|  |  |
| --- | --- |
| ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ Государственное бюджетное общеобразовательное учреждение города Москвы **«Школа № 1533 «ЛИТ»** | |
| **ВЫПУСКНОЙ ПРОЕКТ** | |
| учащихся группы 10.5 и 10.4  **Колышкина Дмитрия Андреевича, Кромского Петра Владимировича, Бронникова Леонида Владимировича**  (Специальность: программирование) | |
| **ГЕНЕРАТОР ДЖАЗА НА ОСНОВЕ ML ПО ДЕСЯТИЛЕТИЯМ** | |
|  |  |
| Заказчик и руководитель | **Завриев Николай Константинович** |
| Консультант | **Балаян Степан Хосровович** |
| Москва  2021 | |

**Содержание**

**Введение 3**

**актуальность, целевая аудитория, аналоги, постановка задачи 4**

**Введение 4  
Постановка задачи 4  
Аналоги 5**

**решение 7**

**Анализ предметной области 7**

Джаз 7  
Нейросети и основные методы 7

**Программная реализация 9  
Ход работы 10**

Июнь – Сентябрь 2020 10  
Октябрь – Декабрь 2020 10

Январь 2021 11

Февраль – Март 2021 11

Апрель 2021 11

Май 2021 12  
Июнь 2021 12

**Сложности на этапе разработки 14**

**результаты 15**

**Изучено 15  
Результаты эксперимента 15  
Направления дальнейших разработок 18**

**Список Источников 20  
Приложение 22**



**Введение**

ML и Нейросети – вот два главных слова, вызывающие восторг как у многих идейных заказчиков, так и у самих программистов. Если подумать, нейросети уже давно стали часть нашей жизни. Вы открываете Яндекс, Google или любой другой поисковый сервис, и вы видите рекламу, на ходу подстраивающуюся под ваши интересы. Вы включаете «Открытия недели» или «Новою музыку» в Spotify или Apple music, и слышите именно то, что хотели бы услышать. Но рамки использования нейросетей намного больше – большие компании, пытающиеся понять что понравится их целевой аудитории и камеры с системой распознавания лиц тому пример.  
Нейросети уже давно научились распознавать картинки, создавать новые и уже научились генерировать недостающие их части. Они научились распознавать речь и осознанно отвечать на нее. Они научились дописывать за пользователем музыкальные фразы, а также создавать аккомпанемент. Но есть нейросети, самостоятельно создающие музыку – это что-то относительно новое.  
Джаз, в свою очередь - огромное культурное наследие. Появившись в 20-х годах прошлого столетия, сегодня он уже стал неотъемлемой частью культуры. За сотню лет он много раз изменялся, в каждом десятилетии приобретая самые разные формы. В каждом из десятилетий были свои новаторы и основоположники жанров, свои гениальные композиторы и виртуозные инструменталисты. Джаз - прародитель многих жанров, от фанка до хард-рока, даже сегодня он вдохновляет музыкантов, художников, писателей. Каждый человек хоть немного слышал о Луисе Армстронге или Фрэнке Синатре.   
Наш проект - музыковедческий эксперимент, цель которого - выяснить, есть ли композиционная разница между джазом разных десятилетий, и попытаться формализовать эту разницу, или показать, что это самообман, и все эти временные отрезки человек определяет по второстепенным факторам, вроде качества записи.

**Актуальность, целевая аудитория, аналоги, постановка задачи**

**Введение**

В последние годы популярным и востребованным направлением в искусствоведении стало применение математических методов к исследованию произведений искусства (поэзии, живописи и т.д.). Однако таких исследований, касающийся музыки еще не так много, и даже те, что уже есть, не помогают в нашей проблеме. Именно поэтому мы решили проводить данный эксперимент при помощи нейросети.   
Задачу можно считать актуальной, так как все известные аналоги не ставят перед собой музыковедческий целей, и потому не позволяют провести наш эксперимент.  
Наш проект не имеет четкой целевой аудитории. Он предназначен, в первую очередь, для музыковедов, которые желают изучить музыку в самых разных ее проявлениях, а также просто в развлекательных целях. Не стоит забывать и о тех людях, что захотят использовать наш проект аналогично Mubert, т.е генерировать музыку для последующего ее использования в личных целях.

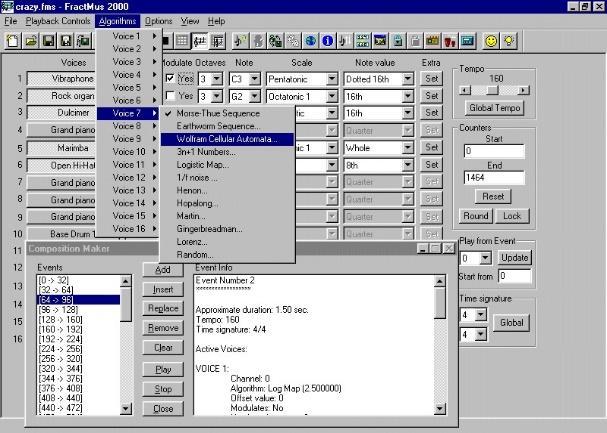
**Постановка задачи**

Наша цель делится на несколько этапов:

1. Создать нейросеть, способную генерировать клавишные партии, опираясь на
   1. Десятилетие (1930–1999)
   2. Темп
   3. Тональность
2. Обучить нейросеть для каждого из десятилетий
   1. Найти материал в формате MIDI
   2. Разбить найденные композиции по времени выбора
   3. Убрать слишком экспериментальные и неподходящие
   4. Определить подходящее количество эпох для каждого десятилетия
3. Сделать вывод о результате, полученном после обучения
   1. Можно ли результат отнести к эпохе на основе которой он был создан, опираясь на звучание?
   2. Если файл можно отнести к нескольким эпохам, то что их объединяет?
   3. Схожи ли полученные в результате генерации композиции между собой?
   4. Удачные и неудачные композиции

**Аналоги**

Поскольку работа нашего проекта базируется на генерации музыки, имеет смысл провести сравнительный анализ нашей разработки с другими генераторами на основе нейросетей или при помощи других методов. Заметим однако, что все нижеперечисленные проекты не несут в себе музыковедческой компоненты.

**FractMus 2000** – эксперимент, который генерирует музыку, основываясь на математических моделях. Является уникальным в своем роде проектом, но, к сожалению, он был заброшен, и последняя его версия адаптирована только для Windows 7  
  
**MuseNet** – нейросеть от OpenAI. Она не имеет приложения, так как является, в первую очередь, экспериментом. Генерирует музыку, основываясь на «начальном» и «желаемом» исполнителе (например Led Zeppelin в стиле Селин Дион). На данный момент разработка приостановлена, так как компания занимается другим проектом.  
**JukeBox** – еще один проект от OpenAI, который и пришел на замену MuseNet. JukeBox, в отличие от своего предшественника, генерирует несуществующие песни в стиле конкретного существующего исполнителя, и справляется с этим настолько хорошо, что авторы на полном серьезе поднимают проблему авторского права. Его единственная проблема – фоновый шум, который затрудняет прослушивание композиций.  
**Mubert** - проект, генерирующий музыку в определенном стиле длинной в определенное время. Сайт (где и находится весь функционал) рассчитан на людей, которым нужна музыка в своих проектах, но они не желают связываться с авторами и авторским правом. Не совсем понятно, является ли проект нейросетью или своеобразным диджеем уже заранее записанных композиций.Изображение выглядит как текст

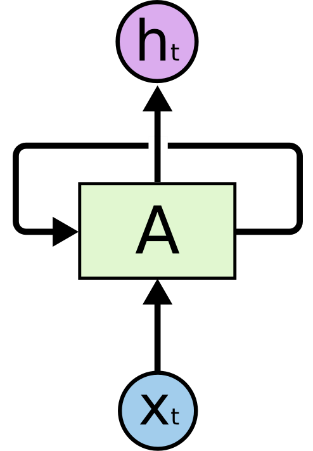
Автоматически созданное описание

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **FractMus 2000** | **MuseNet** | **JukeBox** | **Mubert** | **Наш проект** |
| Генерация во множестве стилей | +- | ++ | ++ | +- | - |
| Внимание к джазу | - | + | + | - | ++ |
| User Friendly | - | - | - | ++ | + |
| Нет проблем с копирайтом | + | - | - | + | + |

**Решение**

**Анализ предметной области**

**Джаз**Джаз – один из самых популярных жанров в мире. Начав свое существование около сотни лет назад, сегодня он является огромным культурным наследием. Жанров самого джаза уже существует огромное количество, и оно лишь увеличивается. Разграничить эти жанры сложно, ведь каждый из них является одновременно смесью предыдущих и чем-то новым. Чтобы хоть как-то понять, как классифицировать эти поджанры, и как же обучить нейросеть «различать» их, мы решили выделить следующее:  
1. Десятилетие  
2. Основная волна десятилетия  
Эти два критерия были приняты, так как десятилетие – одновременно достаточно узкий промежуток, чтобы быть уникальным, но при этом достаточно большой, чтобы мы нашли достаточно материала для нейросети. Основная волна – способ понять, какому жанру отдавать предпочтение при составлении датасета. Он менее значим, чем десятилетие, но игнорировать его невозможно. Свинг 40 и Кул джаз 60 будет сильно отличаться, но важно дать нейросети и оба произведения, а потом указать на их различие.  
Вот что у нас в итоге получилось:  
**30** – большинство джазовых стандартов [Duke Ellington](https://open.spotify.com/artist/4F7Q5NV6h5TSwCainz8S5A?si=tBqudJR-S-KTg7v2Z2mARg&dl_branch=1) - [Take the A Train](https://www.youtube.com/watch?v=cb2w2m1JmCY)**40** – свинг и бибоп [The Glenn Miller Orchestra](https://open.spotify.com/artist/6N1ycRllcq7fwHMdeTQD1z?si=k5AxNAq1Si6ftlzWSq8BzA&dl_branch=1) - [In the Mood](https://www.youtube.com/watch?v=6vOUYry_5Nw)**50** – Кул джаз [Miles Davis](https://open.spotify.com/artist/0kbYTNQb4Pb1rPbbaF0pT4?si=PqUCeOMmQyqHoDsHB01A1A&dl_branch=1) - [So What](https://www.youtube.com/watch?v=zqNTltOGh5c)**60** – Фри и Соул джаз [Buddy Rich](https://open.spotify.com/artist/1pVtwG5Up1OZOEpSHJ4AAs?si=ObOvXSrRS9CNqxWtEo2d0Q&dl_branch=1) -[Soft Wind](https://www.youtube.com/watch?v=rX2qPyFI2P8)**70** - Фьюжн [Weather Report](https://open.spotify.com/artist/162DCkd8aDKwvjBb74Gu8b?si=YHbMhKCgSbmQRFbIErAXHQ&dl_branch=1) - [Teen Town](https://www.youtube.com/watch?v=lSUk8bSVHYc)**80** – Фьюжн и Эйсид джаз [Weather Report](https://open.spotify.com/artist/162DCkd8aDKwvjBb74Gu8b?si=YHbMhKCgSbmQRFbIErAXHQ&dl_branch=1) - [Volcano For Hire](https://www.youtube.com/watch?v=JAIMdokWDgg)**90** - Эйсид джаз [Jamiroquai](https://open.spotify.com/artist/6J7biCazzYhU3gM9j1wfid?si=80e388b244ec40af) - [Space Cowboy](https://www.youtube.com/watch?v=OPkjnRIdQXQ)Безспорно, на 90х джаз не заканчивается, и есть как множество современных исполнителей в уже не новых жанрах (Victor Wooten), так и множество исполнителей со своим абсолютно уникальным стилем (исполнители в жанре Dark Jazz). **Нейросети и основные методы**Для реализации поставленной задачи были выбраны нейросети, так как именно они лучше всего подходят для генерации чего-либо нового. Использование каких-либо строгих математических алгоритмов хоть и возможно, как нам доказал FractMus 2000, но крайне сложно и менее эффективно. В данном случае мы используем модель s2s (sequence to sequence), которая обрабатывается при помощи РНС (рекуррентных нейросетей). Так как музыка – последовательность нот, а не случайный их набор, рекуррентные нейронные сети, которые имеют некое подобие динамической памяти – то, что нам нужно.  
На изображении ниже находится схема РНС с одним слоем, получающей на вход последовательность и дающая на выход последовательность. Рекуррентные сети, в отличие от обычных, имеют обратную связь: синапс, который передает информацию от одного шага сети к другому.



