令和元年度　卒業論文

AI による音楽自動生成の研究

函館工業高等専門学校　生産システム工学科・情報コース
12 番　倉﨑　文平
指導教員　東海林　智也

目次

第 1 章　序論

第 1 節　英文アブストラクト

It is a purpose to generate music using deep learning which is one of AI techniques. In this research, development was done in python environment using Visual Studio. Tensorflow was used as the AI library. The five input songs are divided into melodies and accompaniments, and the accompaniment is trained as teacher data and the melodies as labels. After that, the randomized accompaniment is input to the model and the output melody and accompaniment are output to a MIDI file. In conclusion, we were able to make music, but it was not accurate and we had to think about improving accuracy. Probably due to lack of teacher data and input data.

Key words : AI, deep learning, Tensorflow

第 2 節　研究目的

本研究は、自分自身の python と AI への理解を深め、ディープラーニングを用いた音楽生成を研究することを目的としている。そのため、音源を用意すれば簡単に音楽を自動生成できるツールの作成を試みる。

第 3 節　研究背景

昨今、社会では AI によって人々から仕事を奪ってしまうのではないかと懸念されるほど AI の発達は目覚ましい。そのような AI や Python などの技術や、ピアノを習っていたことから音楽作成に興味があったため、AI 用いた音楽生成をテーマとして研究を進めることとした。

第 4 節　開発環境

使用 PC：

OS：macOS Mojave

CPU：1.4 GHz Intel Core i5

実装 RAM：4.00GB

開発言語：Python

開発環境：Visual Studio Code

AI ライブラリ：Tensorflow

第 2 章　関連技術

第 1 節　ディープラーニング

ディープラーニングは機械学習とよばれる技術の一つであり、DNN（Deep Neural Network）というシステムを利用している。DNN は人間の神経細胞の構造を元にして作ったニューラルネットワークを多層化したシステムで入力層、中間層、出力層の 3 つに分けられる。特に中間層が 1 層だけのニューラルネットワークのことを三層ニューラルネットワークと呼ぶ(図 1)。

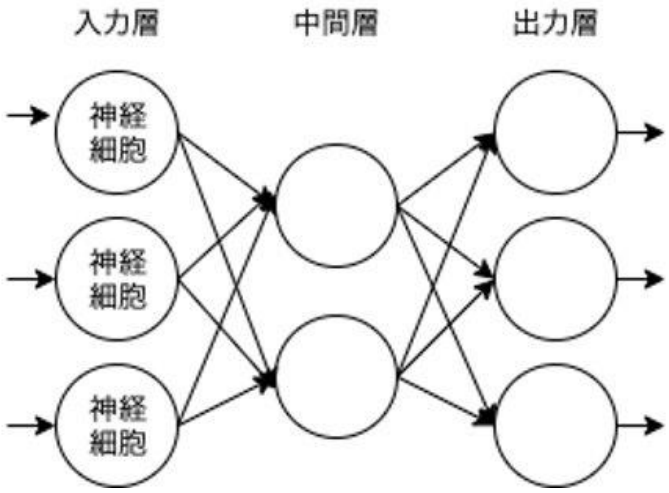

図 1　三層ニューラルネットワーク

第 2 節　Tensorflow

TensorFlow は、2015 年に Google が開発した機械学習ライブラリである。TensorFlow は「データフローグラフ」を作成して機械学習を行う。データフローグラフとは図 2 のようなグラフである。データフローグラフを作成した後、グラフにデータを入力することによって結果が出力される。

