（速度（响度））映射到不同的通道内。有着如下的实现方式：让每个通道接收它的128个单音，其中较高的通道的值高于前一个通道128。然后通过除以128找到映射的通道，用128的模运算来计算。

MIDI编码系统不能够处理连续相同的速度和音调，虽然这样的情况是非常罕见的（在人声音乐中是不可能出现的，因为没有人能够在表演的完全在同一时刻开始或停止演奏乐器），但是在一些电子音乐中它又是确实存在的。由于算法的限制，这些音调会被在编码过后变成连续的音调。有如下解决方案：预处理MIDI文件，检查对于MIDI文件的操作，手动设置序列的速度值略高于或低于上一序列。

虽然对于不同变形的批选择进行了研究，但是在研究过程中忽略了批处理可变步长的使用。随机批次选择不使用步长，其他使用的步长大小为1。在实验的早期，1的步长似乎比更大的步长更成功，从而导致尝试其他步长变成了日后的考虑。然而，在不同的数据集中进行更大步长的实验是有意义的。特别是对于重复性很高的结构，这可能是非常有效的，因为这样选择批次可能能跳过这一部分，它也能够降低音乐生成停留在循环中的可能性，虽然这样做有学习过少的风险。同时值得注意的是，较长的步长能减少训练批次的总数，这可能会大大减少训练的时间。

后期处理在滤除噪声和平滑速度值方面证明是有用的。然而，在许多预测的过程中，一些较短的音调（通常只有1 ticks）偶尔会在整个预测产生的音乐中出现。这些音调几乎总是是跑调的。虽然当前方法的后期处理在实验中能够获得较好的效果（增加最小阈值，增大允许最高和当前速度值相差的百分比），但是让后期处理再加上一个新的步骤可能会更好。这个新的步骤将包括如何确定当前的音调，并计算与当前的音调相比，预测所产生的非空向量中的音调是否存在跑调的序列。如果这样做的话，应该要进行一个检查来确定这个向量是否属于一个新的序列，而且并没有跑调。如果两个检查都失败了，那么将速度值设置为0。这样的操作需要non-trivial来确定一个音调在序列中有没有跑调。然而，这需要对音调检测有一些研究。「15」对于第二次检测，确定一个向量是否属于一个新的序列可能需要对预测数组进行进一步检查，由于这些数据都将在后期处理中进行处理，这就意味着进入数组的向量需要是可用的。

在后期处理中并不需要做上述的工作，将音调检测作为今后研究自动生成音乐的一个重点。如果一个神经网络可以使用音调检测来对数据集中的音乐进行训练，神经网络能随机生成未跑调的音乐，然后利用所学的转换方式将这些单一的音调转换为旋律和音乐。

虽然增加序列长度和使用随机批量选择来收集具有相应标签向量的训练序列的结果并不令人满意，但增加训练集的大小可能会让结果变得更加优秀。由于数据集中只有74个文件，所以在较长序列中查找结构是非常困难的。事实上，更大的数据集能够改进所有训练方式的结果。对与神经网络来说用74个文件的数据集来学习音乐结构还是太小了。如2.1节所解释，选择一个较小的数据集是由于当使用同一乐队或者艺术家的时候，这些歌曲的风格之类的是类似的。然而，使用较大的数据集是有意义的。

一种使用二进制速度值批次选择的神经网络变形在实验中被简单地测试了。所有非空向量的速度值都被设置为1。当时这样做的原因是由于可以降低训练中所产生的损耗。但是由于没能够成功地预测产生音乐，所以这样方式被废弃了，但是研究这个方法并不是毫无意义的，因为神经网络是将数据进行分类而非递归，一个音调就是一个类。这样神经网络就可以进行更多关于损失函数和激励函数的实验了。

同样，不同神经网络的变形能够取长补短相互结合。虽然在本文中并没有先去除编码数组的空向量再进行随机选择批次，但相信这一定能够产生一些有趣的结果。更多的实验能够作为未来实验的基础，产生更多的思考。

考虑到音乐的主观性和将小样本推广到更大人群的问题，在今后的研究中，也许有必要进行重复调查，或者创建一个参与人数更多的新调查。

### 5 结论

据推测，将MIDI格式编码的歌曲数据集正确转换为可以用于训练递归神经网络的数据，那么就能通过任何MIDI文件产生复调音乐。此外，处理转换后的数据集的方式可能会影响生成的音乐的质量。