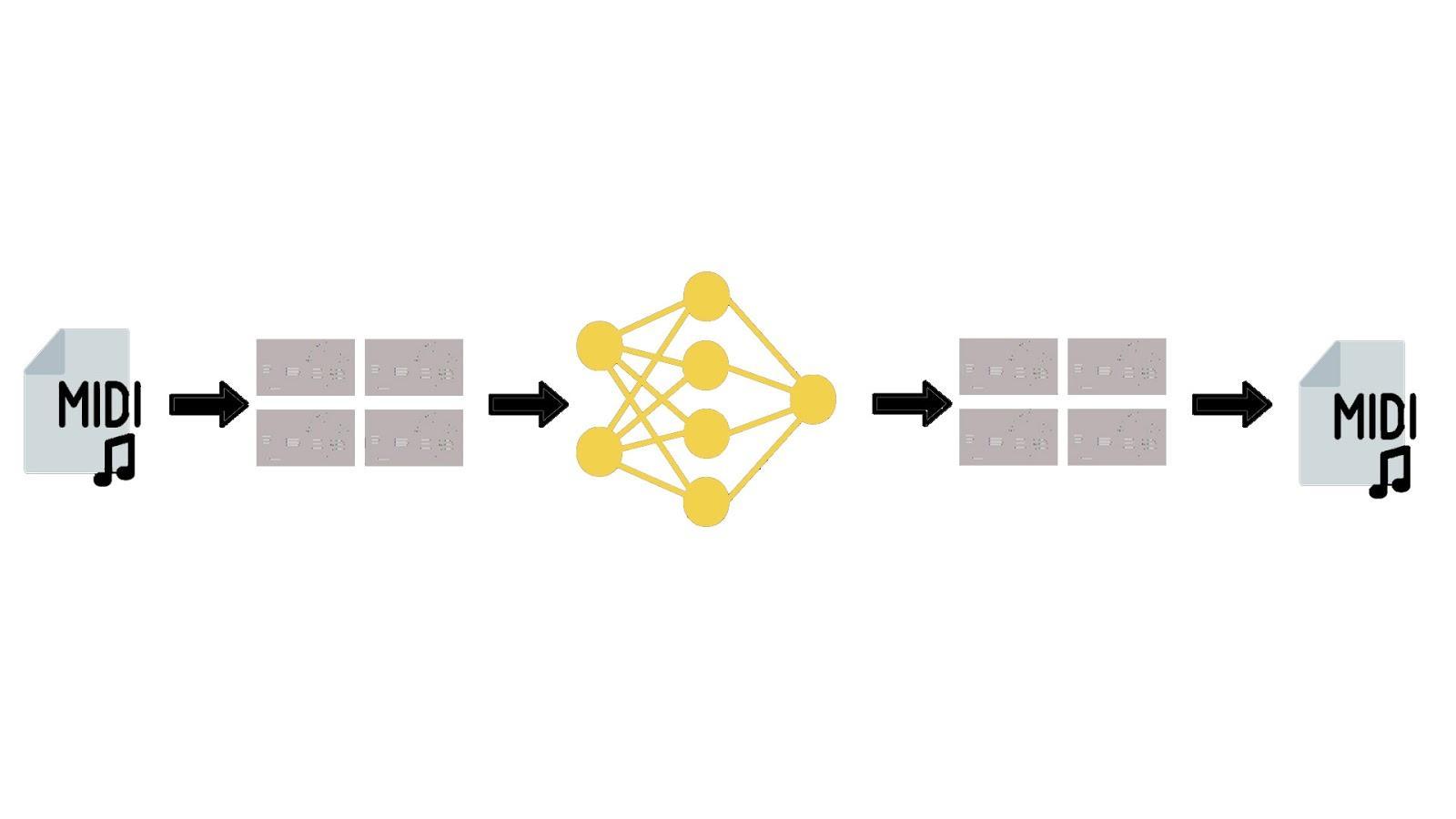
Так эта же схема выглядит в развернутом состоянии:Изображение выглядит как текст, датчик

Автоматически созданное описание

К сожалению, несмотря на возможность составления кратковременных зависимостей, РНС имеет проблемы с составлением долговременных. Нейроны помнят результаты ряда предыдущих нейронов, но более старые нейроны почти не оказывают влияния на них.  
Чтобы генерировать музыку, наш проект использует достаточно интересный метод. Программа преобразовывает MIDI файл в ряд изображений, где изображены ноты, их длительность и громкость, а потом, генерирует на основе этих изображений собственные, которые потом преобразует обратно в MIDI. Именно для этого в проекте и используются библиотеки компьютерного зрения. Пример того, как выглядит фрагмент MIDI, преобразованный в картинку:

Изображение выглядит как легкий, темный

Автоматически созданное описание  
Ниже краткая схема того, как работает программа:



**Программная реализация**

Для реализации этого проекта были использованы несколько языков и сред программирования.  
Для нейросети был использован Python и PyCharm, так как Python – самый удобный и адаптированный язык для написания нейросетей. Изначально так же использовался Jupyter Colaboratory и Jupyter Notebook, но позже мы отказались от их использования из-за сложностей переноса частей программы из одной среды в другую.  
Использованные библиотеки:

* Numpy
* Random
* Mido
* PyAudio
* PyGame
* TensorFlow
* Keras
* Theano
* CV2
* Matplotlib

**Ход работы**

Начав поиски в июне 2020, на данном этапе мы можем разделить этапы работы на несколько частей:

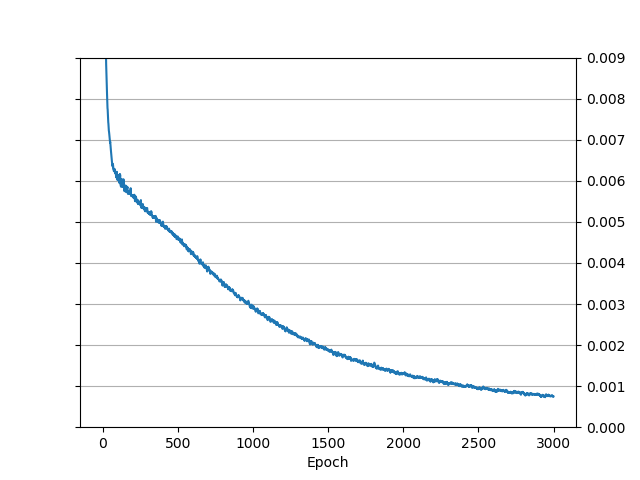
**Июнь — Сентябрь 2020**После того, как мы поняли “основные тезисы” нашего будущего проекта, мы принялись изучать предметную область. Мы начали искать методы, с помощью которых сможем реализовать, а также тщательно изучать историю джаза и его культурный след. Почти сразу стало понятно, что основной метод будет реализован с помощью ML, но конкретных выводов еще сделано не было. За данный период был изучен Python, а также основные методы ML, в частности, нейросетей.

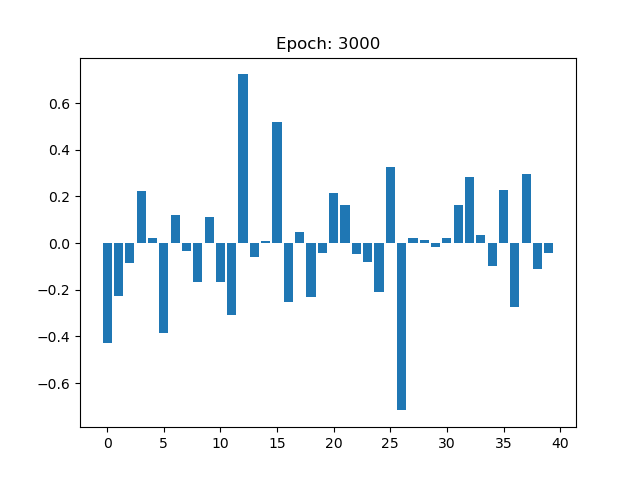
**Октябрь — Декабрь 2020**Во время изучения предметной области мы нашли несколько людей, которые занимались темами, близкие к нашим. После отсечения не подходящих нам вариантов мы остановились на трех похожих, но различающихся вариантах:  
1. Модель способна генерировать игровую музыку. Она делает это успешно, так как автор уже учел многие проблемы, с которыми столкнулись авторы других моделей. Хотя выбор подразумевает внесений большого ряда изменений в модель, мы сможем избежать большого ряда проблем.  
2. Модель генерирует джаз, и использует те же средства, что и первый претендент, но из-за ряда отличий работает намного менее эффективно. Хотя изначально эта модель больше всего располагала к себе, после более тщательного изучения мы поняли, что она имеет слишком низкую работоспособность.  
3. Прекрасно работающая, уже организованная нейросеть, генерирующая свинг. Отмели почти сразу – почти вся документация написана на немецком, и никем не переведена.  
В итоге мы остановились на первом варианте. Интересно, что впервые эта модель была использована именно как генератор изображений, в частности она генерировала лица несуществующих людей, опираясь на небольшой датасет.

**Январь 2021**В течение этого месяца наша команда столкнулась с рядом проблем при установке модели, так как она работала на Python 2, и к концу месяца смогли запустить модель

**Февраль — Март 2021**В этот период мы начали адаптировать модель под цели нашего эксперимента, а также составлять датасеты. К концу периода было [сгенерировано первое произведение](http://midiplayer.ehubsoft.net/?state=%7B%22ids%22:%5B%221zYV3WS_GX-JE8WynGu6vzv2tJ_6t8Z0m%22%5D,%22action%22:%22open%22,%22userId%22:%22106231975263567504203%22,%22resourceKeys%22:%7B%7D%7D), а также составлен датасет для 1930х годов.

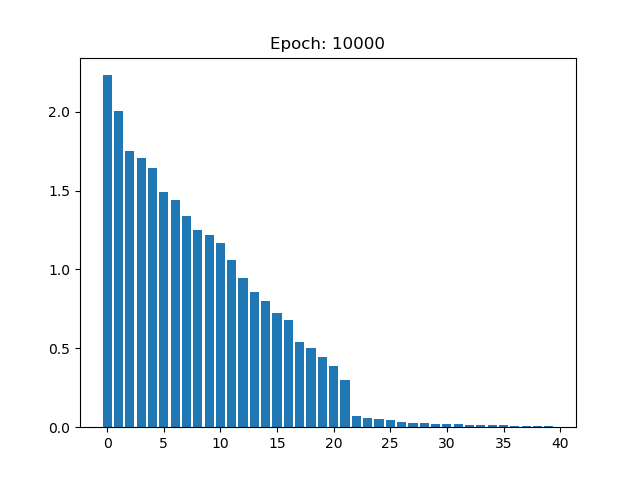
**Апрель 2021**Мы продолжили составлять датасеты и обучать нейросеть. Составлены датасеты для 1940-х и 1970-х

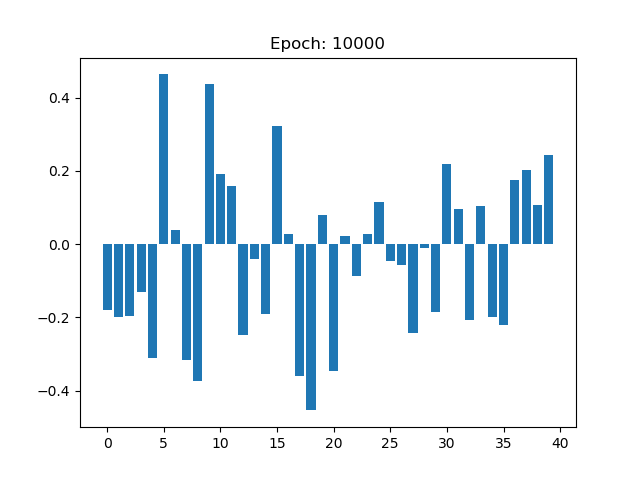
**Май 2021**  
В мае мы столкнулись с рядом проблем, связанных с тем, что программа работает на Python 2, и начали работу по переносу программы на Python 3. Ниже график потерь на 3000 эпох после переноса нейросети на Python 3 и график работы латентно-семантического анализа.  




**Июнь 2021**Составлены датасеты для 1940-х, 1950-х, 1960-х, (доделать 80 и 90). Устранения неполадок в работе нейросети. Определено оптимальное количество эпох для всех десятилетий - 2000  
Примеры сгенерированных композиций:

* [Пример 1 (1930е)](http://midiplayer.ehubsoft.net/?state=%7B%22ids%22:%5B%221CLjkaVBBU-vStgnFBsdFXLEiOjQNxOJN%22%5D,%22action%22:%22open%22,%22userId%22:%22106231975263567504203%22,%22resourceKeys%22:%7B%7D%7D)
* [Пример 2 (1940е)](http://midiplayer.ehubsoft.net/?state=%7B%22ids%22:%5B%22187QWenuoM8bdqyidJORG0ftx6uxUps6s%22%5D,%22action%22:%22open%22,%22userId%22:%22106231975263567504203%22,%22resourceKeys%22:%7B%7D%7D)
* [Пример 3 (1950е)](http://midiplayer.ehubsoft.net/?state=%7B%22ids%22:%5B%221b-u-YUlXPBY0s35wMi6Xg_ppbJAdTpd_%22%5D,%22action%22:%22open%22,%22userId%22:%22106231975263567504203%22,%22resourceKeys%22:%7B%7D%7D)
* [Пример 4 (1970е)](http://midiplayer.ehubsoft.net/?state=%7B%22ids%22:%5B%221T0V0UR0_O4izf52SZgm-yJm8tW108Z3P%22%5D,%22action%22:%22open%22,%22userId%22:%22106231975263567504203%22,%22resourceKeys%22:%7B%7D%7D)

Метод главных компонент при 10000 эпох:   


Метод латентно-семантического анализа при 10000 эпох:  


**Сложности на этапе разработки**

**Отсутствие документации у модели**Когда мы начали пытаться изменить модель под нужды эксперимента, мы столкнулись с полным отсутствием документации. Сам автор модели поместил пункт “написание документации” в раздел “//TODO” еще в 2018 году.