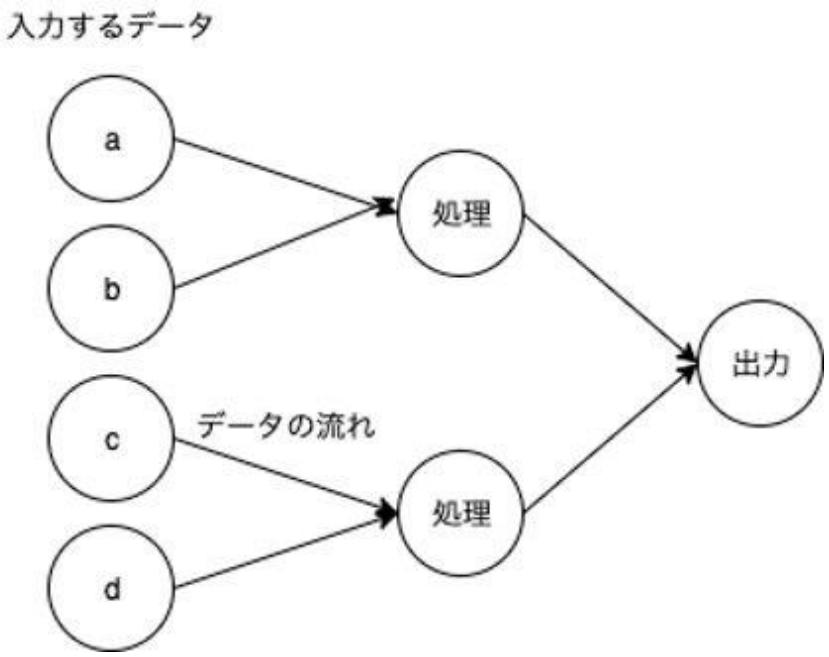

図 2　データフローグラフ

第 3 節　pretty_midi

pretty_midi とは MIDI データを処理するため python ライブラリである。PrettyMIDI() でファイル名を指定することによって、トラックとノートをリストで取得できる。トラックごとにファイルを分けたり、特定の楽器の音や名前、ドラムかそうでないかなどを判別してあてはまるトラックごとに取り出し別ファイルにしたりすることもできる。楽器の変更やトラック分けはこの関数で行う。

第 4 節　deepjazz

deepjazz は自動ジャズジェネレーターである。Deepjazz は LSTM を使用し、MIDI ファイルを入力ファイルとして与えると音楽を学習してオリジナルのジャズを作曲する AI である。LSTM(Long short-term memory)はリカレントニューラルネットワークの一種であり、図 1 で示した三層ニューラルネットワークの中間層の出力が、ひとつ前の中間層の出力と入力されたデータによって決まるという時系列を考慮したネットワーク構造である。

第 3 章　音楽生成プログラムの開発

第 1 節　プログラムの概要

本研究ではフィードフォワードニューラルネットワークを使用する。このニューラルネットワークは入力層、中間層、出力層の流れを単一方向に行うプログラムである。場合によって中間層の数は増えるが本研究では 1 層のみの三層ネットワーク(図 1)を用いる。今回のプログラムでは入力層、中間層、出力層のパーセプトロンを 20,10,20 と設定する。

入力側教師データと出力側教師データを 32 ビットの浮動小数点数の要素として定数テンソルにする。次に中間層と出力層の重みとバイアスを設定する。重みとは入力値ごとに決められ、その入力値の価値を決めるものである。バイアスは値を偏らせるために広く同じ値を設定する際に使う。中間層の重みは 20 行 10 列,バイアスは 1 行 10 列で出力層の重みは 10 行 20 列,バイアスは 1 行 20 列でそれぞれ平均値 0、標準偏差 0.1 の正規乱数で初期化する。次に三層ニューラルネットワークを作成する。まず入力層は恒等関数とし受け取った値をそのまま返す。次に中間層で入力された信号と重みを掛ける。その後バイアスの値を足す。そして活性化関数に通して出力される。活性化関数はシグモイド関数を利用した。その後出力された値を出力層へ送る。出力層は中間層と同じく入力された信号に重みを掛けてバイアスを足す。この層では活性化関数に通さずそのまま出力する。その後に誤差と勾配降下法を用いてディープラーニングを実行する。誤差とは入力に対して正しい答えがあるとき、自分のモデルと実データとの差異のことである。勾配降下法は誤差を最小化するために微分を繰り返すことである。TensorFlow では学習率と誤差を入力することによって勾配降下法を簡単に行うことができる関数が存在するためそれを用いる。学習率は学習を行う精度と速度を表している。学習率の値が大きいほどニューラルネットワークは適当に学習するが速く学習が進み、逆に値が小さいとニューラルネットワークはしっかり学習するが遅く学習が進む。今回の学習率は 0.01 と設定した。

