JAIST Repository
https://dspace.jaist.ac.jp/

Title	リズムゲームの上達を支援するコンテンツ自動生成法
Author(s)	梁，鈺彬
Citation	
Issue Date	2019-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/15957
Rights	
Description	Supervisor:池田　心，先端科学技術研究科，修士（情報科学）

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修士論文

リズムゲームの上達を支援するコンテンツ自動生成法

1610211　LIANG YUBIN

主指導教員　池田 心
審査委員主査　池田 心
審査委員　飯田 弘之
長谷川 忍
白井 清昭

北陸先端科学技術大学院大学
先端科学技術研究科
(情報科学)

平成 31 年 2 月

Abstract

Rhythm game is a genre of music-themed (action video) game in which players play by taking actions in accordance with rhythm and music. Since music or songs are familiar to ordinary people, it is easy for people to understand how to play such games. In addition, both of easy stage and hard stage can be created from one music, therefore, rhythm game becomes a popular game genre in the whole world.

In many cases, the contents (required action and its timing) of rhythm game are handcrafted by human designers from music material. Also, there are countless pieces of music, but only a part of them have already been used as game contents. Therefore, in our opinions, automatic contents generation is required.

Apart from this, it is frequently pointed out that rhythm games are hard to practice. For example, assume that a player is not good at a part (5 seconds) of a rhythm game stage. Even when he wants to repeat only the part, it is impossible and he needs to play the whole stage (3 minutes). This is one of reasons that rhythm games are hard to practice.

Based on those existing requirements, we propose a self-training support system with automatic content generation capability. The system possesses capabilities corresponding to each task such as "generate content from audio file", "evaluate the level of a player", "modify the action combinations in content to produce appropriate difficulty".

In this research, we proposed an approach that generates contents automatically from music materials by deep learning. We used supervised learning method which inputs audio data, outputs timestamps (timing of action) and action type. As a machine learning task, this task has some difficulties such as, 1) even when the same music is used as an input, the outputs may be completely different, due to author variations and level variations, 2) proportion of positive/negative samples is ill. To deal with those tasks, in this research we adopt that, 1) handle the difficulty settings as one input feature, 2) using fuzzy labels to increase positive samples. By applying the proposed training methods, it has been proved that the training performance and prediction accuracy were increased. The combined training method has improved the F-score of the timestamp prediction from 0.8159 (by the existing method) up to 0.8430.

Furthermore, in this research we proposed various methods to support self-training for players. First, after a player played some games, his/her mistakes will be analyzed and shown to the player, from several viewpoints such as, "frequent

mistake types (too late, too early, pushed a wrong button and so on)", "the situations which the player often failed (many actions, frequently changing buttons, many long press actions and so on)", or "what kind of action combinations are difficult for the player". Those analysis results allow players to notice their own weaknesses.

Besides, to make the player's practice more efficient, we implemented a function to increase proportion of difficult action combinations in the next content generation. Finally, regarding the self-training support capabilities, we conducted a questionnaire on subjects such as "was the analysis result accurate?", "were weak action combinations increased?", to verify the usefulness of the proposed methods, and received positive answers.

概要

リズムゲームとは，リズムや音楽に合わせてプレイヤがアクションをとることで進行する形式のゲームである．リズムゲームは誰にでも馴染みやすい「音楽」を主な題材として扱っており，遊び方が感覚的に解りやすく，同じ曲でも簡単なステージや難しいステージを作れるため，広い層に人気のゲームジャンルとなっている．

多くの場合リズムゲームのコンテンツ（求められるアクションとそのタイミング）は，音楽素材から人間デザイナーが作成しており，素材となる音楽が無数にあったとしても，ゲーム上で遊べるコンテンツの数は限られている場合も多い．そのため，作成の自動化が要請される．

これとは別に，リズムゲームでは練習の困難さがしばしば指摘される．すなわち，一つの曲は通常数分程度は続き，その中には個人ごとに得意なアクション組み合わせと苦手なアクション組み合わせが登場することもあるが，その苦手なものだけを練習することは通常できない．

それらの要請から，我々は上達支援機能を備えたリズムゲーム自動生成システムを提案する.「音声ファイルからコンテンツを生成する」「プレイヤの実力を把握する」「適切な難易度になるようにコンテンツ中のアクションの組み合わせを調整する」といった課題それぞれに対応する機能を備えたコンテンツ自動生成システムである．

本研究では，ディープラーニングを主な手法として，音楽素材からコンテンツを自動的に生成することを目的として研究を行った．我々は音声データを入力，タイムスタンプ（アクションのタイミング）と種類を出力とする教師あり学習を行った．機械学習問題としては，1) 学習データに用いたコンテンツの難易度がさまざまで，同じ音楽でも出力が全く異なるものがあること，2) アクションが求められるタイムスタンプ（正例）の割合が非常に低いこと，という困難さがある．これに対し，本研究では，1) 難易度を特別な入力として扱うこと，2) 曖昧ラベルを与えて正例の焼き増しを行うこと，で対応する．本研究の学習手法を用いて，学習性能と予測正確率が上がったことが分かった．学習手法を組み合わせることによって，タイムスタンプの予測について既存手法では 0.8159 だった予測の F-score を 0.8430 まで向上させることができた．

さらに本研究では，プレイヤの上達を支援するための様々な工夫を提案した．まず第一に，プレイヤがミスを犯した部分について,「どのような種類のミスなのか（遅すぎ・早すぎ・横ずれなど）」「どのような状況でミスしやすいのか（アクション数が多い・ボタン変化が多い・長押しが多いなど）」あるいは「具体的にどのようなアクションのパターンが苦手なのか」という多様な分析を行い，それをプレイヤに示せるようにした．これによってプレイヤは弱点に気付くことができるよ

うになった．そのうえで，次回以降のコンテンツ生成で苦手なパターンを増やすことができるようにして，練習を効率的に行えるようにした．最後に，上達支援機能について，“分析結果は的確だったか”，“自分の苦手なところが増えたか”などに関して被験者アンケートを行い，構築したシステムの有用性を検証し，肯定的な回答を得た．

目 次

図 目 次

表 目 次

第1章　はじめに

リズムゲーム [1] とは，リズムや音楽に合わせてプレイヤがアクションをとることで進行する形式のゲームである．リズムゲームは誰にでも馴染みやすい「音楽」を主な題材として扱っており，遊び方が感覚的に解りやすく，同じ曲でも簡単なステージや難しいステージを作れるため，広い層に人気のゲームジャンルとなっている．

多くの場合リズムゲームのコンテンツ（求められるアクションとアクションのタイミング）は，音楽素材から人間デザイナーが作成しており，素材となる音楽が無数にあったとしても，ゲーム上で遊べるコンテンツの数は限られている。さらに，プレイヤの好みの音楽のものがなかったり，仮に複数の難易度があったとしても，適した難易度がなかったりする場合も多い．

本研究では，ディープラーニングを主な手法として，音楽素材からコンテンツを自動的に生成することを目的として研究を行った．我々は音声データを入力，タイムスタンプ（アクションのタイミング）と種類を出力とする教師あり学習を行った．機械学習問題としては，1) 学習データに用いたコンテンツの難易度がさまざまで，同じ音楽でも出力が全く異なるものがあること，2) アクションが求められるタイムスタンプ（正例）の割合が非常に低いこと，という困難さがある．これに対し，本研究では，既存研究 [5] も参考に 1) 難易度を特別な入力として扱うこと，2) 曖昧ラベルを与えて正例の焼き増しを行うこと，で対応する．

さらに，リズムゲームを上達したいプレイヤにとって，リズムゲームは練習が難しいという課題もある．ステージの一部分だけが苦手な場合にそこだけを練習することが難しいなどの理由である．それは，遊んだコンテンツをうまく省みられなく，一箇所を練習するために最初からやり直す必要もある．本研究はこういった練習に不向きなところを課題とする．そして,「遊んでもらい」「ログを分析し」「苦手なところを見つけてうまく説明し」「苦手なところを効率よく練習させる」という流れで，プレイヤの上達を支援する機能を構築した．