使用一个通过74首 Led Zeppelin MIDI格式的歌曲数据集转化成的数组对三层的LSTM神经网络进行训练。训练结束后，程序开始预测输入的歌曲之后应该如何继续。通过使用四种不同选择数组内数据的方式来验证怎样使用神经网络来生成音乐是最佳的。这些方法中最为成功的方式是去除数据集中歌曲无声的部分，用这种方式产生的音乐结构最让人感觉到舒适、优美，这与通过调查得出这种方式比其他方式得到的音乐更好的结论相一致。然而，所产生的音乐要是存在极端值存在的噪音和易重复性，这些都容易让歌曲在调查中收到不太好的评价。作为此篇论文的主题，固然通过LSTM神经网络能够产生通过短音乐段作为输入产生Tick作为时间格式的MIDI文件，但是在宣布所有音乐家失业之前，该算法还需要进一步改进。

### 参考文献

[1] Zaripov, R. X. (1960). Об алгоритмическом описании процесса сочинения музыки (On the algorithmic description of the process of composing music). In Доклады АН СССР (Vol. 132).

[2] Lonsdale, A. J., & North, A. C. (2011). Why do we listen to music? A uses and gratifications analysis. British Journal of Psychology, 102(1), 108-134.

[3] McCormack, J. (1996). Grammar based music composition. Complex systems, 96, 321-336.

[4] Fox, R., & Crawford, R. (2016). A Hybrid Approach to Automated Music Composition. In Artificial Intelligence Perspectives in Intelligent Systems (pp. 213-223). Springer International Publishing.

[5] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural Networks, 61, 85-117.

[6] Chen, C. C. J., & Miikkulainen, R. (2001). Creating melodies with evolving recurrent neural networks. In Neural Networks, 2001. Proceedings. IJCNN’01. International Joint Conference on (Vol. 3, pp. 2241-2246). IEEE.

[7] Eck, D., & Schmidhuber, J. (2002). A first look at music composition using lstm recurrent neural networks. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 103. Chicago

[8] Franklin, J. A. (2006). Recurrent neural networks for music computation. INFORMS Journal on Computing, 18(3), 321-338.

[9] Sak, H., Senior, A. W., & Beaufays, F. (2014, September). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In INTERSPEECH (pp. 338-342).

[10] Johnston, L. (2016). Using LSTM Recurrent Neural Networks for Music Generation.

[11] Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[12] Dauphin, Y. N., de Vries, H., Chung, J., & Bengio, Y. (2015). RMSProp and equilibrated adaptive learning rates for non-convex optimization. arXiv preprint arXiv:1502.04390.

### 附录

**输入的文件**

下图为用于预测的四个输入文件

实例文件（example.mid）

MIDI音轨仅有90 ticks，也就是说在使用序列大小大于90时必须向该文件内部附加空向量。（）

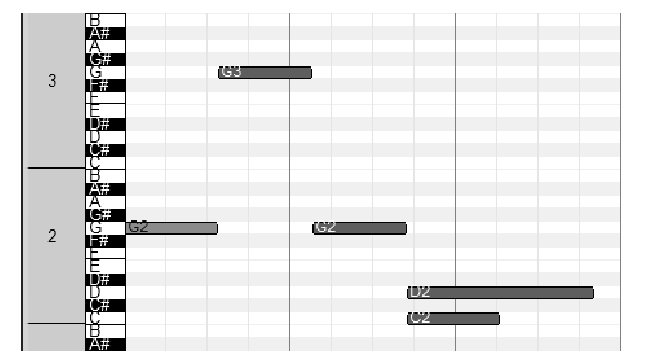


图25：example.mid输入文件，每一条表示8 ticks。

Let The Light Shine In (jono.mid)

由于这是一首真正意义上的歌曲，该首歌曲的音轨会持续很久并且比图片所表示的要长得多。但是，展示它的重复部分也是极为意义的。

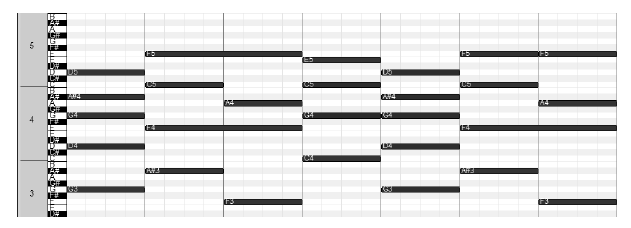


图26：输入文件jono.mid，在此很难看见单独的条形图，但是被灰色竖线所分割的部分包含着32 ticks。

Stella by Starlight (stella.mid)

这是一首无重复复杂旋律的JAZZ。如论文中所示，使用了前几ticks的部分，但是最初的几ticks中包含了许多不同的音调。

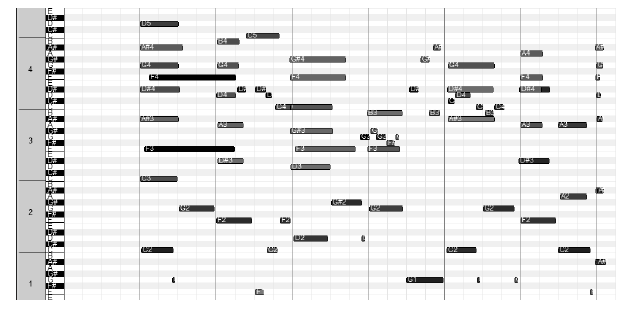


图27：输入文件stella.mid，前32ticks是无声的。

Stairway to Heaven (stair.mid)