ここまででテンソルを活用してデータフローグラフを構築が完了したため、ここからそのデータフローグラフを学習させてモデルを完成させるためセッションを開始する。今回の学習回数は 1000 回とする。ディープラーニング中は学習回数分重みとバイアスを更新し続けて最も誤差が少なくなる値を探す。ディープラーニング終了後は完成したモデルに入力するメロディーパートを通し、出力された値を main 文へ返す。作成したモデルの学習の流れを図 3 に示す。

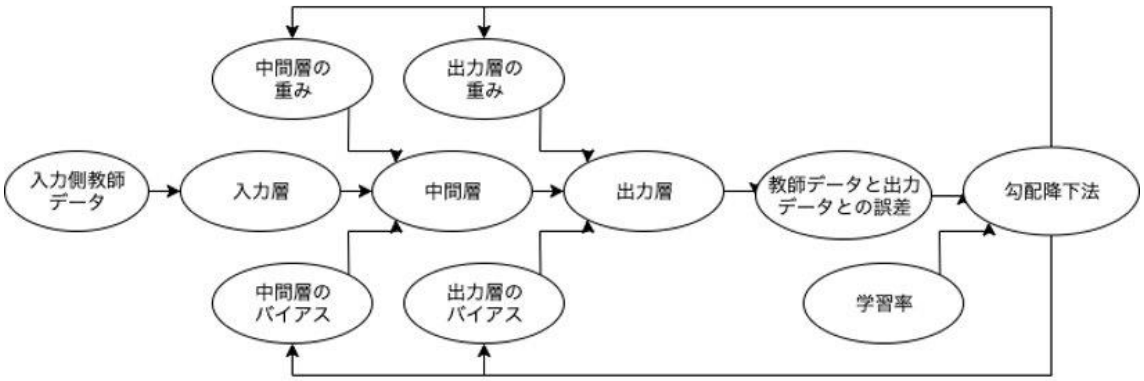

図 3 作成したモデルの学習の流れ

第 2 節　作成したプログラム

図 3 のモデルを使い、音階、速度、音階、音の開始時間、音の終了時間を入力し 1 音ずつ出力するプログラムを作成した。教師データはメロディパートと伴奏パートにあらかじめ分けた 5 曲を使用する。図 4 にモデルを示す。