具体的には，プレイヤの実力を測るにあたり，用いる評価尺度を定義する.「プレイヤが行ったアクションがどれくらい上手なのか」「プレイヤがどんな間違い方をするのか」「どんな状況が苦手なのか」「どんなアクションパターンが苦手なのか」など様々な面から分析を行い，苦手なポイントを見つけて説明する．その上，コンテンツの中にプレイヤ個人にとっての簡単なアクションパターンと苦手なアクションパターンを検出，簡単な部分に苦手なアクションパターンを多めに入れ換えるなど「効率向上」の手法を検討し，実装する．

本論文は，2 章に用いるシミューレーション用リズムゲームの紹介及び関連研究について述べる．3 章に提案システムの概要及び全体図について述べる．4 章では，本研究に用いる学習データについて述べる．5 章から 6 章までは，本研究に用いるリズムゲームのコンテンツの自動生成手法について述べる．7 章に本研究が提案する上達支援手法について述べる．最後は 8 章に本研究をまとめ，今後の課題について述べる．付録では，音楽のメロディ抽出手法，及びそれに基づく可能な Beatmap 自動生成改善案を述べる．

第2章　背景

2.1　リズムゲーム:「OSU!」

与えられたコンテンツのみならず，プレイヤがコンテンツを作ることもできるオープンソースのリズムゲーム「OSU!」（「」は正式名称に含まれていない）が2007年に開発された[2]．現在,「OSU!」の登録アカウントは1千万を超え，世界中にいろんなコンテストが不定期に開催されている．

「OSU!」のゲームコンテンツはBeatmapと呼ばれる単位で扱われ，音楽に対して,「どのタイミングで」「どのアクションを取るべきか」が記録されている.「OSU!」のゲーム形式はいくつかあり，本研究で想定するのはそのうちの一つmaniaモード（図2.1）である．このモードでは，上部から落下する「マーカー」が「判定ライン」と重なる瞬間にアクションする（ボタンを押すなど）ことが求められる．このタイミングが適切なら高得点となり，押し間違えればミスとなる．ほとんどの場合このタイミングは音楽のリズムと同期しているため，楽曲を自分が演奏しているような臨場感を味わうことができる．

本研究では,「OSU!」のmaniaモードの4Kマップ（4 key）において，音楽素材からBeatmapを自動的に生成する研究を行う．4Kマップとは，操作するボタンが4つのゲームコンテンツである．ボタンの操作は，押してすぐ離すものと，“長押し”と呼ばれるものがある．長押しでは，離すタイミングも適切でなければならない．

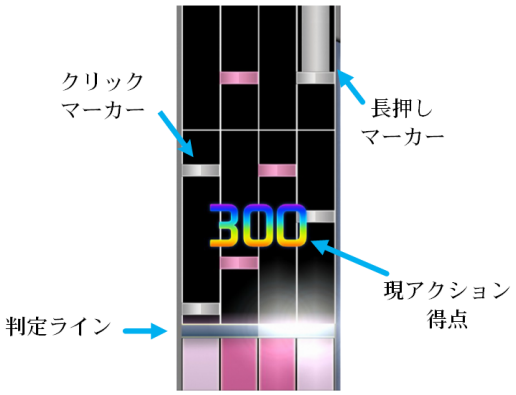


図 2.1:「OSU!」のmaniaモード
プレイ中のキャプチャー

2.2 関連研究

Jan らの論文 [4] では，ディープラーニングを用いて音の始まり（musical onsets）を抽出する実験が行われた．そして，リズムゲームでのアクションタイミングの選出は一般的に音楽の中に一番はっきり聞こえた音やリズムに合わせており，音の始まりや変化するタイミングがそれに当てはまることが分かった．この研究はリズムゲームコンテンツの自動生成の際には音の特徴を捉えることが重要であることを示している．

Chris らの論文 [5] では，LSTM（Long short-term memory）[6] を用いて Dance Dance Revolution というリズムゲームのコンテンツの自動生成が行われている．LSTM は循環ニューラルネットワークユニットの 1 種で，時間上の勾配消失を有効に抑えられ，音声や動画などの時系列データに適している．しかしこれらの自動作成法では，数通りの難易度を指定できるものの,「ある（アクション組み合わせの）パターンは得意だが、あるパターンは苦手」といった“ 個人の癖 ”には対応できていない．リズムゲームのプレイヤの各々の習熟段階に合わせて上達を支援するようなコンテンツ自動生成法が求められている．

多くのリズムゲームでは，長押しというアクションが存在する．一般的に音楽の長い音に対応しており，これを適切に自動生成するには音楽のメロディの分析が有益である．論文 [5] では，アクション生成におけるメロディ情報の有効性を検討したものの，応用までは至っていない．一方 Salamon らの論文 [7] では,（リズムゲームとは関係なく）音楽の主メロディを抽出する手法について述べている．本研究では，これを用い，音楽の中の長い音を検出し，長押しアクションの生成に援用する試みを行う．

上達支援のためには，どんなゲームであれ，プレイヤの実力を測ったり，苦手を見つけて提示したり，上達のための道筋を示したりそのための環境を与えることが有益である．Ikeda らの論文 [8] では，囲碁における初級者中級者の悪手を検出し，その理由を説明する試みを行っている．まずは人間初級者プレイヤのプレイデータを収集し，指導経験豊富な上級者がそこから“ 指導すべき悪手 ”を選びだし，“ なぜ悪いのか ”理由を 10 程度の選択肢からラベル付けしてもらう．その上で，盤面と着手の特徴量を入力とし，指導すべきかどうかと，悪さの理由を出力とする 2 種類の教師あり学習を行った．結果的には上級者に迫る精度の悪手判別やラベル付けが可能になったと述べられている．そのうえで，悪手が導く結末を見せて，それを防ぐ手段を説明するなど，指導システムが提案されている．

本研究ではこれらの既存研究を踏まえ，音楽情報からメロディ抽出も援用してリズムゲームコンテンツを生成し，さらに上達支援のためにプレイヤのミスを分類，苦手を計測，さらに苦手を重点的に練習できるシステムを提案する．

第3章　提案システム概要

現在、多くのリズムゲームのコンテンツは、音楽素材から人間デザイナーが作成しているため、素材となる音楽が無数にあっても，ゲームの中で遊べるコンテンツの数は限られている。また，一つの曲は通常数分程度は続き，その中には個人ごとに得意なアクションパターンと苦手なアクションパターンが登場することもあるが，その苦手なものだけを練習することは通常できない．例えば，ピアノの練習であれば一曲のうち苦手な場所だけを繰り返し弾くことができるが，一般のリズムゲームでは一曲の最初から該当部位に至るまでやりなおす必要がある．それらの現状を踏まえ，リズムゲームのプレイヤの各々の習熟段階に合わせて上達を支援するようなコンテンツ自動生成法が求められていると考えた．

本章では，前述した上達支援システムの全体図を説明する．上達支援システムは,「Beatmap 生成」「苦手分析」「Beatmap 調整」といった 3 つのモジュールに分けられる．

A Beatmap 生成（図 3.1）は 2 ステップに分けて行う．音楽ファイルからタイムスタンプ（アクションを置くべきタイミング）を生成し，各タイムスタンプに適切なアクション種類を生成する 2 ステップである．このモジュールの詳細は第 5 章と第 6 章に説明する．

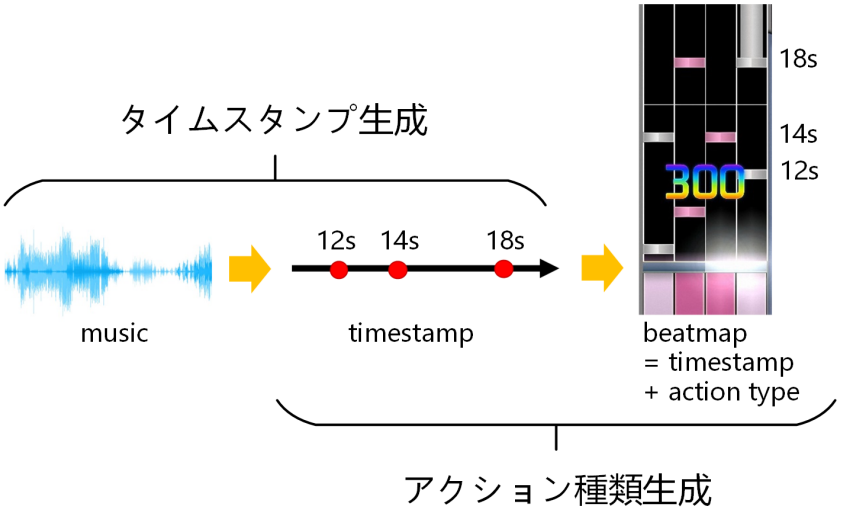


図 3.1: Beatmap 生成の全体概念図

B 実力分析にあたり，プレイログ解析にタイミング誤差とミス種別，Beatmap に Beatmap 属性とアクションパターン，合計 4 つの評価尺度を用い，プレイヤの実力を多角的に分析する．十分なデータが記録された後に，プレイヤの実力を分析し，分析結果を実力シートとしてプレイヤに分かりやすく説明する機能を備えてある．このモジュールの詳細は第 7 章に説明する．