**Перенос на Python 3 и проблема с установкой библиотек**Изначальный код программы находился на языке программирования Python 2, и использовал ряд устаревших функций и костылей. Один энтузиаст смог в конце августа самостоятельно переписать модель на Python 3, однако это подразумевало и то, что мы будем вынуждены самостоятельно изменять уже новую версию сети. Стоит добавить, что из-за плохой документации новой версии, возникли проблемы с несоответствиями версий библиотек. Отдельного стоят не очень популярные библиотеки-утилиты, не все версии которых были доступны для установки, так как их авторы выкладывают только последние стабильные версии.

**Материал для датасетов**Неожиданно для нашей команды, найти достаточное количество MIDI файлов для нашего проекта, оказалось сложной задачей. Большинство сайтов с MIDI музыкой имеет очень скудный набор в разделе “джаз”, если вообще имеют таковой. Хотя мы и нашли несколько сайтов, подходящих нам, этого все равно недостаточно. Стоит отметить, что конвертирование WAV и MP3 в MIDI - не лучшее решение, так как после конвертирования все инструменты в произведении сжимаются до одной дорожки, и итоговый звук превращается в кашу ([пример](http://midiplayer.ehubsoft.net/?state=%7B%22ids%22:%5B%2213qRuYQDlgcmFjTPQ_wH3Hg2deN-MZuQm%22%5D,%22action%22:%22open%22,%22userId%22:%22106231975263567504203%22,%22resourceKeys%22:%7B%7D%7D) (ориг. Jamiroquai - [Virtual Insanity](https://www.youtube.com/watch?v=4JkIs37a2JE)))

**Мелодия и аккорды**Некоторые файлы содержат либо несколько разных партий фортепиано, либо одну множество раз гармонизированную. Если такие файлы оставлять неизменными, результат генерации сети станет хуже, так как значительно повысится количество “лишних” нот.

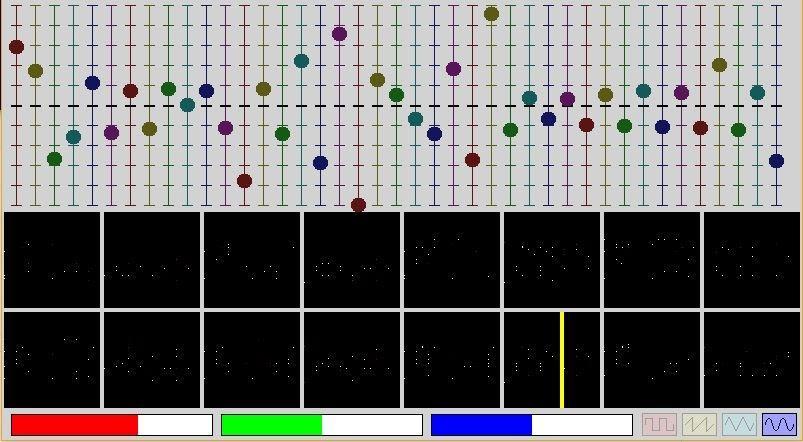
**Результаты**

**Изучены:**

* История джаза
* Python 2 и 3
* Внимательно изучены принципы работ библиотек TensorFlow и Keras
* Методы машинного обучения
  + В общем
  + Нейросетей
  + Рекуррентных нейросетей
* Реализована модель
* Собраны датасеты
* Сгенерированы фортепианные партии

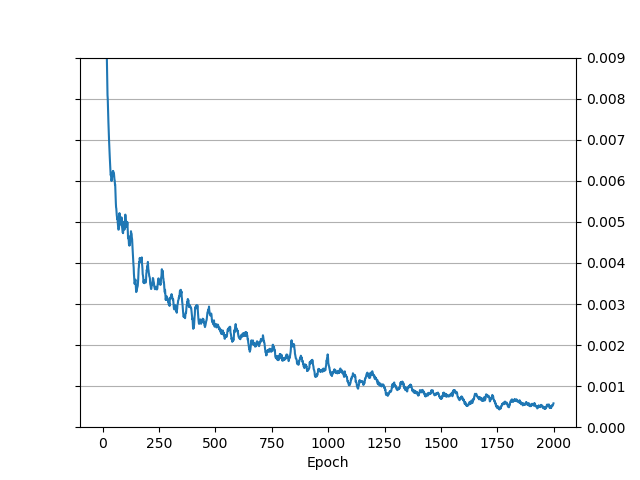
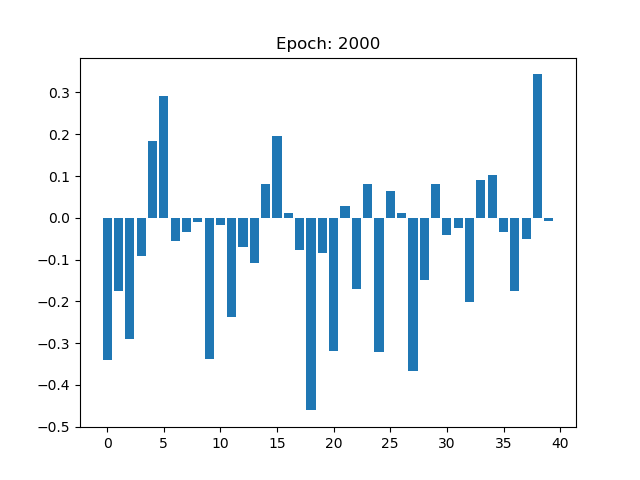
**Эксперимент показал, что:**

* Нейросеть способна генерировать произведения в разных тональностях
* Нейросеть способна улавливать динамику
* Нейросеть способна создавать произведения в разных размерах, но не изменять их
* Даже результаты внутри одного десятилетия могут отличаться, хотя их все еще можно отнести к одной эпохе
* Соседние десятилетия (например 1930-е и 1940-е) отличить с первого раза не просто, но при внимательном слушании они отличаются. Стоит заметить, что в данных десятилетиях нет активного развития мелодии, но при этом много обыгрываний последовтельности аккордов
* Несоседние десятилетия (1930-е и 1970-е) разительно отличаются. В отличие от 30-х и 40-х, 70-е отдают предпочтение мелодии и развитию композиции, использую аккорд как средство, а не основу.

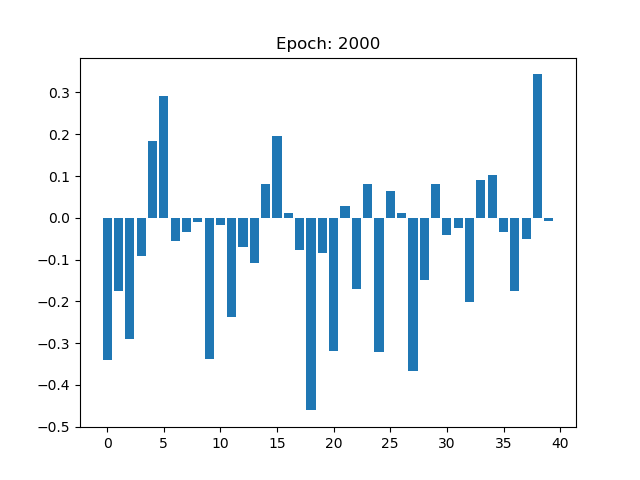
Из вышеперечисленного можно сделать вывод, что джаз действительно развивался, но наибольшие скачки наблюдались в 1970-х. В 1930-х и 1940-х наблюдается очень близкое сходство, но если результат генерации в первом случае напоминает классическую музыку, хотя очевидно ей не является, то результат сороковых почти идеально похож на то, каким мы его себе представляли. Как итог - нейросеть для 1930-х генерирует джазовые стандарты, похожи на стандарты начала 30-х годов прошлого века, а для 1940-х - бибоп.  В свою очередь, 1970-е походят на 1980-е, но тут разница уже намного хуже заметна, и отличить их на слух удается не всегда. Ниже примеры работы и графики вычислений конечной версии  
  
  
Пример работы программы:  
  
  
Пример сгенерированного файла в MIDI:  
Изображение выглядит как текст, стена, белый

Автоматически созданное описание

График потерь:

  
  
  
График работы латентно-семантического анализа:  


Методы главных компонент:



**Направления дальнейших разработок**

* Генерация аккомпанемента в соответствии с десятилетием и выбранными инструментами
  + Гитара (если позволяет временной промежуток)
  + Барабаны
  + Контрабас/Бас-гитара
  + Духовые
* Написание и расширение функционала плагина
  + Возможность изменения тональности
  + Возможность настраивать размеры тактов
* Пополнение датасетов
* Добавление возможности генерации разных размеров такта внутри одного произведения
* Добавление в сеть эволюционной модели
* Реализация LSTM
* Добавление в программу возможность обрабатывать MIDI как математический ряд, а не изображение
* Реализовать алгоритм, с помощью которого пользователь сам сможет загружать MIDI файл в программу, а та будет возвращать ему десятилетие, в котором написано это произведение, и добавлять в соответствующий датасет

**Список источников**

1. **Антонио Джулли Суджит Пал** Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow / пер. с англ. Слинкин А.А [Книга]. - Москва : ДМК Пресс, 2018.
2. **Гребеник Виктор** История. Стили. Мастера. [Книга]. - Санкт-Петербург : Издательство «Лань»; «Издательство ПЛАНЕТА МУЗЫ­КИ», 2011.
3. **Рашка Себастьян, Мирджалили, Вахид.** Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow 2 [Книга]. - Санкт-Петербург : ООО "Диалектика", 2020.
4. **Седжвик Роберт, Уэйн, Кевин, Дондеро, Роберт.** “Программирование на языке Python: учебный курс.” [Книга]. - Cанкт-Петербург : ООО "Альфа-книга', 2017.
5. <http://midkar.com/jazz>
6. [midi-karaoke.info](https://www.midi-karaoke.info/)
7. [bitmidi.com](https://bitmidi.com/)
8. [freemidi.org](https://freemidi.org/)
9. [hotmo.org](http://hotmo.org/) (MP3)
10. [discogs.com](https://www.discogs.com/)
11. [spotify.com](https://www.spotify.com/ru-ru/)
12. [stackoverflow.com](https://ru.stackoverflow.com/)
13. <https://github.com/openai/pixel-cnn>
14. <https://github.com/HackerPoet/Composer>

**Приложение**

**Composer**

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
  
*"""  
Neural composer: Play and edit music generated by the trained model.  
"""*import argparse  
import math  
import wave  
  
import numpy as np  
import pyaudio  
import pygame  
import params  
  
import midi\_utils  
  
import keras  
from keras.models import Model, load\_model  
from keras import backend as K  
  
# User constants  
dir\_name = 'results/history/'  
sub\_dir\_name = 'e2000/'  
sample\_rate = 48000  
note\_dt = 2000 # num samples  
note\_duration = 20000 # num samples  
note\_decay = 5.0 / sample\_rate  
num\_params = params.num\_params  
num\_measures = 16  
num\_sigmas = 5.0  
note\_threshold = 32  
use\_pca = True  
is\_ae = True  
autosave = False  
autosavenum = 1  
autosavenow = False  
blend = False  
blendfactor = np.float32(1.0)  
#0 first sond 1 first to second 2 second song 3 second to first  
blendstate = 0  
  
# colors  
background\_color = (210, 210, 210)  
edge\_color = (60, 60, 60)  
slider\_colors = [(90, 20, 20), (90, 90, 20), (20, 90, 20),  
 (20, 90, 90), (20, 20, 90), (90, 20, 90)]  
  
note\_w = 96  
note\_h = 96  
note\_pad = 2  
  
notes\_rows = int(num\_measures / 8)  
notes\_cols = 8  
  
slider\_num = min(40, num\_params)  
slider\_h = 200  
slider\_pad = 5  
tick\_pad = 4  
  
control\_w = 200  
control\_h = 30  
control\_pad = 5  
control\_num = 5  
control\_colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (0, 255, 255), (255, 255, 0)]  
control\_inits = [0.75, 0.5, 0.5, 0.5, 0.5]  
  