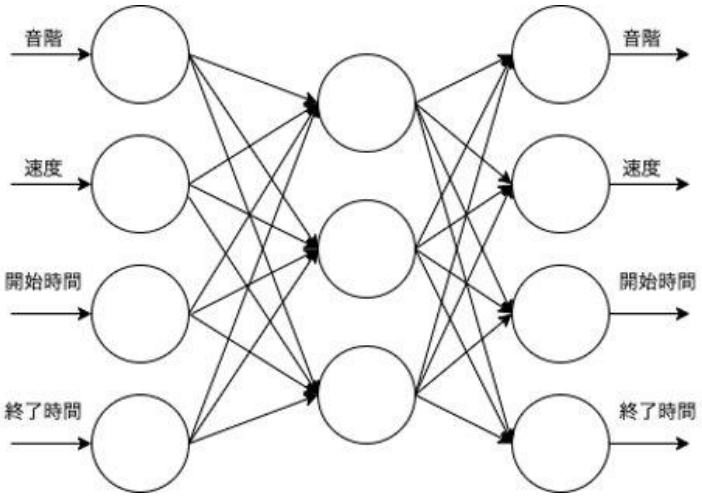

図 4 作成したプログラムのモデル

第 4 章　結果

入力データをわかりやすくするために今回はきらきらぼしのメロディー部分を与え教師データにはフリー音源サイトから集めたジャズを中心とした 5 曲を用いてプログラムを動かした。学習回数は 100,1000,10000 と変更し学習させたところ、モデルは問題なく動き、Midi ノート番号でいうと 50 から 60、中央のドから下 10 音が出力された。曲として聞くと一部あっているように聞こえるが不協和音となってしまっている部分が多く曲として完成しているとは言い難いものが出来上がってしまった。また開始時間がと測度がうまくいっておらず複数の音が同時に鳴ってしまうなどの不具合が見られた。表 1 に入力側教師信号、表 2 に出力側教師信号、表 3 に結果を示す。

表 1 入力側教師信号

[[69.0000000000 65.0000000000 62.0000000000 60.0000000000 46.0000000000
55.0000000000 58.0000000000 62.0000000000 65.0000000000 51.0000000000
38.0000000000 64.0000000000 60.0000000000 57.0000000000 53.0000000000
57.0000000000 60.0000000000 53.0000000000 38.0000000000 64.0000000000]
[57.0000000000 62.0000000000 58.0000000000 65.0000000000 43.0000000000
64.0000000000 60.0000000000 57.0000000000 55.0000000000 41.0000000000
60.0000000000 64.0000000000 57.0000000000 41.0000000000 55.0000000000
62.0000000000 53.0000000000 58.0000000000 57.0000000000 48.0000000000]
[71.0000000000 55.0000000000 62.0000000000 64.0000000000 71.0000000000
55.0000000000 62.0000000000 64.0000000000 71.0000000000 55.0000000000
62.0000000000 64.0000000000 69.0000000000 55.0000000000 60.0000000000
64.0000000000 69.0000000000 55.0000000000 60.0000000000 64.0000000000]
[69.0000000000 55.0000000000 60.0000000000 64.0000000000 64.0000000000
48.0000000000 55.0000000000 57.0000000000 64.0000000000 48.0000000000
55.0000000000 57.0000000000 64.0000000000 48.0000000000 55.0000000000
57.0000000000 64.0000000000 48.0000000000 55.0000000000 57.0000000000]
[55.0000000000 71.0000000000 59.0000000000 62.0000000000 74.0000000000
55.0000000000 79.0000000000 62.0000000000 59.0000000000 74.0000000000
50.0000000000 72.0000000000 57.0000000000 54.0000000000 69.0000000000
50.0000000000 54.0000000000 57.0000000000 67.0000000000 55.0000000000]
[71.0000000000 62.0000000000 59.0000000000 55.0000000000 74.0000000000
59.0000000000 62.0000000000 79.0000000000 52.0000000000 71.0000000000
55.0000000000 59.0000000000 69.0000000000 67.0000000000 52.0000000000
55.0000000000 59.0000000000 64.0000000000 48.0000000000 52.0000000000]
[76.0000000000 82.0000000000 76.0000000000 82.0000000000 76.0000000000
79.0000000000 72.0000000000 76.0000000000 70.0000000000 74.0000000000
71.0000000000 76.0000000000 69.0000000000 74.0000000000 71.0000000000
76.0000000000 74.0000000000 79.0000000000 71.0000000000 76.0000000000]
[69.0000000000 79.0000000000 71.0000000000 76.0000000000 72.0000000000
76.0000000000 70.0000000000 74.0000000000 72.0000000000 76.0000000000
91.0000000000 94.0000000000 91.0000000000 94.0000000000 88.0000000000
91.0000000000 90.0000000000 93.0000000000 88.0000000000 91.0000000000]
[35.0000000000 35.0000000000 35.0000000000 30.0000000000 35.0000000000
30.0000000000 33.0000000000 33.0000000000 33.0000000000 33.0000000000
33.0000000000 33.0000000000 33.0000000000 33.0000000000 33.0000000000
33.0000000000 28.0000000000 33.0000000000 28.0000000000 33.0000000000]
[33.0000000000 33.0000000000 33.0000000000 33.0000000000 33.0000000000
33.0000000000 33.0000000000 33.0000000000 33.0000000000 28.0000000000
33.0000000000 28.0000000000 30.0000000000 30.0000000000 30.0000000000
30.0000000000 30.0000000000 30.0000000000 30.0000000000 30.0000000000]