実力分析結果と説明は以下の内容を含む（図 3.2 例）:

- perfect/great/normal/miss といった**タイミング誤差**に対する評価の統計結果．本研究では誤差が 56ms 以内は Perfect，108ms 以内は Great，160ms 以内は Normal，160ms より大きいのは MISS（間違い）と判定する．
- 良くする**ミス種別**，例えば「よく早く押してしまう」などプレイヤ個人的な癖を分析する．
- 苦手な**状況の特徴**，例えば,「短時間内多量なアクション」が出たら間違いしやすくなる．
- 苦手な**アクションパターン**，特定なアクション組み合わせである（例に参照）．

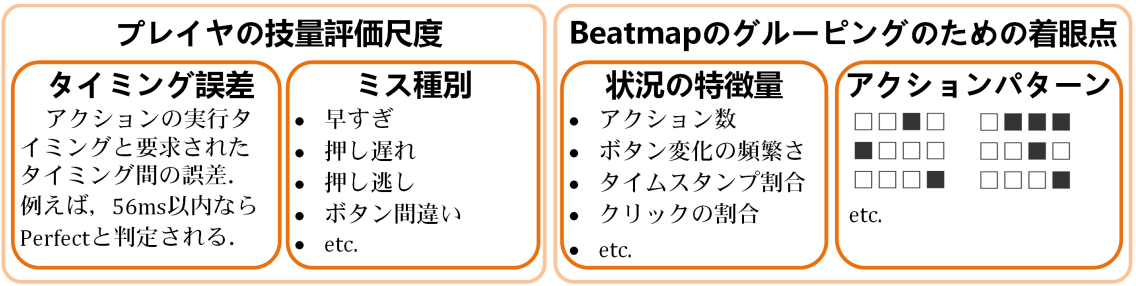


図 3.2: 実力分析に用いる評価尺度の概要

C Beatmap 調整（図 3.3）は，プレイヤ個人の実力を基づいて未知な Beatmap の難易度を予測し,「苦手なパターンを増やす（調整例 1）」「難易度を増やす（調整例 2）」といった上達支援目標を向けて Beatmap を調整する．このモジュールの詳細は第 7 章に説明する．

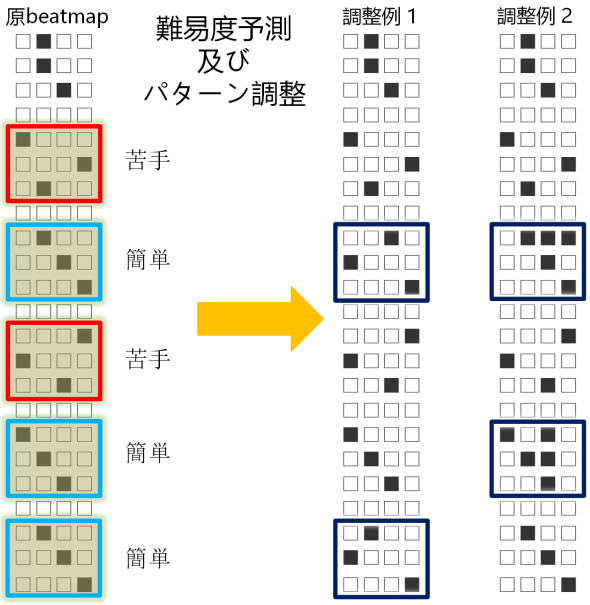


図 3.3: Beatmap 調整の概念図

そして，図 3.4 に全体像を示し，説明は概ね図中の数字に対応している．

0) 事前に,「Beatmap の生成」モジュールのオフライン学習を実行する．これは全プレイヤ共通に行い，以降はプレイヤごと個別に行う．

1) プレイヤが「自分の好みの音楽ファイル（wav, mp3 など）」と「初期難易度」をシステムに入力する．

2) 生成モジュールが万人向けの Beatmap を生成し，個人への対応はしない．初期段階ではデータベースは空もしくは意味のないほどに少ないので，Beatmap は調整せず「OSU!」のメインシステムに与えられる (5 へ)．

3) 調整モジュールが，データベースから過去のそのプレイヤのプレイ履歴を参照する，各プレイヤ向けの調整を行う．

4) 調整された Beatmap が,「OSU!」のメインシステムに与えられる．調整モジュールが未作動の時は調整せずに与える．

5) プレイヤは「OSU!」を通じて生成された Beatmap をプレイする．

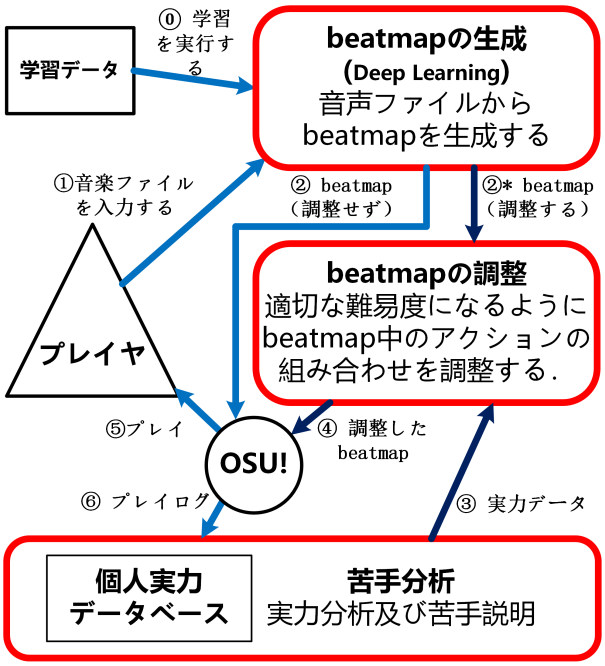


図 3.4: 提案システムの全体フレームワークおよびシステム作動の流れ図

6) そのプレイ結果はデータベースに蓄積され，そのプレイヤの実力が多角的に測定される (1 へ).

*) データが蓄積されていくと，そのプレイヤにあった難易度調整や，苦手パターンの追加が行えるようになる.

第4章　用いるデータの特徴と前処理

本章では，本研究に用いる学習データの詳細について説明する．さらに，同じ音楽の入力に対して，出力の Beatmap の難易度がさまざまであることを踏まえ，教師あり学習ための前処理を行う．

4.1　学習データの詳細

本研究の学習データは「OSU!」のホームページ [3] から収集した．アクセス時点まで累計 10 万回以上プレイされた（つまり人気の高い）Beatmap パックを集計対象とし，学習データとして前処理を行う．一般的に，Beatmap パックは音楽 1 曲と，異なる難易度の Beatmap を複数含んでおり，プレイヤが同じ曲に対して，“ある程度は ” 自分のできる難易度に合わせて選ぶことができる．学習データの統計を表 4.1 に示す．

なお，“長押し ” は，押すことと離すことの 2 つの離れたタイムスタンプがあるので，これらを別のアクションとしてカウントした．すなわち，各アクションは（クリック，長押し始まり，長押し終わり）の種別と，タイミングを持つということである．本研究の学習データは，特定・少数のデザイナーではなく，アマチュアも含む世界中の多数の作者によって作られたものである．そのため，作者の個性や作風の違いは非常に多様であり，同じ曲・同じ難易度であっても全くタイムスタンプや種類・配置が違う Beatmap が提供されていることもある．これらは教師あり学習を行う際には不都合な特徴であり，正解率 100%は原理上無理である．