# derived constants  
notes\_w = notes\_cols \* (note\_w + note\_pad \* 2)  
notes\_h = notes\_rows \* (note\_h + note\_pad \* 2)  
sliders\_w = notes\_w  
sliders\_h = slider\_h + slider\_pad \* 2  
controls\_w = control\_w \* control\_num  
controls\_h = control\_h  
window\_w = max(notes\_w, controls\_w)  
window\_h = notes\_h + sliders\_h + controls\_h  
slider\_w = int((window\_w - slider\_pad \* 2) / slider\_num)  
notes\_x = 0  
notes\_y = sliders\_h  
text\_x = notes\_w + 5  
text\_y = notes\_y + 5  
text\_h = 40  
text\_w = 200  
sliders\_x = slider\_pad  
sliders\_y = slider\_pad  
controls\_x = int((window\_w - controls\_w) / 2)  
controls\_y = notes\_h + sliders\_h  
  
# global variables  
keyframe\_paths = np.array(("song 1.txt", "song 2.txt", ))  
prev\_mouse\_pos = None  
mouse\_pressed = 0  
cur\_slider\_ix = 0  
cur\_control\_ix = 0  
volume = 3000  
balance = 0.5  
instrument = 0  
needs\_update = True  
current\_params = np.zeros((num\_params,), dtype=np.float32)  
keyframe\_params = np.zeros((len(keyframe\_paths),num\_params),dtype=np.float32)  
current\_notes = np.zeros((num\_measures, note\_h, note\_w), dtype=np.uint8)  
cur\_controls = np.array(control\_inits, dtype=np.float32)  
keyframe\_controls = np.zeros((len(keyframe\_paths),len(cur\_controls)),dtype=np.float32)  
blend\_slerp = False  
keyframe\_magnitudes = np.zeros((len(keyframe\_paths),),dtype=np.float32)  
songs\_loaded = False  
  
# setup audio stream  
audio = pyaudio.PyAudio()  
audio\_notes = []  
audio\_notes\_lengths = {}  
audio\_time = 0  
note\_time = 0  
note\_time\_dt = 0  
audio\_reset = False  
audio\_pause = False  
  
  
def audio\_callback(in\_data, frame\_count, time\_info, status):  
 *"""  
 Audio call-back to influence playback of music with input.* ***:param*** *in\_data:* ***:param*** *frame\_count:* ***:param*** *time\_info:* ***:param*** *status:* ***:return****:  
 """* global audio\_time  
 global audio\_notes  
 global audio\_reset  
 global note\_time  
 global note\_time\_dt  
 global autosavenow  
 global autosave  
 global audio\_pause  
 global blendstate  
 global blendfactor  
 global keyframe\_paths  
 global balance  
 global audio\_notes\_lengths  
  
 # check if needs restart  
 if audio\_reset:  
 audio\_notes = []  
 audio\_notes\_lengths = {}  
 audio\_time = 0  
 note\_time = 0  
 note\_time\_dt = 0  
 audio\_reset = False  
  
 # check if paused  
 if audio\_pause and status is not None:  
 data = np.zeros((frame\_count,), dtype=np.float32)  
 return data.tobytes(), pyaudio.paContinue  
  
 # find and add any notes in this time window  
 cur\_dt = note\_dt  
 while note\_time\_dt < audio\_time + frame\_count:  
 measure\_ix = int(note\_time / note\_h)  
 if measure\_ix >= num\_measures:  
 break  
 note\_ix = note\_time % note\_h  
 notes = np.where(  
 current\_notes[measure\_ix, note\_ix] >= note\_threshold)[0]  
 for note in notes:  
 freq = 2 \* 38.89 \* pow(2.0, note / 12.0) / sample\_rate  
 if params.encode\_length:  
 if not note in audio\_notes\_lengths or audio\_notes\_lengths[note][1] < audio\_time:  
 audio\_notes\_lengths[note] = (note\_time\_dt, note\_time\_dt + note\_dt, freq, current\_notes[measure\_ix, note\_ix, note])  
 else:  
 audio\_notes\_lengths[note] = (audio\_notes\_lengths[note][0], note\_time\_dt + note\_dt, freq, current\_notes[measure\_ix, note\_ix, note])  
 else:  
 audio\_notes.append((note\_time\_dt, note\_time\_dt + note\_duration, freq, current\_notes[measure\_ix, note\_ix, note]))  
 note\_time += 1  
 note\_time\_dt += cur\_dt  
  
 # generate the tones  
 data = np.zeros((frame\_count,), dtype=np.float32)  
 for t, e, f, v in audio\_notes\_lengths if params.encode\_volume else audio\_notes:  
 if e < audio\_time:  
 continue  
 startTime = 0 if params.encode\_volume else t;  
 x = np.arange(audio\_time - startTime, audio\_time + frame\_count - startTime)  
 x = np.maximum(x, 0)  
  
 if instrument == 0:  
 w = np.sign(1 - np.mod(x \* f, 2)) # Square  
 elif instrument == 1:  
 w = np.mod(x \* f - 1, 2) - 1 # Sawtooth  
 elif instrument == 2:  
 w = 2 \* np.abs(np.mod(x \* f - 0.5, 2) - 1) - 1 # Triangle  
 elif instrument == 3:  
 w = np.sin(x \* f \* math.pi) # Sine  
 elif instrument == 4:  
 w = -1 \* np.sign(np.mod(2\*x\*f,4)-2) \* np.sqrt( 1-( ( np.mod(2\*x\*f,2)-1) \*( ( np.mod(2\*x\*f,2)-1) ) )) # Circle  
  
 # w = np.floor(w\*8)/8  
 w[x == 0] = 0  
 n = 12 \* np.log (f \* sample\_rate / 38.89) / np.log(2);  
 w \*= volume \* np.exp(-x \* note\_decay) \* pow(balance, (n - 60) / 12.0) / np.log(2)  
 if params.encode\_volume:  
 w \*= v / 255  
 data += w  
 data = np.clip(data, -32000, 32000).astype(np.int16)  
  
 # remove notes that are too old  
 audio\_time += frame\_count  
 audio\_notes = [(t, e, f, v)  
 for t, e, f, v in audio\_notes if audio\_time < t + note\_duration]  
 blendfactor = (np.cos( ((note\_time / note\_h)/num\_measures) \* math.pi )+1)/2  
 #print(blendfactor)  
 # reset if loop occurs  
 if note\_time / note\_h >= num\_measures:  
 audio\_time = 0  
 note\_time = 0  
 note\_time\_dt = 0  
 audio\_notes = []  
 audio\_notes\_lengths = {}  
 blendstate = (blendstate+1)%(2\*len(keyframe\_paths))  
 #if blendstate == 0:  
 #audio\_pause = True  
 blendfactor = 1  
 if autosave and not autosavenow:  
 autosavenow = True  
  
 # return the sound clip  
 return data.tobytes(), pyaudio.paContinue  
  
  
def update\_mouse\_click(mouse\_pos):  
 *"""  
 Update control stated based on where the mouse clicked.* ***:param*** *mouse\_pos:* ***:return****:  
 """* global cur\_slider\_ix  
 global cur\_control\_ix  
 global mouse\_pressed  
 x = (mouse\_pos[0] - sliders\_x)  
 y = (mouse\_pos[1] - sliders\_y)  
  
 if 0 <= x < sliders\_w and 0 <= y < sliders\_h:  
 cur\_slider\_ix = int(x / slider\_w)  
 mouse\_pressed = 1  
  
 x = (mouse\_pos[0] - controls\_x)  
 y = (mouse\_pos[1] - controls\_y)  
 if 0 <= x < controls\_w and 0 <= y < controls\_h:  
 cur\_control\_ix = int(x / control\_w)  
 mouse\_pressed = 2  
  
  
def apply\_controls():  
 *"""  
 Change parameters based on controls.* ***:return****:  
 """* global note\_threshold  
 global note\_dt  
 global volume  
 global note\_duration  
 global note\_decay  
 global sample\_rate  
 global balance  
  
 note\_threshold = (1.0 - cur\_controls[0]) \* 200 + 10  
 note\_dt = (1.0 - cur\_controls[1]) \* 1800 + 200  
 volume = cur\_controls[2] \* 6000  
 balance = pow(2, cur\_controls[3] \* 4 - 2);  
  
 note\_duration = 10000 / ((1-cur\_controls[4]) + 0.001)  
 note\_decay = 10 \* (1 - cur\_controls[4]) / sample\_rate  
  
  
def update\_mouse\_move(mouse\_pos):  
 *"""  
 Update sliders/controls based on mouse input.* ***:param*** *mouse\_pos:* ***:return****:  
 """* global needs\_update  
 t = 1  
 if int(cur\_control\_ix) == 0:  
 t = 210.0 / 200  
 if mouse\_pressed == 1:  
 # change sliders  
 y = (mouse\_pos[1] - sliders\_y)  
 if 0 <= y <= slider\_h:  
 val = (float(y) / slider\_h - 0.5) \* (num\_sigmas \* 2)  
 current\_params[int(cur\_slider\_ix)] = val  
 needs\_update = True  
 elif mouse\_pressed == 2:  
 # change controls  
 x = (mouse\_pos[0] - (controls\_x + cur\_control\_ix \* control\_w))  
 if control\_pad <= x <= control\_w - control\_pad:  
 val = float(x - control\_pad) / (control\_w - control\_pad \* 2)  
 cur\_controls[int(cur\_control\_ix)] = val \* t  
 apply\_controls()  
  
  
def draw\_controls(screen):  
 *"""  
 Draw volume and threshold controls to screen.* ***:param*** *screen:* ***:return****:  
 """* #allows for higher threshold  
 t = 200.0 / 210  
 for i in range(control\_num):  
 x = controls\_x + i \* control\_w + control\_pad  
 y = controls\_y + control\_pad  
 w = control\_w - control\_pad \* 2  
 h = control\_h - control\_pad \* 2  
 col = control\_colors[i]  
  
 pygame.draw.rect(screen, col, (x, y, int(w \* t \* cur\_controls[i]), h))  
 pygame.draw.rect(screen, (0, 0, 0), (x, y, w, h), 1)  
  
 t = 1  
  
  
def draw\_sliders(screen):  
 *"""  
 Draw sliders to screen.* ***:param*** *screen:* ***:return****:  
 """* for i in range(slider\_num):  
 slider\_color = slider\_colors[i % len(slider\_colors)]  
 x = sliders\_x + i \* slider\_w  
 y = sliders\_y  
  
 cx = x + slider\_w / 2  
 cy\_1 = y  
 cy\_2 = y + slider\_h  
 pygame.draw.line(screen, slider\_color, (cx, cy\_1), (cx, cy\_2))  
  
 cx\_1 = x + tick\_pad  
 cx\_2 = x + slider\_w - tick\_pad  
 for j in range(int(num\_sigmas \* 2 + 1)):  
 ly = y + slider\_h / 2.0 + \  
 (j - num\_sigmas) \* slider\_h / (num\_sigmas \* 2.0)  
 ly = int(ly)  
 col = (0, 0, 0) if j - num\_sigmas == 0 else slider\_color  
 pygame.draw.line(screen, col, (cx\_1, ly), (cx\_2, ly))  
  
 py = y + int((current\_params[i] / (num\_sigmas \* 2) + 0.5) \* slider\_h)  
 pygame.draw.circle(screen, slider\_color, (int(  
 cx), int(py)), int((slider\_w - tick\_pad) / 2))  
  
  
def get\_pianoroll\_from\_notes(notes):  
 *"""  
 Draw piano roll of notes.* ***:param*** *notes:* ***:return****:  
 """* output = np.full((3, int(notes\_h), int(notes\_w)), 64, dtype=np.uint8)  
  
 for i in range(notes\_rows):  
 for j in range(notes\_cols):  
 x = note\_pad + j \* (note\_w + note\_pad \* 2)  
 y = note\_pad + i \* (note\_h + note\_pad \* 2)  
 ix = i \* notes\_cols + j  
  
 measure = np.rot90(notes[ix])  
  
 played\_only = np.where(measure >= note\_threshold, 255, 0)  
 output[0, y:y + note\_h, x:x +  
 note\_w] = np.minimum(measure \* (255.0 / note\_threshold), 255.0)  
 output[1, y:y + note\_h, x:x + note\_w] = played\_only  
 output[2, y:y + note\_h, x:x + note\_w] = played\_only  
  
 return np.transpose(output, (2, 1, 0))  
  
  
def draw\_notes(screen, notes\_surface):  
 *"""  
 Draw pianoroll notes to screen.* ***:param*** *screen:* ***:param*** *notes\_surface:* ***:return****:  
 """* pygame.surfarray.blit\_array(  
 notes\_surface, get\_pianoroll\_from\_notes(current\_notes))  
  
 measure\_ix = int(note\_time / note\_h)  
 note\_ix = note\_time % note\_h  
 x = notes\_x + note\_pad + (measure\_ix % notes\_cols) \* \  
 (note\_w + note\_pad \* 2) + note\_ix  
 y = notes\_y + note\_pad + \  
 int(measure\_ix / notes\_cols) \* (note\_h + note\_pad \* 2)  
  
 pygame.draw.rect(screen, (255, 255, 0), (x, y, 4, note\_h), 0)  
  
  
def play():  
 global mouse\_pressed  
 global current\_notes  
 global audio\_pause  
 global needs\_update  
 global current\_params  
 global prev\_mouse\_pos  
 global audio\_reset  
 global instrument  
 global songs\_loaded  
 global autosavenow  
 global autosavenum  
 global autosave  
 global blend  
 global blendstate  
 global blendfactor  
 global keyframe\_params  
 global keyframe\_controls  
 global keyframe\_paths  
 global cur\_controls  
 global keyframe\_magnitudes  
 global blend\_slerp  
  
 print("Keras version: " + keras.\_\_version\_\_)  
  