表 2 出力側教師信号

[[62.0000000000 64.0000000000 64.0000000000 64.0000000000 63.0000000000
63.0000000000 64.0000000000 64.0000000000 62.0000000000 64.0000000000
64.0000000000 64.0000000000 64.0000000000 63.0000000000 63.0000000000 64.0000000000
64.0000000000 62.0000000000 64.0000000000 64.0000000000 64.0000000000]
[63.0000000000 63.0000000000 64.0000000000 64.0000000000 62.0000000000
64.0000000000 64.0000000000 64.0000000000 63.0000000000 63.0000000000
64.0000000000 64.0000000000 62.0000000000 64.0000000000 64.0000000000
64.0000000000 63.0000000000 63.0000000000 64.0000000000 64.0000000000]
[64.0000000000 54.0000000000 68.0000000000 69.0000000000 42.0000000000
54.0000000000 64.0000000000 69.0000000000 42.0000000000 64.0000000000
54.0000000000 61.0000000000 69.0000000000 42.0000000000 67.0000000000
61.0000000000 54.0000000000 69.0000000000 64.0000000000 42.0000000000]
[54.0000000000 64.0000000000 69.0000000000 42.0000000000 64.0000000000
54.0000000000 67.0000000000 69.0000000000 42.0000000000 64.0000000000
54.0000000000 60.0000000000 69.0000000000 42.0000000000 63.0000000000
54.0000000000 67.0000000000 69.0000000000 42.0000000000 54.0000000000]
[60.0000000000 60.0000000000 55.0000000000 60.0000000000 60.0000000000
48.0000000000 60.0000000000 60.0000000000 55.0000000000 60.0000000000
60.0000000000 48.0000000000 60.0000000000 60.0000000000 55.0000000000
60.0000000000 60.0000000000 48.0000000000 60.0000000000 60.0000000000]
[55.0000000000 60.0000000000 60.0000000000 48.0000000000 60.0000000000
60.0000000000 55.0000000000 60.0000000000 60.0000000000 48.0000000000
60.0000000000 60.0000000000 55.0000000000 60.0000000000 60.0000000000
48.0000000000 60.0000000000 60.0000000000 55.0000000000 60.0000000000]
[51.0000000000 51.0000000000 51.0000000000 51.0000000000 51.0000000000
37.0000000000 51.0000000000 51.0000000000 51.0000000000 51.0000000000
51.0000000000 51.0000000000 37.0000000000 51.0000000000 51.0000000000
51.0000000000 51.0000000000 51.0000000000 51.0000000000 37.0000000000]
[51.0000000000 51.0000000000 51.0000000000 51.0000000000 51.0000000000
51.0000000000 37.0000000000 51.0000000000 51.0000000000 51.0000000000
51.0000000000 51.0000000000 51.0000000000 37.0000000000 51.0000000000
51.0000000000 51.0000000000 51.0000000000 49.0000000000 49.0000000000]
[51.0000000000 35.0000000000 51.0000000000 35.0000000000 51.0000000000
41.0000000000 35.0000000000 54.0000000000 51.0000000000 49.0000000000
35.0000000000 54.0000000000 54.0000000000 54.0000000000 54.0000000000
51.0000000000 35.0000000000 54.0000000000 54.0000000000 54.0000000000]
[54.0000000000 51.0000000000 42.0000000000 35.0000000000 54.0000000000
42.0000000000 54.0000000000 46.0000000000 54.0000000000 54.0000000000
51.0000000000 42.0000000000 35.0000000000 54.0000000000 54.0000000000
35.0000000000 54.0000000000 54.0000000000 51.0000000000 35.0000000000]]