データセット	
作者数	約 300 人
曲数	473
全曲の合計長さ	約 1067 分
Beatmap 数	1655
全 Beatmap の合計長さ	約 3584 分
アクション数	約 169 万
アクション数/秒	約 7.85

表 4.1: 学習データの統計

4.2 Beatmapの“難易度”の定義

学習データには同じ曲についても様々な「プレイヤが感じる難易度」のものが含まれている．また，我々は同じ曲について様々な難易度のBeatmapを作成したい．これらの要請から，音楽データとは別に「well-definedな難易度」パラメータを入力して好みの難易度のBeatmapを得られるようにする．このために，教師あり学習もその難易度ごとにデータを分割して独立に行うことにする．

プレイ時に「プレイヤが感じる難易度」に影響する要素は多い．その要素は概ね以下の5つにまとめられる，

(1) アクション組み合わせの複雑さ

(2) 単位時間内のアクションの数（密度）

(3) バー（図2.1参照）が判定ラインへ移動するスピード

(4) アクションのミス判定の厳しさ．押すべきタイミングと実際に押されたタイミングの許容される誤差．

(5) ミスが許容される回数

このうち(3)〜(5)は，コンテンツであるBeatmapというよりは，ゲームシステムの設定である．本研究では，(1)および(2)の項目のみに着目する．

(1)と(2)のうち，(1)は人により得手不得手がありスカラ化も困難であるため，我々はwell-definedな難易度として仮に(2)密度のみを用いることにする．学習データ1655 Beatmapを，密度つまり秒あたりのアクション数を（3個以下，3から17個までは8等分，および17個以上の）10段階に分けたとき，各密度段階の占めるBeatmap数のヒストグラムを図4.1に示す．

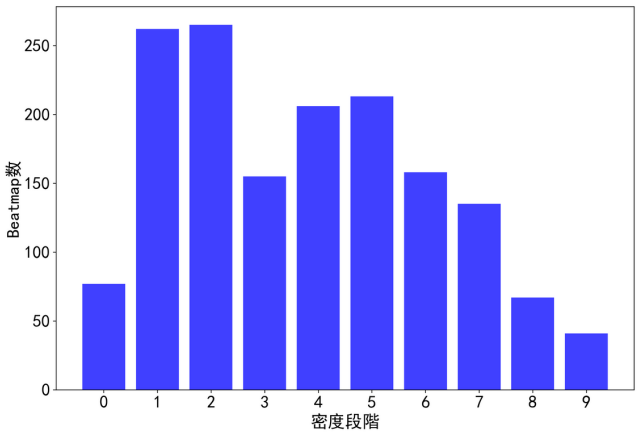


図 4.1: 各密度のBeatmap数．
非常に簡単なものや非常に難しいものは数が少ない．

第5章　タイムスタンプ生成

Beatmap の生成の仕組みについて，我々はこれを二段階の教師あり学習で行う．まず音声データと難易度を入力，タイムスタンプ（アクションのタイミング）を出力とする教師あり学習を行う．続いて，タイムスタンプを入力とし，アクションの種類を出力とする教師あり学習を行う．二段階を通して Beatmap の生成モジュールを構築する．

本章では，図 5.1 に示すように，音楽ファイルからタイムスタンプ（アクション配置すべきタイミング）を生成する詳細を説明する．

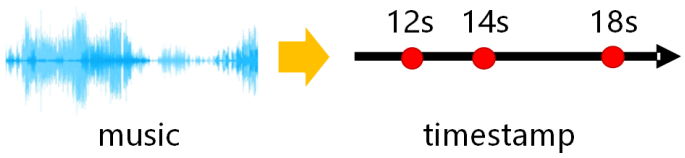


図 5.1: タイムスタンプ生成の概念図

5.1　既存手法

5.1.1　音楽特徴量と学習ラベル

第一段階の教師あり学習では，波形データを入力としてタイムスタンプを出力とする．本研究では過去の論文 [4][5] の方法に基本的に従い，対象問題の違いを考慮した工夫を加える．生の波形データから，特徴量としてメル尺度 [9] を抽出して用いる．抽出の帯域は 27.5Hz から 16kHz まで，80 バンドのメルフィルタバンクを用いてメル尺度を求める．3 種類の時間幅 23ms，46ms，93ms を用い，ウィンドウの移動幅すなわち 1 フレームは 10ms とする．すなわち，1 秒あたり 100 回，そこにアクションがあるかどうかを推定するわけである．音楽の連続性から，各フレームの前後 7 フレームを特徴量として用いる．すなわち，10ms ごとに，全部で 15 フレーム× 80 バンド× 3 ウィンドウサイズの入力があることになる．

学習ラベルは，クリック，押し，離しのタイムスタンプを同じタイムスタンプとし，区別しない．これらを区別し，4 つあるボタンのうちどれを押させるか（図 2.1 参照）を決定したりは第二段階の教師あり学習の担当である．

5.1.2　タイムスタンプ予測のディープラーニング

タイムスタンプ生成のニューラルネットワークは論文 [5] に述べられた C-LSTM モデルである．LSTM は循環ニューラルネットワークユニットの 1 種で，これを用いることにより，現タイミングの前後の音声情報も含めて学習させることができる．これは，タイムスタンプを予測するときに現在の音声情報だけではなくそれまでの音楽情報も必要であるために導入されている．

C-LSTM の概念図を図 5.2 に示す．入力となる音楽特徴量は 15 × 80 × 3 つの 3 次元位相データである．第 1 層の畳みカーネルは 7 × 3 × 3，マックスプールは 1 × 3，出力は 9 × 26 のフィーチャーマップ 10 個である．第 2 層の畳みカーネルは 3 × 3 × 10，マックスプールは 1 × 3，出力は 7 × 8 のフィーチャーマップ 20 個である．各 LSTM 層の入力に難易度（密度）を表す 10 ユニットの one-hot ベクトルを加え，ここの値を変えることで同じ音楽データであっても出力されるタイムスタンプの難易度を調整できるようにする．畳み層の次は 200 ユニットの LSTM 層を 2 層用いる．LSTM 層の次，256 ユニットと 128 ユニットの全結合層を用いる．最後は 1 ユニットの Sigmoid 出力，該当フレームがタイムスタンプに選ばれる確率である．畳み層と全結合層の活性化関数は ReLU，LSTM 層の活性化関数は tanh を利用する．

図 5.2: C-LSTM フレームワーク

論文 [5] に述べられた 2 つのデータセットにおいて，0.756 と 0.721 の F-score[13] を得られた．実際，作者の個性や作風によって，同じ曲・同じ難易度であっても全くタイムスタンプや種類・配置が違う Beatmap が作られることもあって，この F-score は低いとは言えない．

本研究ではこの既存手法 [5] を基本として，いくつかの工夫を加える．対象とするゲームや用いるデータセットが異なるため，上記の F-score との比較は適切ではない．そのため，既存手法を再現したうえで，それと提案手法との比較を 5.3 節で行う．

5.1.3　タイムスタンプの選別

ニューラルネットワークで出力されるのは，該当 10ms 長さのフレームがタイムスタンプ選ばれる確率，あるいは“もっともらしさ”である．実際に用いる場合には，ここから実際にアクションを置くかどうかを決定しなければならない．これには，閾値パラメータにより，ある確率以上ならアクションを置くことにする．閾値が高すぎればアクションは少なくなりすぎ，低すぎれば多くなりすぎる．

実際の出力例を図 5.3 に示す．横軸が時系列，縦軸が出力された確率である．赤点は局所的な最高点，峰である．緑線は学習データにおける正解（アクションがあったタイミング）である．我々は，学習時に「閾値決定用データ」を取り置き，それを用いてアクションを決定するための閾値を求めることにする．すべての閾値決定用データにおいて正しく判定することはできないので，精度と再現度のバランスを F-score で表し，それが最も高くなるように閾値を求めることにする，例えば，図 5.3 の例であれば，閾値として 0.6 程度を使用すれば，（ここだけでの）F-score は 1 となる．