 K.set\_image\_data\_format('channels\_first')  
  
 print("Loading encoder...")  
 model = load\_model(dir\_name + 'model.h5')  
 encoder = Model(inputs=model.input,  
 outputs=model.get\_layer('encoder').output)  
 decoder = K.function([model.get\_layer('decoder').input, K.learning\_phase()],  
 [model.layers[-1].output])  
  
 print("Loading gaussian/pca statistics...")  
 latent\_means = np.load(dir\_name + sub\_dir\_name + '/latent\_means.npy')  
 latent\_stds = np.load(dir\_name + sub\_dir\_name + '/latent\_stds.npy')  
 latent\_pca\_values = np.load(  
 dir\_name + sub\_dir\_name + '/latent\_pca\_values.npy')  
 latent\_pca\_vectors = np.load(  
 dir\_name + sub\_dir\_name + '/latent\_pca\_vectors.npy')  
  
 # open a window  
 pygame.init()  
 pygame.font.init()  
 screen = pygame.display.set\_mode((int(window\_w), int(window\_h)))  
 notes\_surface = screen.subsurface((notes\_x, notes\_y, notes\_w, notes\_h))  
 pygame.display.set\_caption('Neural Composer')  
  
 # start the audio stream  
 audio\_stream = audio.open(  
 format=audio.get\_format\_from\_width(2),  
 channels=1,  
 rate=sample\_rate,  
 output=True,  
 stream\_callback=audio\_callback)  
 audio\_stream.start\_stream()  
  
 # main loop  
 running = True  
 random\_song\_ix = 0  
 cur\_len = 0  
 blendcycle = 0  
 apply\_controls()  
 while running:  
 # process events  
 if autosavenow:  
 # generate random song  
 current\_params = np.clip(np.random.normal(  
 0.0, 1.0, (num\_params,)), -num\_sigmas, num\_sigmas)  
 needs\_update = True  
 audio\_reset = True  
 # save slider values  
 with open("results/history/autosave" + str(autosavenum)+".txt", "w") as text\_file:  
 text\_file.write(sub\_dir\_name + "\n")  
 text\_file.write(str(instrument) + "\n")  
 for iter in cur\_controls:  
 text\_file.write(str(iter) + "\n")  
 for iter in current\_params:  
 text\_file.write(str(iter) + "\n")  
 # save song as wave  
 audio\_pause = True  
 audio\_reset = True  
 save\_audio = b''  
 while True:  
 save\_audio += audio\_callback(None, 1024, None, None)[0]  
 if audio\_time == 0:  
 break  
 wave\_output = wave.open('results/history/autosave' + str(autosavenum)+'.wav', 'w')  
 wave\_output.setparams(  
 (1, 2, sample\_rate, 0, 'NONE', 'not compressed'))  
 wave\_output.writeframes(save\_audio)  
 wave\_output.close()  
 audio\_pause = False  
 autosavenum += 1  
 autosavenow = False  
 needs\_update = True  
 audio\_reset = True  
 blendcycle += 1  
 if blend and blendcycle > 10:  
 blendcycle = 0  
 if blendstate%2 == 0:  
 needs\_update = True  
 current\_params = np.copy(keyframe\_params[int(blendstate/2)])  
 cur\_controls = np.copy(keyframe\_controls[int(blendstate/2)])  
 apply\_controls()  
 elif blendstate%2 == 1:  
 for x in range(0,len(current\_params)):  
 current\_params[x] = (blendfactor \* keyframe\_params[int(blendstate/2),x]) + ((1-blendfactor)\*keyframe\_params[((int(blendstate/2))+1)%len(keyframe\_paths),x])  
 if blend\_slerp:  
 magnitude = (blendfactor \* keyframe\_magnitudes[int(blendstate/2)]) + ((1-blendfactor)\*keyframe\_magnitudes[((int(blendstate/2))+1)%len(keyframe\_paths)])  
 current\_params = current\_params \* ((sum(current\_params\*current\_params)\*\*-0.5) \* magnitude)  
 for x in range(0,len(cur\_controls)):  
 cur\_controls[x] = (blendfactor \* keyframe\_controls[int(blendstate/2),x]) + ((1-blendfactor)\*keyframe\_controls[((int(blendstate/2))+1)%len(keyframe\_paths),x])  
 apply\_controls()  
 needs\_update = True  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT: # QUIT BUTTON HIT  
 running = False  
 break  
  
 elif event.type == pygame.MOUSEBUTTONDOWN: # MOUSE BUTTON DOWN  
 if pygame.mouse.get\_pressed()[0]:  
 prev\_mouse\_pos = pygame.mouse.get\_pos()  
 update\_mouse\_click(prev\_mouse\_pos)  
 update\_mouse\_move(prev\_mouse\_pos)  
 elif pygame.mouse.get\_pressed()[2]:  
 current\_params = np.zeros((num\_params,), dtype=np.float32)  
 needs\_update = True  
  
 elif event.type == pygame.MOUSEBUTTONUP: # MOUSE BUTTON UP  
 mouse\_pressed = 0  
 prev\_mouse\_pos = None  
  
 elif event.type == pygame.MOUSEMOTION and mouse\_pressed > 0: # MOUSE MOTION WHILE PRESSED  
 update\_mouse\_move(pygame.mouse.get\_pos())  
  
 elif event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_r: # KEYDOWN R  
 # generate random song  
 current\_params = np.clip(np.random.normal(  
 0.0, 1.0, (num\_params,)), -num\_sigmas, num\_sigmas)  
 needs\_update = True  
 audio\_reset = True  
 if event.key == pygame.K\_t: # KEYDOWN T  
 for x in range(int(num\_params/3)+1, num\_params):  
 current\_params[x] = np.clip(np.random.normal(0.0,1.0), -num\_sigmas, num\_sigmas)  
 needs\_update = True  
 if event.key == pygame.K\_x: # KEYDOWN X  
 # generate random song  
 current\_params += np.clip(np.random.normal(  
 0.0, 0.3, (num\_params,)), -num\_sigmas, num\_sigmas)  
 needs\_update = True  
 if event.key == pygame.K\_a: # KEYDOWN A  
 autosave = not autosave  
 if event.key == pygame.K\_b: # KEYDOWN B  
 blend = not blend  
 blendstate = 0  
 blendfactor = 1.0  
 if blend:  
 audio\_pause = True  
 audio\_reset = True  
 needs\_update = True  
 blendnum = int(input("The number of songs to be blended "))  
 keyframe\_paths = []  
 keyframe\_controls = np.zeros((blendnum,len(cur\_controls)),dtype=np.float32)  
 keyframe\_params = np.zeros((blendnum,num\_params),dtype=np.float32)  
 for y in range(blendnum):  
 fileName = input("The file name of the next song to be blended ")  
 if "." not in fileName:  
 fileName = fileName + ".txt"  
 keyframe\_paths.append((fileName))  
 fo = open("results/history/" + fileName, "r")  
 if not sub\_dir\_name == fo.readline()[:-1]:  
 running = false  
 print("incompatable with current model")  
 break  
 instrument = int(fo.readline())  
 for x in range(len(cur\_controls)):  
 keyframe\_controls[y,x] = float(fo.readline())  
 for x in range(len(current\_params)):  
 keyframe\_params[y,x] = float(fo.readline())  
 #keyframe\_magnitudes[y] = sum(keyframe\_params[y]\*keyframe\_params[y])\*\*0.5  
 if event.key == pygame.K\_e: # KEYDOWN E  
 # generate random song with larger variance  
 current\_params = np.clip(np.random.normal(0.0, 2.0, (num\_params,)), -num\_sigmas, num\_sigmas)  
 needs\_update = True  
 audio\_reset = True  
 if event.key == pygame.K\_PERIOD:  
 current\_params /= 1.1  
 needs\_update = True  
 if event.key == pygame.K\_COMMA:  
 current\_params \*= 1.1  
 needs\_update = True  
 if event.key == pygame.K\_SLASH:  
 current\_params \*= -1  
 needs\_update = True  
 if event.key == pygame.K\_UP:  
 cur\_controls[0] = (210.0 - note\_threshold + 1) / 200  
 apply\_controls()  
 if event.key == pygame.K\_DOWN:  
 cur\_controls[0] = (210.0 - note\_threshold - 1) / 200  
 apply\_controls()  
 if event.key == pygame.K\_s: # KEYDOWN S  
 # save slider values  
 audio\_pause = True  
 fileName = input("File Name to save into ")  
 if "." not in fileName:  
 fileName = fileName + ".txt"  
 with open("results/history/" + fileName, "w") as text\_file:  
 if blend:  
 text\_file.write(sub\_dir\_name + "\n")  
 text\_file.write("blended song" + "\n")  
 text\_file.write(str(len(keyframe\_paths)) + "\n")  
 for x in range(len(keyframe\_paths)):  
 text\_file.write("" + keyframe\_paths[x] + "\n")  
 else:  
 text\_file.write(sub\_dir\_name + "\n")  
 text\_file.write(str(instrument) + "\n")  
 for iter in cur\_controls:  
 text\_file.write(str(iter) + "\n")  
 for iter in current\_params:  
 text\_file.write(str(iter) + "\n")  
 if event.key == pygame.K\_l: # KEYDOWN L  
 audio\_pause = True  
 needs\_update = True  
 audio\_reset = True  
 fileName = input("File Name to read ")  
 if "." not in fileName:  
 fileName = fileName + ".txt"  
 fo = open("results/history/" + fileName, "r")  
 print (fo.name)  
 if not sub\_dir\_name == fo.readline()[:-1]:  
 running = false  
 print("incompatable with current model")  
 break  
 tempDir = fo.readline()  
 if tempDir.startswith("blended song"):  
 blend = True  
 blendnum = int(fo.readline())  
 keyframe\_paths = []  
 keyframe\_controls = np.zeros((blendnum,len(cur\_controls)),dtype=np.float32)  
 keyframe\_params = np.zeros((blendnum,num\_params),dtype=np.float32)  
 for y in range(blendnum):  
 fileName2 = fo.readline()[:-1]  
 keyframe\_paths.append(fileName)  
 fo2 = open("results/history/" + fileName2, "r")  
 if not sub\_dir\_name == fo2.readline()[:-1]:  
 running = false  
 print("incompatable with current model")  
 break  
 instrument = int(fo2.readline())  
 for x in range(len(cur\_controls)):  
 keyframe\_controls[y,x] = float(fo2.readline())  
 for x in range(len(current\_params)):  
 keyframe\_params[y,x] = float(fo2.readline())  
 else:  
 instrument = int(tempDir)  
 for x in range(len(cur\_controls)):  
 cur\_controls[x] = float(fo.readline())  
 for x in range(len(current\_params)):  
 current\_params[x] = float(fo.readline())  
 apply\_controls()  
 if event.key == pygame.K\_o: # KEYDOWN O  
  
 if not songs\_loaded:  
 print("Loading songs...")  
 try:  
 y\_samples = np.load('data/interim/samples.npy')  
 y\_lengths = np.load('data/interim/lengths.npy')  
 songs\_loaded = True  
 except Exception as e:  
 print("This functionality is to check if the model training went well by reproducing an original song. "  
 "The composer could not load samples and lengths from model training. "  
 "If you have the midi files, the model was trained with, process them by using"  
 " the preprocess\_songs.py to find the requested files in data/interim "  
 "(Load exception: {0}".format(e))  
  
 if songs\_loaded:  
 # check how well the autoencoder can reconstruct a random song  
 print("Random Song Index: " + str(random\_song\_ix))  
 if is\_ae:  
 example\_song = y\_samples[cur\_len:cur\_len + num\_measures]  
 current\_notes = example\_song \* 255  
 latent\_x = encoder.predict(np.expand\_dims(  
 example\_song, 0), batch\_size=1)[0]  
 cur\_len += y\_lengths[random\_song\_ix]  
 random\_song\_ix += 1  
 else:  
 random\_song\_ix = np.array(  
 [random\_song\_ix], dtype=np.int64)  
 latent\_x = encoder.predict(  
 random\_song\_ix, batch\_size=1)[0]  
 random\_song\_ix = (  
 random\_song\_ix + 1) % model.layers[0].input\_dim  
  