表 3 結果

結果
[56 55 57 51 55 51 54 58 52 56 54 55 55 52 58 53 55 58 55 51]

第 5 章　課題と考察

実行結果より今回作成したプログラムの問題点として教師データが少なく偏りが大きくなってしまったことと、音階のみに注目したことが挙げられる。教師データについては現在いろいろな曲を試している最中であり、音階にのみ注目した件については音の長さ、速度などを用いる必要があると理解した。そこでフィードフォワードニューラルネットワークを用いた自動生成を音階だけでなく、音の開始時間と終了時間から計算された一音あたりの長さと速度を合わせた重回帰分析を行うモデルと、deepjazz にも用いられている中間層の出力を記憶し、ひとつ前の中間層と現在入力された信号によって出力が決まるリカレントニューラルネットワークを用いたモデルを作成中である。

また本研究を通じて TensorFlow と keras を用いていろいろなアーキテクチャを勉強してきたが、世の中には自己複合化器という意味の Autoencorder や畳み込みニューラルネットワーク、学習データになるべく近いデータを生成させる生成器と、生成された偽物と(学習元のデータ)を識別する識別器の二つのネットワークを互いに競わせる生成的敵対ネットワークなど様々なものが存在する。今後の課題としてそれらへの理解を深めつつ、新たにモデルを作成していくことが必要だと思われる。

参考文献

[1]Tensorflow ホームページ
https://www.tensorflow.org/
[2]deepjazz ホームページ
https://deepjazz.io/
[3] Deep Learning を用いた音楽生成手法のまとめ［サーベイ］
https://medium.com/@naotokui/deep-learning%E3%82%92%E7%94%A8%E3%81%84%E3%81%9F%E9%9F%B3%E6%A5%BD%E7%94%9F%E6%88%90%E6%89%8B%E6%B3%95%E3%81%AE%E3%81%BE%E3%81%A8%E3%82%81-%E3%82%B5%E3%83%BC%E3%83%99%E3%82%A4-1298d29f8101
[4] 東海林教授ホームページ
https://tmytokai.github.io/open-ed/
[5] pretty-midi ホームページ
http://craffel.github.io/pretty-midi/

付録

ソースコード

main.py

```
import re
import tensorflow as tf
import numpy as np
import pretty_midi
import sys
import random
import separate
import outp
import model_sn
import models


def main(args):

    piano = []
    melody = []
    pnote = []
    mnote = []
    data_fn1 = 'midi/' + 'original_metheny.mid'
    data_fn2 = 'midi/' + 'jazzy034.mid'
    data_fn3 = 'midi/' + 'Dab_Rag.mid'
    data_fn4 = 'midi/' + 'The_Stranger.mid'
    data_fn5 = 'midi/' + 'ADA_3.mid'
    data_in = 'midi/' + 'hurusato.mid'

    print("piano1 melody1")
    piano1,melody1,start1,end1,velocity1 = separate.separates(data_fn1)
    print("piano2 melody2")
    piano2,melody2,start2,end2,velocity2 = separate.separates(data_fn2)
    print("piano3 melody3")
    piano3,melody3,start3,end3,velocity3 = separate.separates(data_fn3)
    print("piano4 melody4")
    piano4,melody4,start4,end4,velocity4 = separate.separates(data_fn4)
    print("piano5 melody5")
    piano5,melody5,start5,end5,velocity5 = separate.separates(data_fn5)

    inm,s,e,v = separate.sepa(data_in)
    pnote = piano1 + piano2 + piano3 + piano4 + piano5
    mnote = melody1 + melody2 + melody3 + melody4 + melody5

    print("piano 入力")
    print(pnote)
    print("")
    print("melody 入力")
    print(mnote)
    print("")

    rp =  models.models(inm,mnote,pnote)
    outp.output(rp,data_in,s,e,v)

if __name__ == '__main__':
    import sys
    main(sys.argv)
```