なお，最終的な性能評価には，学習データ・閾値決定用データとは別に取り置いたテストデータを用いる．

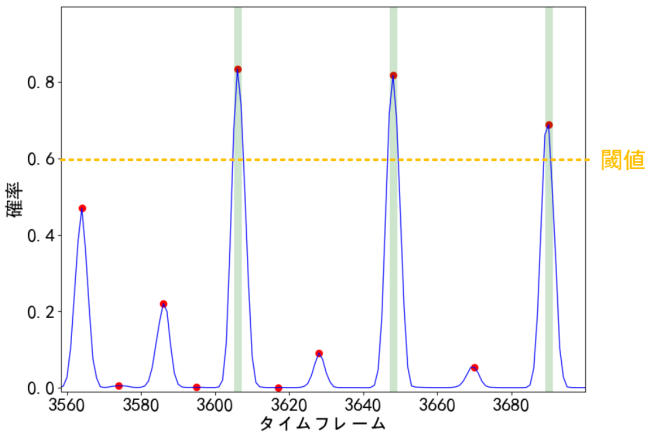


図 5.3: 予測結果における閾値の選び方の例

5.2　性能改善の試み

論文 [4] の既存手法の F-score は悪いものではないが，我々の研究では用いるゲームやデータセットの特徴に基づき，2 つの発展的手法を提案する．一つは 1 秒あたりのアクション数が少ない（学習ラベルの正例が少ない）データセットに対応するものであり，一つは長押しアクションの存在に対応するものである．

5.2.1　曖昧ラベル

本研究のデータセットは，平均しても 1 秒あたりのアクション数は約 7.85 個にすぎない．つまり，100 個のラベルの中，正例は 7.85 個（1 割足りない）しかない．

それは 1 秒を分ける精度にもよるが，例えば精度を 50ms にすれば，20 個のラベルの中，正例は 7.85 個（4 割ぐらい）もある．だが，フレーム長さを長くすれば音楽とアクションのずれが生じてリズムに乗れなくなるため，本研究は 10ms を 1 タイミングフレームとする．こういった正例が少ない学習データを学習することは一般的に困難で，負例の間引きなどが用いられることもある．

本研究では，正例の焼き増しにあたる曖昧ラベルを提案する．曖昧ラベルとは，図 5.4 に示されるように，0（そこにアクションはない）か 1（そこにアクションはある）かのような急激な変化ではなく，ある程度徐々に変化するラベルである．10ms という小さいフレーム幅であれば，1 フレーム前や後ろにアクションが推定されることは“間違い”ではなくて“惜しい”というべきである．こういった状態は評価時には正解として扱うが（論文 [5] でも同じである），それを明示的に学習に取り入れたい．このような“もっともらしさ”を曖昧ラベルは表現でき，時間順が要求されるデータにおいて適性があると考える．

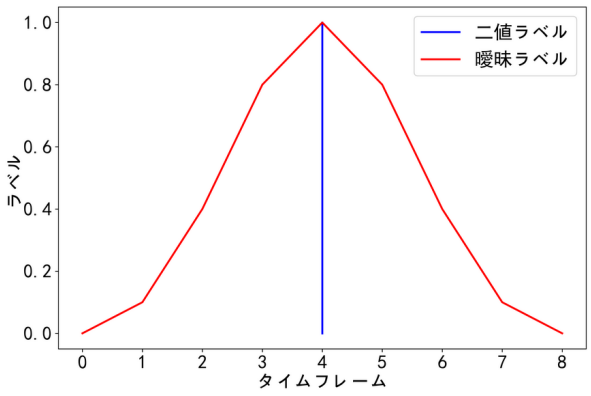


図 5.4: 曖昧ラベル
連続的な変化で正例を増やす

5.2.2 双方向 LSTM

既存手法では，片方向の LSTM を使い，「それまでの（過去の）」音声情報を用いてタイムスタンプを予測する．しかし，本研究で用いる「OSU!」では“長押しアクション”の数が大量であり，そのため，未来の音声情報を考慮しなければならない．長押しアクションは例えば「ある音がこれから長く続くとき」に始まることが多いアクションであり，それを知るためには「それまでの」音声情報の他に「これからの（未来の）音声情報も必要になる．既存手法を「OSU!」に用いた予備実験ではその点がうまくいかないことが多く見られたため，本研究では未来の情報も用いることにする．

なお，過去や将来は入力特徴量としても 7 フレーム分含まれている．当然ながら，入力の幅を大きくするのも一案であるが，計算量が大きくなり，効率が悪くなる．本研究では，図 5.2 を基づいて，工夫を加えて双方向 LSTM[10] とする．双方向 LSTM を用いることにより，タイムスタンプ予測において，先読みすることができる．その上，計算量が既存手法の C-LSTM と大体同じである．

長押しタイムスタンプを適切に判定するには将来の波形も考慮に入れる双方向 LSTM(C-BLSTM 図 5.5) が有効だと考えた．LSTM は現時刻の入力だけではなく，以前の入力の情報も記録しており，前後の入力が関連している場合において，現時刻の予測の正解率を改善できる．

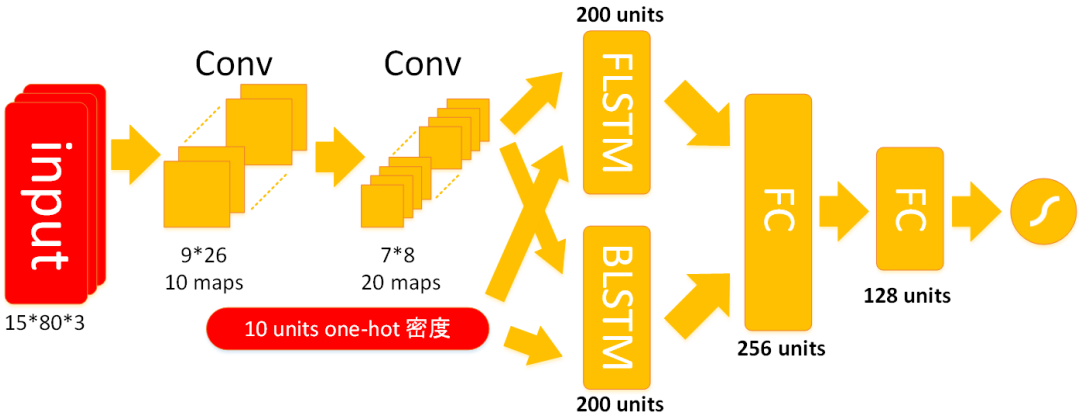


図 5.5: C-BLSTM フレームワーク

5.3 実験

5.3.1 実験設定

本研究では，難易度（密度）ごとのデータ数が概ね揃うように，以下の配分を行った．閾値決定用データには，各難易度 4 曲 4 Beatmap，全部で 40 曲 40 Beatmap を用いた．テストデータにも同様，各難易度 4 曲 4 Beatmap，全部で 40 曲 40 Beatmap を用いた．学習データは，これら 80 曲を除き，残り 393 曲と Beatmap 1321 本（8 割ぐらい）である．これはグループの間は同じ音楽を使うものがないようにしているためである．

図 4.1 にあるように，難易度区分ごとに多くの Beatmap があるものと少ないものがあるが，学習の際にはこれらがほぼ同数となることが望ましいと考えた．そこで，最も多くの Beatmap がある難易度に合わせ，足りない難易度区分では，そのデータをコピー（焼き増し）することで学習させることにした（図 5.6）．

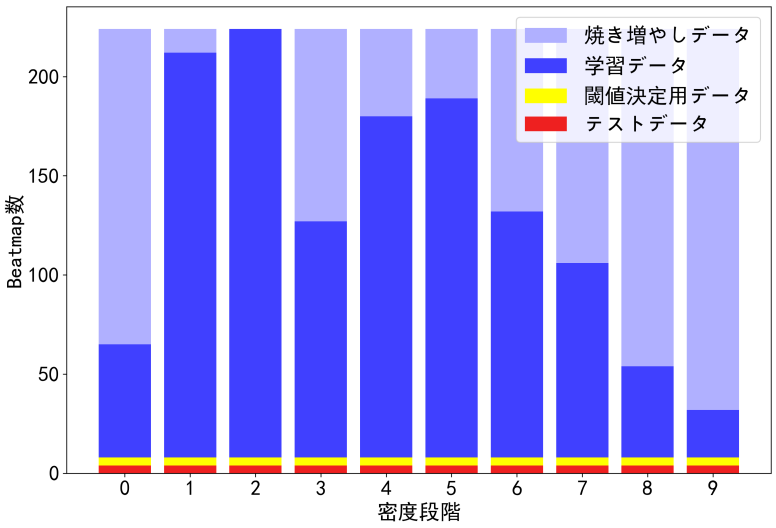


図 5.6: 用いるデータのグループ分け
および焼き増やしデータの例

音声処理ライブラリは librosa 0.5.1[11]，ニューラルネットワークライブラリは TensorFlow 1.2[12] を使用した．実験の評価にあたり，我々は各手法のテストデータにおける micro F-score[13] を用いる．