 if use\_pca:  
 current\_params = np.dot(  
 latent\_x - latent\_means, latent\_pca\_vectors.T) / latent\_pca\_values  
 else:  
 current\_params = (  
 latent\_x - latent\_means) / latent\_stds  
  
 needs\_update = True  
 audio\_reset = True  
  
 if event.key == pygame.K\_m: # KEYDOWN M  
 # save song as midi  
 audio\_pause = True  
 audio\_reset = True  
 fileName = input("File Name to save into ")  
 if "." not in fileName:  
 fileName = fileName + ".mid"  
 midi\_utils.samples\_to\_midi(  
 current\_notes, 'results/history/' + fileName, note\_threshold)  
 audio\_pause = False  
  
 if event.key == pygame.K\_w: # KEYDOWN W  
 # save song as wave  
 audio\_pause = True  
 audio\_reset = True  
 fileName = input("File Name to save into ")  
 if "." not in fileName:  
 fileName = fileName + ".wav"  
 save\_audio = b''  
 while True:  
 save\_audio += audio\_callback(None, 1024, None, None)[0]  
 if audio\_time == 0:  
 break  
 wave\_output = wave.open('results/history/' + fileName + '.wav', 'w')  
 wave\_output.setparams(  
 (1, 2, sample\_rate, 0, 'NONE', 'not compressed'))  
 wave\_output.writeframes(save\_audio)  
 wave\_output.close()  
 audio\_pause = False  
  
 if event.key == pygame.K\_ESCAPE: # KEYDOWN ESCAPE  
 # exit application  
 running = False  
 break  
  
 if event.key == pygame.K\_SPACE: # KEYDOWN SPACE  
 # toggle pause/play audio  
 audio\_pause = not audio\_pause  
  
 if event.key == pygame.K\_TAB: # KEYDOWN TAB  
 # reset audio playing  
 audio\_reset = True  
 if autosave and not autosavenow:  
 autosavenow = True  
  
 if event.key == pygame.K\_1: # KEYDOWN 1  
 # play instrument 0  
 instrument = 0  
  
 if event.key == pygame.K\_2: # KEYDOWN 2  
 # play instrument 1  
 instrument = 1  
  
 if event.key == pygame.K\_3: # KEYDOWN 3  
 # play instrument 2  
 instrument = 2  
  
 if event.key == pygame.K\_4: # KEYDOWN 4  
 # play instrument 3  
 instrument = 3  
  
 if event.key == pygame.K\_5: # KEYDOWN 5  
 # play instrument 4  
 instrument = 4  
  
 if event.key == pygame.K\_c: # KEYDOWN C  
 #  
 y = np.expand\_dims(  
 np.where(current\_notes > note\_threshold, 1, 0), 0)  
 latent\_x = encoder.predict(y)[0]  
 if use\_pca:  
 current\_params = np.dot(  
 latent\_x - latent\_means, latent\_pca\_vectors.T) / latent\_pca\_values  
 else:  
 current\_params = (  
 latent\_x - latent\_means) / latent\_stds  
 needs\_update = True  
  
 # check if params were changed so that a new song should be generated  
 if needs\_update:  
 if use\_pca:  
 latent\_x = latent\_means + \  
 np.dot(current\_params \* latent\_pca\_values,  
 latent\_pca\_vectors)  
 else:  
 latent\_x = latent\_means + latent\_stds \* current\_params  
 latent\_x = np.expand\_dims(latent\_x, axis=0)  
 y = decoder([latent\_x, 0])[0][0]  
 current\_notes = (y \* (255)).astype(np.uint8)  
 needs\_update = False  
  
 # draw GUI to the screen  
 screen.fill(background\_color)  
 draw\_notes(screen, notes\_surface)  
 draw\_sliders(screen)  
 draw\_controls(screen)  
  
 # flip the screen buffer  
 pygame.display.flip()  
 pygame.time.wait(10)  
  
 # if app is exited, close the audio stream  
 audio\_stream.stop\_stream()  
 audio\_stream.close()  
 audio.terminate()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # configure parser and parse arguments  
 parser = argparse.ArgumentParser(  
 description='Neural Composer: Play and edit music of a trained model.')  
 parser.add\_argument('--model\_path', type=str,  
 help='The folder the model is stored in (e.g. a folder named e and a number located in results/history/).', required=True)  
  
 args = parser.parse\_args()  
 if args.model\_path.endswith(".txt"):  
 fo = open("results/history/" + args.model\_path, "r")  
 print (fo.name)  
 sub\_dir\_name = fo.readline()[:-1]  
 tempDir = fo.readline()  
 if tempDir.startswith("blended song"):  
 blend = True  
 blendnum = int(fo.readline())  
 keyframe\_paths = []  
 keyframe\_controls = np.zeros((blendnum,len(cur\_controls)),dtype=np.float32)  
 keyframe\_params = np.zeros((blendnum,num\_params),dtype=np.float32)  
 for y in range(blendnum):  
 fileName2 = fo.readline()[:-1]  
 keyframe\_paths.append(fileName2)  
 fo2 = open("results/history/" + fileName2, "r")  
 if not sub\_dir\_name == fo2.readline()[:-1]:  
 running = false  
 print("incompatable with current model")  
 break  
 instrument = int(fo2.readline())  
 for x in range(len(cur\_controls)):  
 keyframe\_controls[y,x] = float(fo2.readline())  
 for x in range(len(current\_params)):  
 keyframe\_params[y,x] = float(fo2.readline())  
 else:  
 print(sub\_dir\_name)  
 instrument = int(tempDir)  
 for x in range(len(cur\_controls)):  
 cur\_controls[x] = float(fo.readline())  
 for x in range(len(current\_params)):  
 current\_params[x] = float(fo.readline())  
   
 else:  
 sub\_dir\_name = args.model\_path  
 play()

**Midi\_utils**

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
  
*"""  
Utils to read and write midi.  
"""*from mido import MidiFile, MidiTrack, Message  
import numpy as np  
import params  
  
  
def midi\_to\_samples(file\_name, num\_notes=96, samples\_per\_measure=96):  
 *"""  
 Turn a midi file into a sample.* ***:param*** *file\_name:* ***:param*** *num\_notes:* ***:param*** *samples\_per\_measure:* ***:return****:  
 """* has\_time\_sig = False  
 mid = MidiFile(file\_name)  
  
 ticks\_per\_beat = mid.ticks\_per\_beat # get ticks per beat  
 ticks\_per\_measure = 4 \* ticks\_per\_beat # get ticks per measure  
  
 # detect the time signature of the midi  
 for track in mid.tracks:  
 for msg in track:  
 if msg.type == 'time\_signature':  
 new\_tpm = ticks\_per\_measure \* msg.numerator / msg.denominator # adapt ticks per measure of this specific song  
  
 # skip if we find multiple time signatures in the song  
 if has\_time\_sig and new\_tpm != ticks\_per\_measure:  
 raise NotImplementedError('Multiple time signatures not supported')  
  
 ticks\_per\_measure = new\_tpm  
 has\_time\_sig = True  
  
 # turn tracks into pianoroll representation  
 maxVol = 1  
 all\_notes = {}  
 for track in mid.tracks:  
  
 abs\_time = 0  
 for msg in track:  
 abs\_time += msg.time # step time forward  
  
 # we skip programs 0x70-0x7F which are percussion and sound effects  
 if msg.type == 'program\_change' and msg.program >= 0x70:  
 break  
  
 # if a note starts  
 if msg.type == 'note\_on':  
  
 # we skip notes without a velocity (basically how strong a note is played to make it sound human)  
 if msg.velocity == 0:  
 continue  
   
 if msg.velocity > maxVol:  
 maxVol = msg.velocity  
  
 # transform the notes into the 96 heights  
 note = msg.note - (128 - num\_notes) / 2  
 if note < 0 or note >= num\_notes: # ignore a note that is outside of that range  
 print('Ignoring', file\_name, 'note is outside 0-%d range' % (num\_notes - 1))  
 return []  
  
 # count the number of played notes per pitch  
 if note not in all\_notes:  
 all\_notes[note] = []  
 else:  
 single\_note = all\_notes[note][-1]  
 if len(single\_note) == 2:  
 single\_note.append(single\_note[0] + 1)  
  
 # store the time a note has been played  
 all\_notes[note].append([abs\_time \* samples\_per\_measure / ticks\_per\_measure])  
 all\_notes[note][-1].append(msg.velocity)  
  
 # if a note ends  
 elif msg.type == 'note\_off':  
  
 # if the note has already ended before (note\_on, note\_off, note\_off), we skip the event  
 if len(all\_notes[note][-1]) != 2:  
 continue  
 # store the time a note stops playing  
 all\_notes[note][-1].append(abs\_time \* samples\_per\_measure / ticks\_per\_measure)  
  
 # any note did not end playing, we end it one time tick later  
 for note in all\_notes:  
 for start\_end in all\_notes[note]:  
 if len(start\_end) == 2:  
 start\_end.append(start\_end[0] + 1)  
  
 #print(maxVol)  
  
 # put the notes into their respective sample/measure panel (96 x 96)  
 samples = []  
 for note in all\_notes:  
 for start, vel, end in all\_notes[note]:  
 sample\_ix = int(start / samples\_per\_measure) # find the sample/measure this belongs into  
 assert (sample\_ix < 1024 \* 1024)  
  
 # fill in silence until the appropriate sample/measure is reached  
 while len(samples) <= sample\_ix:  
 samples.append(np.zeros((samples\_per\_measure, num\_notes), dtype=np.uint8))  
  
 # get sample and find its start to encode the start of the note  
 sample = samples[sample\_ix]  
 start\_ix = int(start - sample\_ix \* samples\_per\_measure)  
 sample[start\_ix, int(note)] = vel / maxVol if params.encode\_volume else 1  
 #print(vel)  
 #print(maxVol)  
  
 # play note until it ends if we encode length  
 if params.encode\_length:  
 end\_ix = min(end - sample\_ix \* samples\_per\_measure, samples\_per\_measure)  
 while start\_ix < end\_ix:  
 sample[start\_ix, int(note)] = vel / maxVol if params.encode\_volume else 1  
 start\_ix += 1  
   
   
  
 return samples  
  
  
def samples\_to\_midi(samples, file\_name, threshold=0.5, num\_notes=96, samples\_per\_measure=96):  
 *"""  
 Turn the samples/measures back into midi.* ***:param*** *samples:* ***:param*** *file\_name:* ***:param*** *threshold:* ***:param*** *num\_notes:* ***:param*** *samples\_per\_measure:* ***:return****:  
 """* # *TODO: Encode the certainties of the notes into the volume of the midi for the notes that are above threshold* mid = MidiFile()  
 track = MidiTrack()  
 mid.tracks.append(track)  
  
 ticks\_per\_beat = mid.ticks\_per\_beat  
 ticks\_per\_measure = 4 \* ticks\_per\_beat  
 ticks\_per\_sample = ticks\_per\_measure / samples\_per\_measure  
  
 # add instrument for track  
 # https://en.wikipedia.org/wiki/General\_MIDI#Program\_change\_events  
 piano = 1  
 honky\_tonk\_piano = 4  
 xylophone = 14  
 program\_message = Message('program\_change', program=piano, time=0, channel=0)  
 track.append(program\_message)  
  
 abs\_time = 0  
 last\_time = 0  
 for sample in samples:  
 for y in range(sample.shape[0]):  
 abs\_time += ticks\_per\_sample  
 for x in range(sample.shape[1]):  
 note = x + (128 - num\_notes) / 2  
  
 if sample[y, x] >= threshold and (y == 0 or sample[y - 1, x] < threshold):  
 delta\_time = abs\_time - last\_time  
 track.append(Message('note\_on', note=int(note), velocity=127, time=int(delta\_time)))  
 last\_time = abs\_time  
  
 if sample[y, x] >= threshold and (y == sample.shape[0] - 1 or sample[y + 1, x] < threshold):  
 delta\_time = abs\_time - last\_time  
 track.append(Message('note\_off', note=int(note), velocity=127, time=int(delta\_time)))  
 last\_time = abs\_time  
 mid.save(file\_name)