separate.py

```
import pretty_midi
import random


def separates(data_fn):

        # MIDI ファイルを読み込む
        midi_data = pretty_midi.PrettyMIDI(data_fn)
        min = 0
        max = 0
        for instrument in midi_data.instruments:
                if instrument.is_drum == 0:
                        if max < len(instrument.notes):
                                Melody = instrument
                                max = len(instrument.notes)
                elif instrument.is_drum == 1:
                        note = instrument.notes
                        hik = 0
                        long = 5
                        if len(note) < 5:
                                long = 0
                        for x in range(long):
                                hik += abs(note[x].pitch - note[x+1].pitch)
                        if min <= hik and len(note) > 39:
                                Piano = instrument
                                min = hik

       notes1 = Melody.notes
        notes2 = Piano.notes
        p = []
        m = []
        s = []
        e = []
        v = []
        for x in range(20,60):
                if x == 40:
                        piano = [p]
                        melody = [m]
                        start = [s]
                        end = [e]
                        p = []
                        m = []
                        s = []
                        e = []
                        p.append(notes2[x].pitch)
                        m.append(notes1[x].pitch)
                        s.append(notes1[x].start)
                        e.append(notes1[x].end)
                        v.append(notes1[x].velocity)
                else:
                        p.append(notes2[x].pitch)
                        m.append(notes1[x].pitch)
                        s.append(notes1[x].start)
                        e.append(notes1[x].end)
                        v.append(notes1[x].velocity)

        piano.append(p)
        melody.append(m)
        start.append(s)
        end.append(e)

        print(piano)
        print(melody)
        return piano,melody,start,end,v
```

```
def sepa(data_fn):

        # MIDI ファイルを読み込む
        midi_data = pretty_midi.PrettyMIDI(data_fn)
        data = midi_data.instruments
        notes1 = data[0].notes
        m = []
        s = []
        e = []
        v = []
        for x in range(20,40):
                if x == 40:
                        melody = [m]
                        start = [s]
                        end = [e]
                        m = []
                        s = []
                        e = []
                        m.append(notes1[x].pitch)
                        s.append(notes1[x].start)
                        e.append(notes1[x].end)
                        v.append(notes1[x].velocity)
                else:
                        m.append(notes1[x].pitch)
                        s.append(notes1[x].start)
                        e.append(notes1[x].end)
                        v.append(notes1[x].velocity)

        melody.append(m)
        start.append(s)
        end.append(e)

        print(melody)
        return melody,start,end,v
```