ニューラルネットワークの学習において，バッチサイズは 256 にする．入力データはウィンドウかつバンドごとに Z-Score[14] を用いて標準化する．各 LSTM 層に 50%のドロップアウト [15] を用いる．

5.3.2 実験結果

比較するのは，オリジナルの C-LSTM，これに双方向 LSTM の工夫を加えたもの，曖昧ラベルの工夫を加えたもの，両方を加えたものの 4 つである．

図 5.7 に，閾値決定用データの F-score の推移を示す．図 5.7 から，曖昧ラベルを用いるとエポック数進んでも過学習が生じにくくなることが分かった．逆に，双向 LSTM は，最高性能は向上するが，過学習に注意する必要があることが分かった．

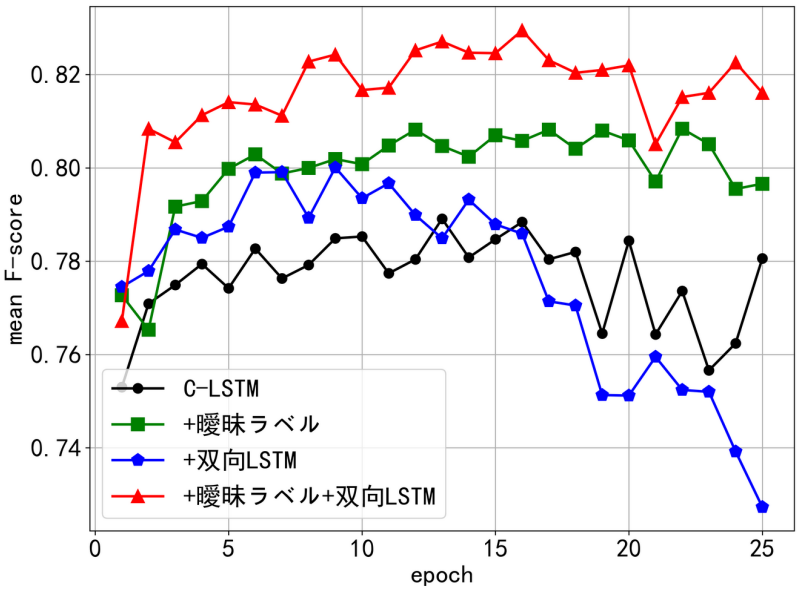


図 5.7: タイムスタンプ予測の
閾値決定用データにおける学習曲線

各手法で 25 エポックまで学習し，その中で一番（閾値決定用データの）性能の良い重みを用いて，最終テストデータにおける micro F-score で評価する（表 5.1）．実験の結果から，曖昧ラベルと双向 LSTM を組み合わせて使った場合は，性能向上をもたらすことが分かった．

	Precision	Recall	F-score
C-LSTM	0.7806	**0.8545**	0.8159
+曖昧ラベル	0.8217	0.8381	0.8298
+双向 LSTM	0.7892	0.8440	0.8157
+曖昧ラベル+双向 LSTM	**0.8320**	0.8543	**0.8430**

表 5.1: テストデータにおける micro F-score の結果

第6章　Beatmap生成

本章では，図6.1に示すように，第一段階で選別されたタイムスタンプから，アクション種類を生成する第二段階について説明する．アクションのいくつかの特徴量を用いて，ニューラルネットワークを用いて予測を実現する．

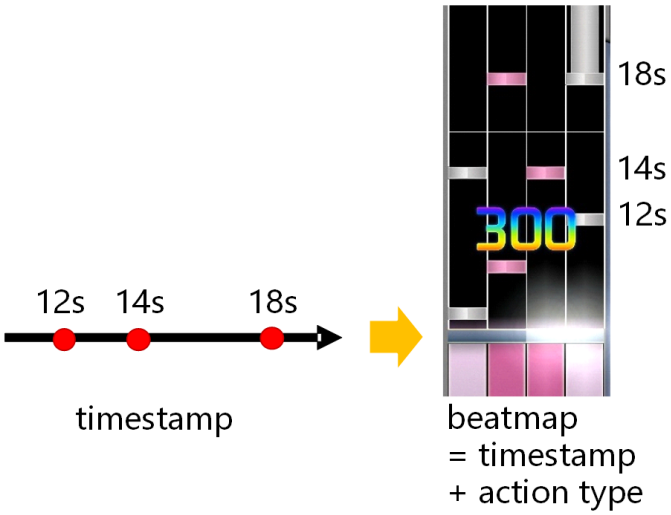


図 6.1: Beatmap 生成の概念図

6.1　アクション種類の予測

5 章で生成されたタイムスタンプは，タイミング情報を記録しており，アクションを置くべきタイミングである．そして，図 6.2 に各タイムスタンプに置くアクションの生成流れを説明する．

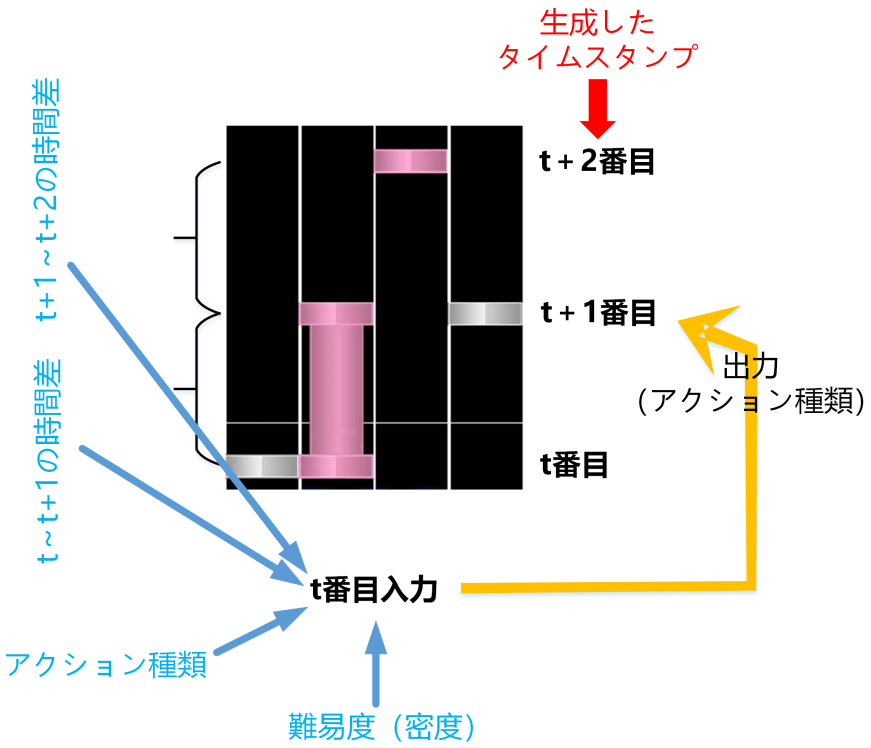


図 6.2: アクション種類生成の流れ

1. 5 章の手法でタイムスタンプを生成し，タイムスタンプ間の時間差を求める．
2. 生成するアクションの難易度を指定する．
3. 最初の 1 番目のアクションは 0 番目の入力（アクションなし，Beatmap に入れない仮想アクション）で生成する．
4. t 番目のタイムスタンプまでアクションが定まっており，t+1 番目のタイムスタンプのアクションをディープラーニングによって定める．
5. このとき，入力として，(難易度，t 番目と t+1 番目の時間差，t+1 番目と t+2 番目の時間差，t 番目のアクション種類）を用いる．出力されるのは t+1 番目のアクション種類である．