**Models**

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
  
*"""  
The models used for music generation.  
"""*from keras import backend as K  
from keras.layers import Input, Dense, Activation, Dropout, Flatten, Reshape, TimeDistributed, Lambda  
from keras.layers.embeddings import Embedding  
from keras.layers.normalization import BatchNormalization  
from keras.models import Model  
  
import params  
  
  
def vae\_sampling(args):  
 z\_mean, z\_log\_sigma\_sq, vae\_b1 = args  
 epsilon = K.random\_normal(shape=K.shape(z\_mean), mean=0.0, stddev=vae\_b1)  
 return z\_mean + K.exp(z\_log\_sigma\_sq \* 0.5) \* epsilon  
  
  
def create\_autoencoder\_model(input\_shape, latent\_space\_size, dropout\_rate, max\_windows, batchnorm\_momentum, use\_vae=False, vae\_b1=0.02, use\_embedding=False, embedding\_input\_shape=None, embedding\_shape=None):  
 *"""  
 Create larger autoencoder with the options of making it variational and embedding.* ***:param*** *input\_shape:* ***:param*** *latent\_space\_size:* ***:param*** *dropout\_rate:* ***:param*** *max\_windows:* ***:param*** *batchnorm\_momentum:* ***:param*** *use\_vae:* ***:param*** *vae\_b1:* ***:param*** *use\_embedding:* ***:param*** *embedding\_input\_shape:* ***:param*** *embedding\_shape:* ***:return****:  
 """* if use\_embedding:  
 x\_in = Input(shape=embedding\_input\_shape)  
 print((None,) + embedding\_input\_shape)  
  
 x = Embedding(embedding\_shape, latent\_space\_size, input\_length=1)(x\_in)  
 x = Flatten(name='encoder')(x)  
 else:  
 x\_in = Input(shape=input\_shape)  
 print((None,) + input\_shape)  
  
 x = Reshape((input\_shape[0], -1))(x\_in)  
 print(K.int\_shape(x))  
  
 if params.noise\_rate > 0:  
 x = Lambda(lambda x: 1 - x)(x)  
 x = Dropout(params.noise\_rate)(x)  
 x = Lambda(lambda x: 1 - x)(x)  
  
 print(K.int\_shape(x))  
  
 x = TimeDistributed(Dense(2000, activation='relu'))(x)  
 print(K.int\_shape(x))  
  
 x = TimeDistributed(Dense(200, activation='relu'))(x)  
 print(K.int\_shape(x))  
  
 x = Flatten()(x)  
 print(K.int\_shape(x))  
  
 x = Dense(1600, activation='relu')(x)  
 print(K.int\_shape(x))  
  
 if use\_vae:  
 z\_mean = Dense(latent\_space\_size)(x)  
 z\_log\_sigma\_sq = Dense(latent\_space\_size)(x)  
 x = Lambda(vae\_sampling, output\_shape=(latent\_space\_size,), name='encoder')([z\_mean, z\_log\_sigma\_sq, vae\_b1])  
 else:  
 x = Dense(latent\_space\_size)(x)  
 x = BatchNormalization(momentum=batchnorm\_momentum, name='encoder')(x)  
 print(K.int\_shape(x))  
  
 # LATENT SPACE  
  
 x = Dense(1600, name='decoder')(x)  
 x = BatchNormalization(momentum=batchnorm\_momentum)(x)  
 x = Activation('relu')(x)  
 if dropout\_rate > 0:  
 x = Dropout(dropout\_rate)(x)  
 print(K.int\_shape(x))  
  
 x = Dense(max\_windows \* 200)(x)  
 print(K.int\_shape(x))  
 x = Reshape((max\_windows, 200))(x)  
 x = TimeDistributed(BatchNormalization(momentum=batchnorm\_momentum))(x)  
 x = Activation('relu')(x)  
 if dropout\_rate > 0:  
 x = Dropout(dropout\_rate)(x)  
 print(K.int\_shape(x))  
  
 x = TimeDistributed(Dense(2000))(x)  
 x = TimeDistributed(BatchNormalization(momentum=batchnorm\_momentum))(x)  
 x = Activation('relu')(x)  
 if dropout\_rate > 0:  
 x = Dropout(dropout\_rate)(x)  
 print(K.int\_shape(x))  
  
 #if params.encode\_volume:  
 #x = TimeDistributed(Dense(input\_shape[1] \* input\_shape[2]))(x)  
 #else:  
 x = TimeDistributed(Dense(input\_shape[1] \* input\_shape[2], activation='sigmoid'))(x)  
 print(K.int\_shape(x))  
 x = Reshape((input\_shape[0], input\_shape[1], input\_shape[2]))(x)  
 print(K.int\_shape(x))  
  
 model = Model(x\_in, x)  
  
 return model

**Music\_utils**

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
  
*"""  
Utils to edit music.  
"""*import numpy as np  
  
  
def find\_sample\_range(samples):  
 *"""  
 Find sample range.* ***:param*** *samples:* ***:return****:  
 """* # merge all samples  
 merged\_sample = np.zeros\_like(samples[0])  
 for sample in samples:  
 merged\_sample = np.maximum(merged\_sample, sample)  
  
 # get all pitches being played  
 merged\_sample = np.amax(merged\_sample, axis=0)  
  
 # get min and max note  
 min\_note = np.argmax(merged\_sample)  
 max\_note = merged\_sample.shape[0] - np.argmax(merged\_sample[::-1])  
 return min\_note, max\_note  
  
  
def generate\_centered\_transpose(samples):  
 *"""  
 Center samples towards the middle of the pitch range.* ***:param*** *samples:* ***:return****:  
 """* num\_notes = samples[0].shape[1]  
 min\_note, max\_note = find\_sample\_range(samples)  
  
 # find deviation from pitch center  
 center\_deviation = num\_notes / 2 - (max\_note + min\_note) / 2  
 out\_samples = samples  
 out\_lengths = [len(samples), len(samples)]  
  
 # center every sample by moving it by center\_deviation  
 for i in range(len(samples)):  
 out\_sample = np.zeros\_like(samples[i])  
 out\_sample[:, min\_note + int(center\_deviation):max\_note + int(center\_deviation)] = samples[i][:, min\_note:max\_note]  
 out\_samples.append(out\_sample)  
 return out\_samples, out\_lengths

**Plot\_utils**

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
  
*"""  
Plot utility functions.  
"""*import os  
import cv2  
import numpy as np  
  
  
def plot\_sample(file\_name, sample, threshold=None):  
 if threshold is not None:  
 inverted = np.where(sample > threshold, 0, 1)  
 else:  
 inverted = 1.0 - sample  
 cv2.imwrite(file\_name, inverted \* 255)  
  
  
def plot\_samples(folder, samples, threshold=None):  
 if not os.path.exists(folder):  
 os.makedirs(folder)  
  
 for i in range(samples.shape[0]):  
 plot\_sample(folder + '/s' + str(i) + '.png', samples[i], threshold)

**Preprocess\_songs**

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
  
*"""  
Load songs from midi, preprocess and save them in numpy format.  
"""*import midi\_utils  
import os  
import music\_utils  
import numpy as np  
import argparse  
import params  
  
def preprocess\_songs(data\_folders):  
 *"""  
 Load and preprocess the songs from the data folders and turn them into a dataset of samples/pitches and lengths of the tones.* ***:param*** *data\_folders:* ***:return****:  
 """* all\_samples = []  
 all\_lengths = []  
  
 # keep some statistics  
 succeeded = 0  
 failed = 0  
 ignored = 0  
  
 # load songs  
 print("Loading songs...")  
 # walk folders and look for midi files  
 for folder in data\_folders:  
 for root, \_, files in os.walk(folder):  
 for file in files:  
 path = os.path.join(root, file)  
 if not (path.endswith('.mid') or path.endswith('.midi')):  
 continue  
  
 # turn midi into samples  
 try:  
 samples = midi\_utils.midi\_to\_samples(path)  
 except Exception as e:  
 print("ERROR ", path)  
 print(e)  
 failed += 1  
 continue  
  
 # if the midi does not produce the minimal number of sample/measures, we skip it  
 if len(samples) < 16:  
 print('WARN', path, 'Sample too short, unused')  
 ignored += 1  
 continue  
  
 # transpose samples (center them in full range to get more training samples for the same tones)  
 samples, lengths = music\_utils.generate\_centered\_transpose(samples)  
 all\_samples += samples  
 all\_lengths += lengths  
 print('SUCCESS', path, len(samples), 'samples')  
 succeeded += 1  
  
 assert (sum(all\_lengths) == len(all\_samples)) # assert equal number of samples and lengths  
  
 # save all to disk  
 print("Saving " + str(len(all\_samples)) + " samples...")  
 all\_samples = np.array(all\_samples, dtype=np.uint8) # reduce size when saving  
 all\_lengths = np.array(all\_lengths, dtype=np.uint32)  
 np.save('data/interim/samples.npy', all\_samples)  
 np.save('data/interim/lengths.npy', all\_lengths)  
 print('Done: ', succeeded, 'succeded,', ignored, 'ignored,', failed, 'failed of', succeeded + ignored + failed, 'in total')  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # configure parser and parse arguments  
 parser = argparse.ArgumentParser(description='Load songs, preprocess them and put them into a dataset.')  
 parser.add\_argument('--data\_folder', default=["data/raw"], type=str, help='The path to the midi data', action='append')  
  
 args = parser.parse\_args()  
 preprocess\_songs(args.data\_folder)

**Train**

#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-  
  
*"""  
Train an autoencoder model to learn to encode songs.  
"""*import argparse  
import random  
  
import numpy as np  
from matplotlib import pyplot as plt  
  
import midi\_utils  
import models  
import params  
import plot\_utils  
  
# Load Keras  
print("Loading keras...")  
import os  
import keras  
  
print("Keras version: " + keras.\_\_version\_\_)  
  
from keras.models import Model, load\_model  
#from keras.utils import plot\_model  
from keras import backend as K  
from keras.losses import binary\_crossentropy  
from keras.optimizers import Adam, RMSprop  
  
EPOCHS\_QTY = 10000  
EPOCHS\_TO\_SAVE = [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 140, 160, 180, 200, 250, 300, 350, 400, 450]  
LEARNING\_RATE = 0.001 # learning rate  
CONTINUE\_TRAIN = False  
GENERATE\_ONLY = False  
  
WRITE\_HISTORY = True  
NUM\_RAND\_SONGS = 10  
  
# network params  
DROPOUT\_RATE = 0.1  
BATCHNORM\_MOMENTUM = 0.9 # weighted normalization with the past  
USE\_EMBEDDING = False  
USE\_VAE = False  
VAE\_B1 = 0.02  
VAE\_B2 = 0.1  
  
BATCH\_SIZE = 350  
MAX\_WINDOWS = 16 # the maximal number of measures a song can have  
LATENT\_SPACE\_SIZE = params.num\_params  
NUM\_OFFSETS = 16 if USE\_EMBEDDING else 1  
  
K.set\_image\_data\_format('channels\_first')  
  
# Fix the random seed so that training comparisons are easier to make  
np.random.seed(42)  
random.seed(42)  
  
  
def vae\_loss(x, x\_decoded\_mean, z\_log\_sigma\_sq, z\_mean):  
 *"""  
 Variational autoencoder loss function.* ***:param*** *x:* ***:param*** *x\_decoded\_mean:* ***:param*** *z\_log\_sigma\_sq:* ***:param*** *z\_mean:* ***:return****:  
 """* xent\_loss = binary\_crossentropy(x, x\_decoded\_mean)  
 kl\_loss = VAE\_B2 \* K.mean(1 + z\_log\_sigma\_sq - K.square(z\_mean) - K.exp(z\_log\_sigma\_sq), axis=None)  
 return xent\_loss - kl\_loss  
  
  
def plot\_losses(scores, f\_name, on\_top=True):  
 *"""  
 Plot loss.* ***:param*** *scores:* ***:param*** *f\_name:* ***:param*** *on\_top:* ***:return****:  
 """* plt.clf()  
 ax = plt.gca()  
 ax.yaxis.tick\_right()  
 ax.yaxis.set\_ticks\_position('both')  
 ax.yaxis.grid(True)  
 plt.plot(scores)  
 plt.ylim([0.0, 0.009])  
 plt.xlabel('Epoch')  
 loc = ('upper right' if on\_top else 'lower right')  
 plt.draw()  
 plt.savefig(f\_name)  
  
  
def save\_training\_config(num\_songs, model, learning\_rate):  
 *"""  
 Save configuration of training.* ***:param*** *num\_songs:* ***:param*** *model:* ***:return****:  
 """* with open('results/config.txt', 'w') as file\_out:  
 file\_out.write('LEARNING\_RATE: ' + str(learning\_rate) + '\n')  
 file\_out.write('BATCHNORM\_MOMENTUM: ' + str(BATCHNORM\_MOMENTUM) + '\n')  
 file\_out.write('BATCH\_SIZE: ' + str(BATCH\_SIZE) + '\n')  
 file\_out.write('NUM\_OFFSETS: ' + str(NUM\_OFFSETS) + '\n')  
 file\_out.write('DROPOUT\_RATE: ' + str(DROPOUT\_RATE) + '\n')  
 file\_out.write('num\_songs: ' + str(num\_songs) + '\n')  
 file\_out.write('optimizer: ' + type(model.optimizer).\_\_name\_\_ + '\n')  
  