此输入文件是数据集中的部分，也是另外一首无重复的音乐。如下图所示，这首歌有些复杂。

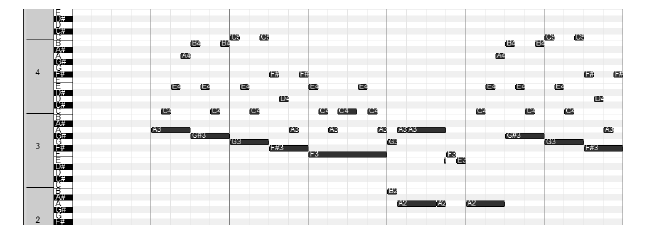
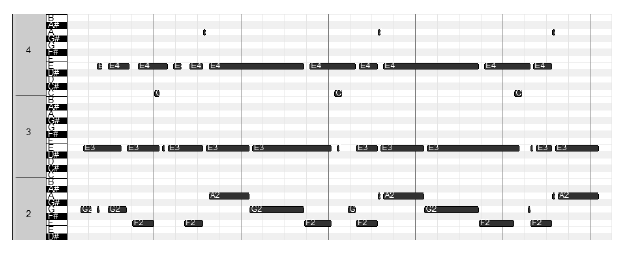


图28:输入文件stair.mid，同样前32tick是无声的。

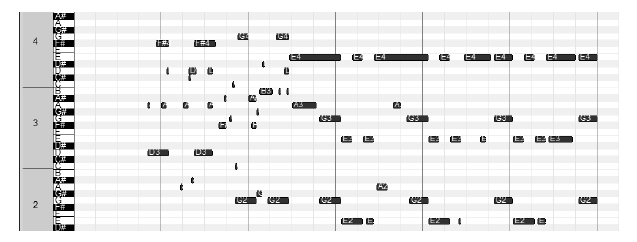
### 用于调查的音乐样本

下面九个样本是由神经网络所产生的并用于调查人们对于听到的音乐的满意度。

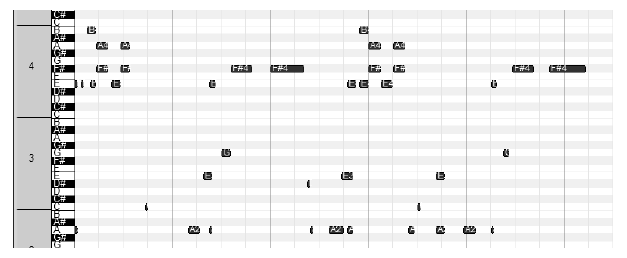
常规/标准的选择批次用example.mid所产生的survey1.mid



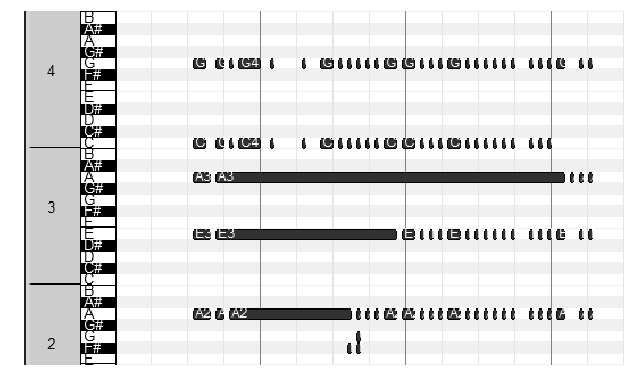
去除空向量用example.mid所产生的survey2.mid



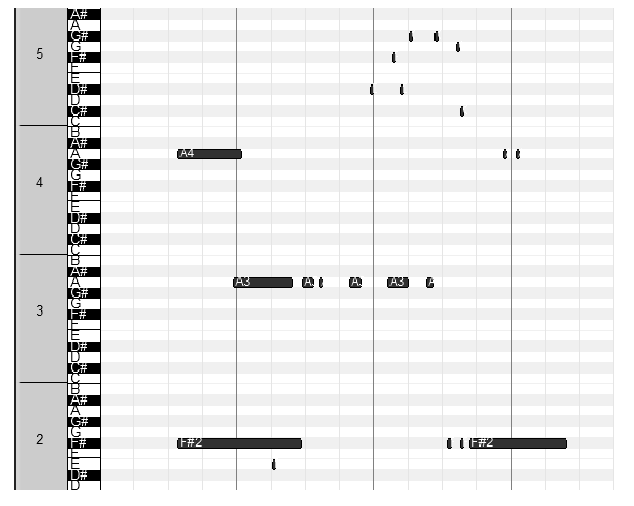
去除空向量用jono.mid所产生的survey3.mid



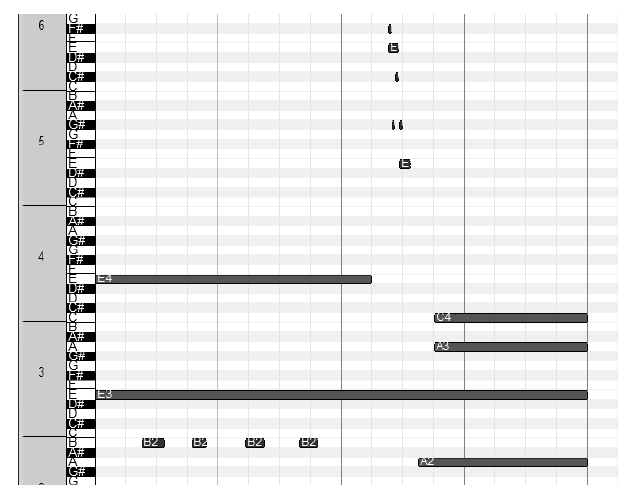
中序批量选择 用jono.mid所产生的survey4.mid



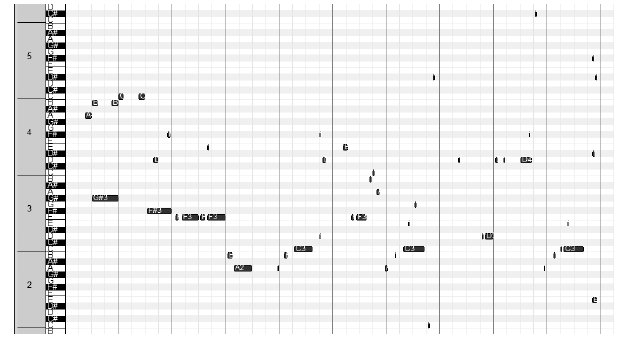
去除零向量 用stell.mid所产生的survey5.mid



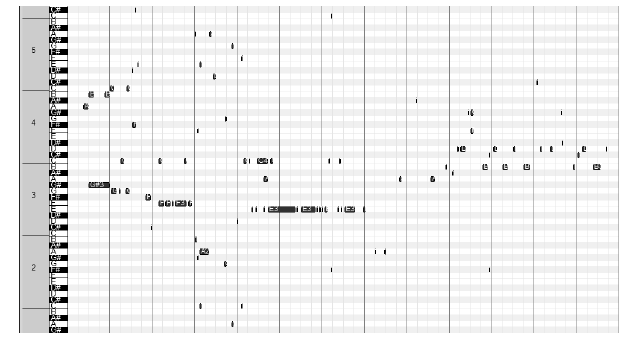
随机批次选择用stella.mid产生的survey6.mid