6.2　入力特徴量

本研究では，あるタイムスタンプに対応するアクションを出力するために,「一つ前のアクション」「指定難易度」「前／後のタイムスタンプとの間隔」を入力とする（図 6.3）．このとき，アクションは 4 ボタン分の,「アクションなし・クリック・長押し開始・長押し終わり」によって構成されるので，4 bit の one-hot ベクトルが 4 つある，16bit のベクトルとなる．これを“アクションコーディング”と呼ぶことにする．

そして，時間差とは，隣接のアクションの時間上の距離である．8bit の one-hot ベクトルで表記し，順で～50ms，～100ms，～200ms，～400ms，～800ms，～1600ms，～3200ms，3200ms～の 8 つの区間を意味する．次のアクションとの距離と，次のアクションともう一つ次のアクションの距離，2 つの時間差を用いて，総計 16bit のベクトルになる．

論文 [5] に用いたダンスゲームが最大両足のダブルクリックであることと違って,「OSU!」では，トリプルや 4 つ押しのアクションが高難易度でよくある．そして，難易度によってアクション組み合わせも変わる．本研究では，それらを区別するために難易度（密度）として 10bit の one-hot ベクトルを用いる．

以上，学習特徴量は，全部 42bit で，1 が 7 つ立つようなベクトルとなっている．

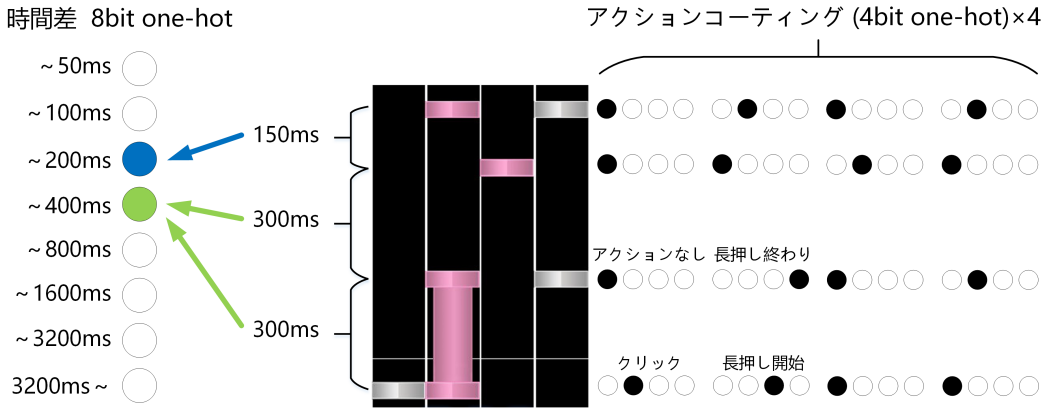


図 6.3: アクション種類予測の入力例

6.3　アクション種類予測のディープラーニング

ニューラルネットワークのフレームワークは論文 [3] に述べられた LSTM64 モデルを利用した．入力層の次は 128 ユニットの LSTM 層を 2 層用いる．出力層は 256 ユニットの Softmax 層である．各列で 4 通りの種類があるので，全部で 4^4=256 種類の可能アクションがあることになる．LSTM 層の活性化関数は tanh を利用する．図 6.4 にフレームワークを示す．

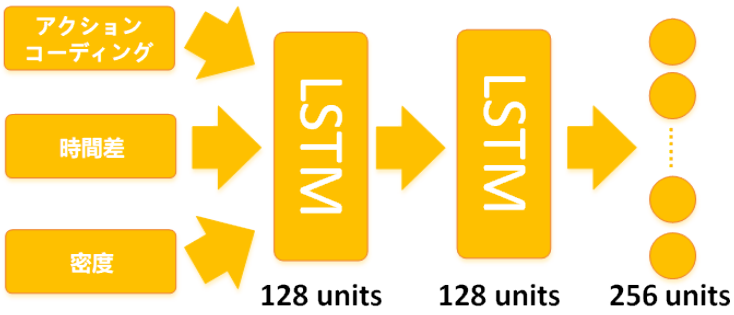


図 6.4: アクション種類予測のフレームワーク

図 6.5 に，タイムスタンプ t 番目の入力から t+1 のアクション種類を生成する例を示す．

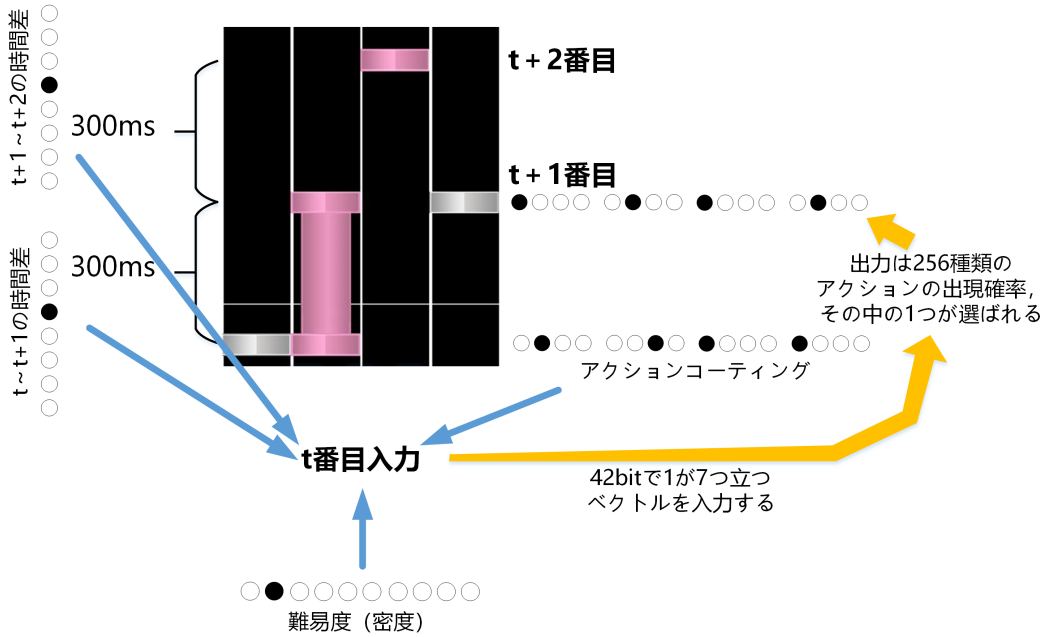


図 6.5: アクション種類の予測例

6.4　アクション種類予測の実験条件及び結果

アクション種類予測のディープラーニングの学習にあたり，データセットを 5 章の実験と同じ配分として行う．そして，難易度区分ごとに多くの Beatmap があるものと少ないものがあるが，学習の際にはこれらがほぼ同数となるように，最も多くの Beatmap がある難易度に合わせ，足りない難易度区分では，そのデータをコピー（焼き増し）することで学習させることにした（図 5.6 に参考）．

アクション種類予測では，バッチサイズは 128 にする．各 LSTM 層の出力に 50% のドロップアウトを用いる．正則化スケールは 0.0001 にする．

図 6.6 がアクション種類予測の正解率の推移を示したものである．最終的なテストデータの正確率は 0.4366 になった．実際には，不正解の中にも,「アクションの種類は合っているが場所が 1 つ違う」などの惜しいものもあり，この 43.66%という正解率はさほど小さいとは言えない．この結果は，出力の中に確率一番高いアクションタイプを選択した結果である．

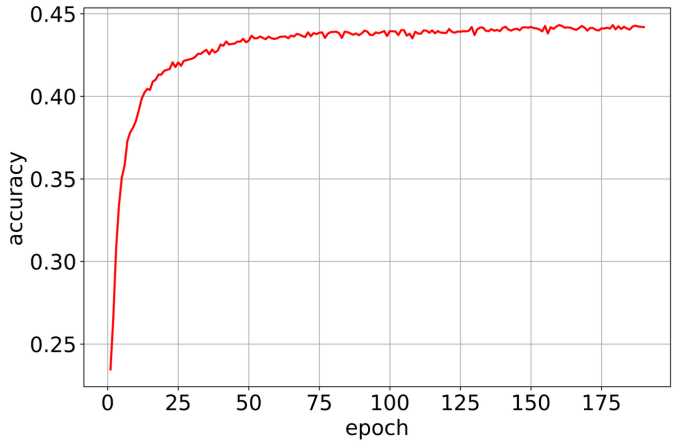


図 6.6: アクション種類予測の学習曲線

図 6.7 に，一つの楽曲について人間が手作りした Beatmap と，提案手法が予測した行動群を示す．かなりのところで違うアクションにはなっているが，遊ぶ際にはそれなりに似通ったものになっていると言える．