models2.py

```
import tensorflow as tf
import numpy as np

def input_layer( x ):
    return x

def hidden_layer( x, w, b ):
    op_matmul = tf.matmul(x,w)
    op_add = tf.add(op_matmul,b)
    op_sigmoid = tf.sigmoid( op_add )
    return op_sigmoid

def output_layer( x, w, b ):
    op_matmul = tf.matmul(x,w)
    op_add = tf.add(op_matmul,b)
    return op_add




def models(input1,teach,lavel):
    np.set_printoptions(precision=10,suppress=True,floatmode='fixed')
    # セッション作成
    sess = tf.Session()
    # パーセプトロンの数
    N = 20
    K = 10
```

```
    M = 20
    # 教師信号
    op_const_teacher = tf.constant(teach, tf.float32)
    # ラベル
    op_const_label = tf.constant(lavel, tf.float32)

    # 隠れ層の重みとバイアス
    op_var_W_h = tf.Variable( tf.random_normal( [N, K], mean=0.0, stddev=0.1
) )
    op_var_B_h = tf.Variable( tf.random_normal( [1, K], mean=0.0, stddev=0.1
) )

    # 出力層の重みとバイアス
    op_var_W_o = tf.Variable( tf.random_normal( [K, M], mean=0.0, stddev=0.1
) )
    op_var_B_o = tf.Variable( tf.random_normal( [1, M], mean=0.0, stddev=0.1
) )

    # 3 層ニューラルネットワーク作成
    op_input_layer = input_layer(op_const_teacher)
    op_hidden_layer = hidden_layer(op_input_layer, op_var_W_h,op_var_B_h)
    op_output_layer = output_layer(op_hidden_layer, op_var_W_o,op_var_B_o)

    # 学習率
    r = 0.01
    # 勾配降下法 OP ノードの作成
    loss = tf.reduce_mean(tf.square(op_output_layer - op_const_label))
    op_grad_optimizer = tf.train.GradientDescentOptimizer( r ).minimize(loss)

    # セッション開始
    sess.run( tf.initialize_all_variables() )
    print('教師信号')
    print( sess.run( op_const_teacher ) )
    print('ラベル')
    print( sess.run( op_const_label ) )
    #print('教師信号の判別結果(学習前)')
    result = sess.run( op_output_layer )
    for i in range( len(result) ):
        print(result[i])
        print('ディープラーニング中･･･')
    for i in range( 1000 ):
        if i == 10 or i == 100 or i == 1000 or i == 9999:
            print(sess.run( loss ))
        sess.run( op_grad_optimizer )
    #print('教師信号の判別結果(学習後)')
    result = sess.run( op_output_layer )

    # 入力信号
    op_const_data = tf.constant(input1, tf.float32)

    # 学習済の重みとバイアスを利用して入力信号を入力
    op_input_layer = input_layer(op_const_data)
    op_hidden_layer = hidden_layer(op_input_layer, op_var_W_h,op_var_B_h)
    op_output_layer = output_layer(op_hidden_layer, op_var_W_o,op_var_B_o)
    #op = np.sum(((op_output_layer-t)**2)/2)/x.shape[0]
    print('入力信号')
    result = sess.run( op_input_layer )
    print(result)
    print('結果')
    op = tf.cast(op_output_layer, tf.int32)
    result = sess.run( op )
    print(result)

    return result
```

outp.py

```
import numpy as np
import pretty_midi
import itertools

def output(piano,melody,S,N,V):

    P = np.ravel(piano)
    s = np.ravel(S)
    n = np.ravel(N)
    v = np.ravel(V)

    data = pretty_midi.PrettyMIDI(melody)
    data = data.instruments[0].notes

    print("piano 出力")
    print(P)
    print("")
    print("melody 出力")
    print("")
    Piano_chord  = pretty_midi.PrettyMIDI()
    Music_chord  = pretty_midi.PrettyMIDI()

    Piano_program = pretty_midi.instrument_name_to_program('Acoustic Grand Pi
ano')
    Melody_program = pretty_midi.instrument_name_to_program('Acoustic Grand P
iano')

    Piano = pretty_midi.Instrument(Piano_program)
    Melody = pretty_midi.Instrument(Melody_program)

    for x in range(15):
        note = pretty_midi.Note(velocity=data[x].velocity, pitch=data[x].pitc
h, start=data[x].start-data[0].start, end=data[x].end-data[0].end)
        Melody.notes.append(note)
    for x in range(15):
        note = pretty_midi.Note(velocity=v[x], pitch=P[x], start=data[x].star
t-data[0].start, end=data[x].end-data[0].end)
        Piano.notes.append(note)

    Music_chord.instruments.append(Melody)
    Music_chord.write('hMelody.mid')
    Piano_chord.instruments.append(Piano)
    Music_chord.write('hPiano.mid')
    Music_chord.instruments.append(Piano)
    Music_chord.write('hMusic.mid')
```