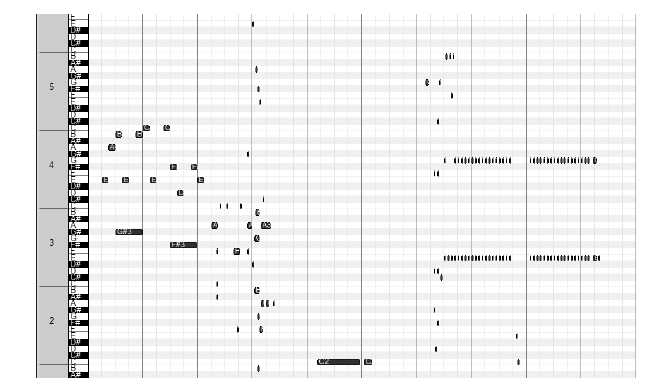
常规/标准的选择批次用stair.mid所产生的survey7.mid



去除零向量用stair.mid所产生的survey8.mid



中序批量选择用stair.mid产生的survey9.mid



1. 一个四分音符的音长为480 Ticks，则480 Ticks为1秒，单个Tick的时间长度就是（1/480）秒。 [↑](#footnote-ref-0)
2. Hall, G. (2016). Python Midi: A Library for MIDI in Python. https://github.com/vishnubob/python-mid [↑](#footnote-ref-1)
3. Olah, C. (2015). Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [↑](#footnote-ref-2)
4. Chollet, F. (2016). Keras: Deep Learning library for Theano and TensorFlow. http://keras.io// [↑](#footnote-ref-3)
5. DAS-4: A six-cluster wide-area distributed system for researchers. http://www.cs.vu.nl/das4/ 13 [↑](#footnote-ref-4)