  
def generate\_random\_songs(decoder, write\_dir, random\_vectors):  
 *"""  
 Generate random songs using random latent vectors.* ***:param*** *decoder:* ***:param*** *write\_dir:* ***:param*** *random\_vectors:* ***:return****:  
 """* for i in range(random\_vectors.shape[0]):  
 random\_latent\_x = random\_vectors[i:i + 1]  
 y\_song = decoder([random\_latent\_x, 0])[0]  
 midi\_utils.samples\_to\_midi(y\_song[0], write\_dir + 'random\_vectors' + str(i) + '.mid', 32)  
  
  
def calculate\_and\_store\_pca\_statistics(encoder, x\_orig, y\_orig, write\_dir):  
 *"""  
 Calculate means, stddevs, covariance singular values (pca values), covariance singular vectors (pca vectors)  
 to more efficiently navigate/find configurations in the latent space.* ***:param*** *encoder:* ***:param*** *x\_orig:* ***:param*** *y\_orig:* ***:param*** *write\_dir:* ***:return****:  
 """* if USE\_EMBEDDING:  
 latent\_x = np.squeeze(encoder.predict(x\_orig))  
 else:  
 latent\_x = np.squeeze(encoder.predict(y\_orig))  
  
 latent\_mean = np.mean(latent\_x, axis=0)  
 latent\_stds = np.std(latent\_x, axis=0)  
 latent\_cov = np.cov((latent\_x - latent\_mean).T)  
 \_, latent\_pca\_values, latent\_pca\_vectors = np.linalg.svd(latent\_cov)  
 latent\_pca\_values = np.sqrt(latent\_pca\_values)  
  
 print("Latent Mean values: ", latent\_mean[:6])  
 print("Latent PCA values: ", latent\_pca\_values[:6])  
  
 np.save(write\_dir + 'latent\_means.npy', latent\_mean)  
 np.save(write\_dir + 'latent\_stds.npy', latent\_stds)  
 np.save(write\_dir + 'latent\_pca\_values.npy', latent\_pca\_values)  
 np.save(write\_dir + 'latent\_pca\_vectors.npy', latent\_pca\_vectors)  
 return latent\_mean, latent\_stds, latent\_pca\_values, latent\_pca\_vectors  
  
  
def generate\_normalized\_random\_songs(x\_orig, y\_orig, encoder, decoder, random\_vectors, write\_dir):  
 *"""  
 Generate a number of random songs from some normal latent vector samples.* ***:param*** *encoder:* ***:param*** *x\_orig:* ***:param*** *y\_orig:* ***:param*** *decoder:* ***:param*** *write\_dir:* ***:param*** *random\_vectors:* ***:return****:  
 """* latent\_mean, latent\_stds, pca\_values, pca\_vectors = calculate\_and\_store\_pca\_statistics(encoder, x\_orig, y\_orig, write\_dir)  
  
 latent\_vectors = latent\_mean + np.dot(random\_vectors \* pca\_values, pca\_vectors)  
 generate\_random\_songs(decoder, write\_dir, latent\_vectors)  
  
 title = ''  
 if '/' in write\_dir:  
 title = 'Epoch: ' + write\_dir.split('/')[-2][1:]  
  
 plt.clf()  
 pca\_values[::-1].sort()  
 plt.title(title)  
 plt.bar(np.arange(pca\_values.shape[0]), pca\_values, align='center')  
 plt.draw()  
 plt.savefig(write\_dir + 'latent\_pca\_values.png')  
  
 plt.clf()  
 plt.title(title)  
 plt.bar(np.arange(pca\_values.shape[0]), latent\_mean, align='center')  
 plt.draw()  
 plt.savefig(write\_dir + 'latent\_means.png')  
  
 plt.clf()  
 plt.title(title)  
 plt.bar(np.arange(pca\_values.shape[0]), latent\_stds, align='center')  
 plt.draw()  
 plt.savefig(write\_dir + 'latent\_stds.png')  
  
  
def train(samples\_path='data/interim/samples.npy', lengths\_path='data/interim/lengths.npy', epochs\_qty=EPOCHS\_QTY, learning\_rate=LEARNING\_RATE):  
 *"""  
 Train model.* ***:return****:  
 """* # Create folders to save models into  
 if not os.path.exists('results'):  
 os.makedirs('results')  
 if WRITE\_HISTORY and not os.path.exists('results/history'):  
 os.makedirs('results/history')  
  
 # Load dataset into memory  
 print("Loading Data...")  
 if not os.path.exists(samples\_path) or not os.path.exists(lengths\_path):  
 print('No input data found, run preprocess\_songs.py first.')  
 exit(1)  
  
 y\_samples = np.load(samples\_path)  
 y\_lengths = np.load(lengths\_path)  
  
 samples\_qty = y\_samples.shape[0]  
 songs\_qty = y\_lengths.shape[0]  
 print("Loaded " + str(samples\_qty) + " samples from " + str(songs\_qty) + " songs.")  
 print(np.sum(y\_lengths))  
 assert (np.sum(y\_lengths) == samples\_qty)  
  
 print("Preparing song samples, padding songs...")  
 x\_shape = (songs\_qty \* NUM\_OFFSETS, 1) # for embedding  
 x\_orig = np.expand\_dims(np.arange(x\_shape[0]), axis=-1)  
  
 y\_shape = (songs\_qty \* NUM\_OFFSETS, MAX\_WINDOWS) + y\_samples.shape[1:] # (songs\_qty, max number of windows, window pitch qty, window beats per measure)  
 y\_orig = np.zeros(y\_shape, dtype=np.float32) # prepare dataset array  
  
 # fill in measure of songs into input windows for network  
 song\_start\_ix = 0  
 song\_end\_ix = y\_lengths[0]  
 for song\_ix in range(songs\_qty):  
 for offset in range(NUM\_OFFSETS):  
 ix = song\_ix \* NUM\_OFFSETS + offset # calculate the index of the song with its offset  
 song\_end\_ix = song\_start\_ix + y\_lengths[song\_ix] # get song end ix  
 for window\_ix in range(MAX\_WINDOWS): # get a maximum number of measures from a song  
 song\_measure\_ix = (window\_ix + offset) % y\_lengths[song\_ix] # chosen measure of song to be placed in window (modulo song length)  
 y\_orig[ix, window\_ix] = y\_samples[song\_start\_ix + song\_measure\_ix] # move measure into window  
 song\_start\_ix = song\_end\_ix # new song start index is previous song end index  
 assert (song\_end\_ix == samples\_qty)  
 x\_train = np.copy(x\_orig)  
 y\_train = np.copy(y\_orig)  
  
 # copy some song from the samples and write it to midi again  
 test\_ix = 0  
 y\_test\_song = np.copy(y\_train[test\_ix: test\_ix + 1])  
 x\_test\_song = np.copy(x\_train[test\_ix: test\_ix + 1])  
 midi\_utils.samples\_to\_midi(y\_test\_song[0], 'data/interim/gt.mid')  
  
 # create model  
 if CONTINUE\_TRAIN or GENERATE\_ONLY:  
 print("Loading model...")  
 model = load\_model('results/history/model.h5')  
 else:  
 print("Building model...")  
  
 model = models.create\_autoencoder\_model(input\_shape=y\_shape[1:],  
 latent\_space\_size=LATENT\_SPACE\_SIZE,  
 dropout\_rate=DROPOUT\_RATE,  
 max\_windows=MAX\_WINDOWS,  
 batchnorm\_momentum=BATCHNORM\_MOMENTUM,  
 use\_vae=USE\_VAE,  
 vae\_b1=VAE\_B1,  
 use\_embedding=USE\_EMBEDDING,  
 embedding\_input\_shape=x\_shape[1:],  
 embedding\_shape=x\_train.shape[0])  
  
 if USE\_VAE:  
 model.compile(optimizer=Adam(lr=learning\_rate), loss=vae\_loss)  
 #elif params.encode\_volume:  
 #model.compile(optimizer=RMSprop(lr=learning\_rate), loss='mean\_squared\_logarithmic\_error')  
 else:  
 model.compile(optimizer=RMSprop(lr=learning\_rate), loss='binary\_crossentropy')  
 #model.compile(optimizer=RMSprop(lr=learning\_rate), loss='mean\_squared\_error')  
  
 # plot model with graphvis if installed  
 #try:  
 # plot\_model(model, to\_file='results/model.png', show\_shapes=True)  
 #except OSError as e:  
 # print(e)  
  
 # train  
 print("Referencing sub-models...")  
 decoder = K.function([model.get\_layer('decoder').input, K.learning\_phase()], [model.layers[-1].output])  
 encoder = Model(inputs=model.input, outputs=model.get\_layer('encoder').output)  
  
 random\_vectors = np.random.normal(0.0, 1.0, (NUM\_RAND\_SONGS, LATENT\_SPACE\_SIZE))  
 np.save('data/interim/random\_vectors.npy', random\_vectors)  
  
 if GENERATE\_ONLY:  
 print("Generating songs...")  
 generate\_normalized\_random\_songs(x\_orig, y\_orig, encoder, decoder, random\_vectors, 'results/')  
 for save\_epoch in range(20):  
 x\_test\_song = x\_train[save\_epoch:save\_epoch + 1]  
 y\_song = model.predict(x\_test\_song, batch\_size=BATCH\_SIZE)[0]  
 midi\_utils.samples\_to\_midi(y\_song, 'results/gt' + str(save\_epoch) + '.mid')  
 exit(0)  
  
 save\_training\_config(songs\_qty, model, learning\_rate)  
 print("Training model...")  
 train\_loss = []  
 offset = 0  
  
 for epoch in range(epochs\_qty):  
 print("Training epoch: ", epoch, "of", epochs\_qty)  
 if USE\_EMBEDDING:  
 history = model.fit(x\_train, y\_train, batch\_size=BATCH\_SIZE, epochs=1)  
 else:  
 # produce songs from its samples with a different starting point of the song each time  
 song\_start\_ix = 0  
 for song\_ix in range(songs\_qty):  
 song\_end\_ix = song\_start\_ix + y\_lengths[song\_ix]  
 for window\_ix in range(MAX\_WINDOWS):  
 song\_measure\_ix = (window\_ix + offset) % y\_lengths[song\_ix]  
 y\_train[song\_ix, window\_ix] = y\_samples[song\_start\_ix + song\_measure\_ix]  
 #if params.encode\_volume:  
 #y\_train[song\_ix, window\_ix] /= 100.0  
 song\_start\_ix = song\_end\_ix  
 assert (song\_end\_ix == samples\_qty)  
 offset += 1  
  
 history = model.fit(y\_train, y\_train, batch\_size=BATCH\_SIZE, epochs=1) # train model on reconstruction loss  
  
 # store last loss  
 loss = history.history["loss"][-1]  
 train\_loss.append(loss)  
 print("Train loss: " + str(train\_loss[-1]))  
  
 if WRITE\_HISTORY:  
 plot\_losses(train\_loss, 'results/history/losses.png', True)  
 else:  
 plot\_losses(train\_loss, 'results/losses.png', True)  
  
 # save model periodically  
 save\_epoch = epoch + 1  
 if save\_epoch in EPOCHS\_TO\_SAVE or (save\_epoch % 100 == 0) or save\_epoch == epochs\_qty:  
 write\_dir = ''  
 if WRITE\_HISTORY:  
 # Create folder to save models into  
 write\_dir += 'results/history/e' + str(save\_epoch)  
 if not os.path.exists(write\_dir):  
 os.makedirs(write\_dir)  
 write\_dir += '/'  
 model.save('results/history/model.h5')  
 else:  
 model.save('results/model.h5')  
  
 print("...Saved.")  
  
 if USE\_EMBEDDING:  
 y\_song = model.predict(x\_test\_song, batch\_size=BATCH\_SIZE)[0]  
 else:  
 y\_song = model.predict(y\_test\_song, batch\_size=BATCH\_SIZE)[0]  
  
 plot\_utils.plot\_samples(write\_dir + 'test', y\_song)  
 midi\_utils.samples\_to\_midi(y\_song, write\_dir + 'test.mid')  
  
 generate\_normalized\_random\_songs(x\_orig, y\_orig, encoder, decoder, random\_vectors, write\_dir)  
  
 print("...Done.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # configure parser and parse arguments  
 parser = argparse.ArgumentParser(description='Train to reconstruct midi in autoencoder.')  
 parser.add\_argument('--samples\_path', default='data/interim/samples.npy', type=str, help='Path to samples numpy array.')  
 parser.add\_argument('--lengths\_path', default='data/interim/lengths.npy', type=str, help='Path to sample lengths numpy array.')  
 parser.add\_argument('--epochs\_qty', default=EPOCHS\_QTY, type=int, help='The number of epochs to be trained.')  
 parser.add\_argument('--learning\_rate', default=LEARNING\_RATE, type=float, help='The learning rate to train the model.')  
  
 args = parser.parse\_args()  
 epochs\_qty = args.epochs\_qty  
 learning\_rate = args.learning\_rate  
 samples\_path = args.samples\_path  
 lengths\_path = args.lengths\_path  
 train(samples\_path, lengths\_path, epochs\_qty, learning\_rate)