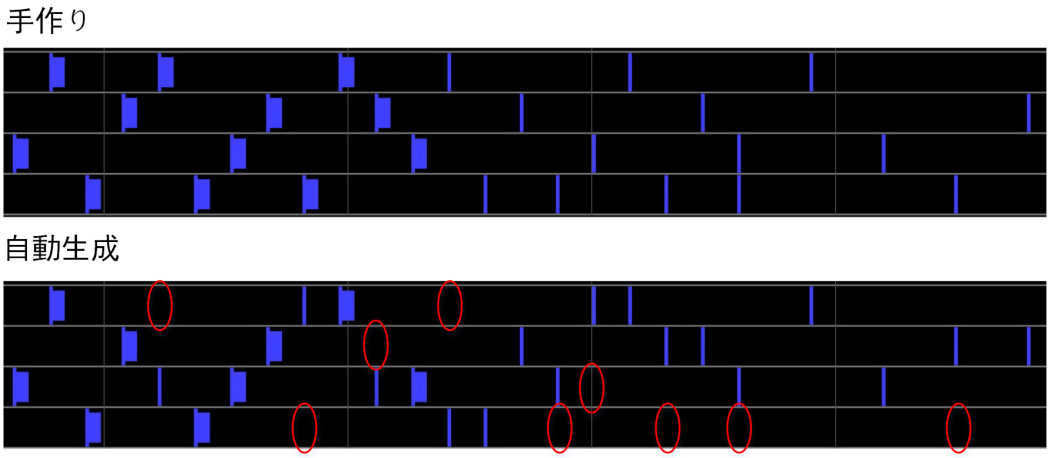


図 6.7: 手作り Beatmap と同じ音楽を用いた生成例
同じ入力で人間デザイナーと自動生成の違いを示す

図 6.8 に，一つの楽曲について異なる人間デザイナーが作成した Beatmap を示す．タイミングは同じ楽曲を使っているだけあって似通っているが，アクションはかなり異なっていることが分かる．これら同士の半分にも満たない一致率を考えれば，我々の一致率 43%は悪いものではない．

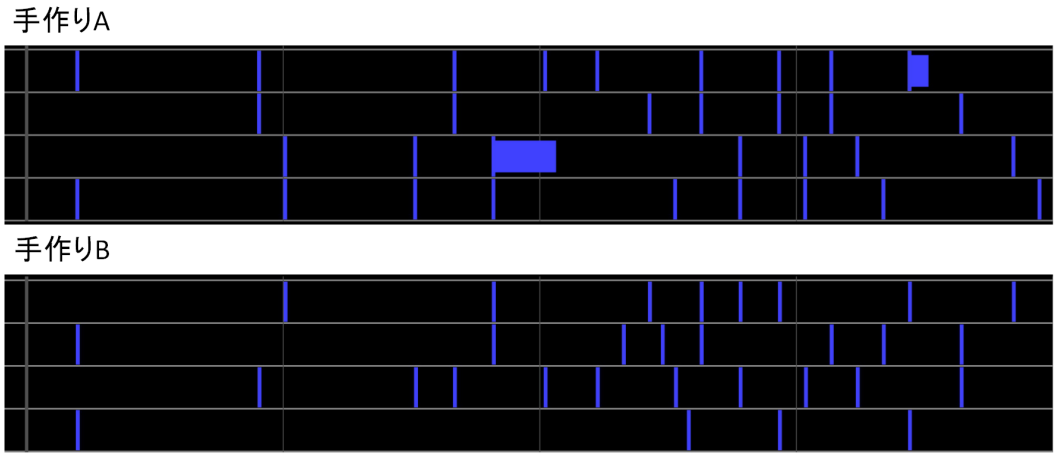


図 6.8: 同じ音楽を用いた手作り Beatmap の比較例

第7章　上達支援

従来のリズムゲームのコンテンツの「ゲーム上の難易度」は，曲単位で決められている.「プレイヤにとっての難易度」に影響する要素は様々だが，プレイヤが自分でそれらを細かく決めることはできない．例えば，簡単だと分類されるコンテンツの中に，あるプレイヤにとって不得手な部分が存在することがあったとする．その部分をプレイヤは繰り返し練習したいが，そのためには最初から遊ぶ必要があり，場合によっては他のほとんどの部分はそのプレイヤにはつまらないものになる．そういった不適切，不便なところに着目し，我々は「アクション組み合わせ」を自動調整する機能を実現する．

プレイヤの上達を支援するためには，プレイヤの総合的技量（得点）だけでなく,「どのような種類のミスなのか（遅すぎ・早すぎ・横ずれなど)」「どのような状況でミスしやすいのか（アクション数が多い・ボタン変化が多い・長押しが多いなど)」「具体的にどのようなアクションのパターンが苦手なのか」などを計測，分析，提示する必要がある．そのうえで，苦手部分が多く現れるような Beatmap の調整を行うのである．

上達支援の流れを図 7.1 に示す.「OSU!」からプレイログを取得し，分析を行って結果と苦手なところ説明し，そのうえで上達支援のため Beatmap を調整する．

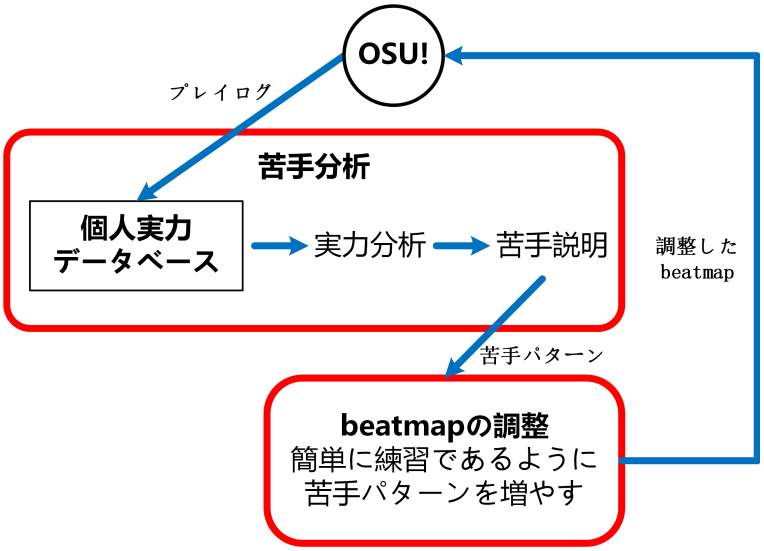


図 7.1: 上達支援の流れ概念図

7.1　プレイヤの技量評価尺度

本システムでは，プレイヤごとおよび 1 プレイごとに詳細なプレイログを記録する．プレイログには，何フレーム目にどのアクションを行ったかが記録されており，これと正解の Beatmap を突き合わせることにより，そのプレイヤの総合的技量やミスの種類を評価することができる．図 7.2 に，ログから抽出するプレイヤの技量評価尺度の一覧を示す．総合的技量については 7.1.1 節で，ミス種類については 7.1.2 節で述べる．

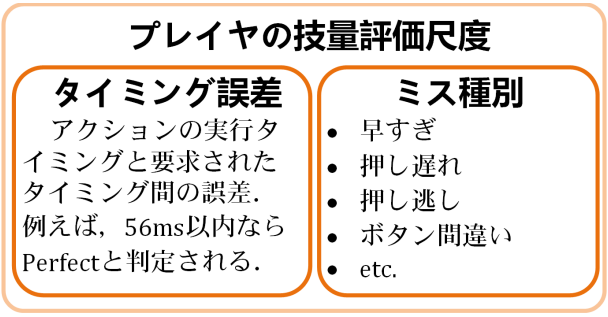


図 7.2: プレイヤの技量評価尺度

7.1.1　アクション実行のタイミング誤差

プレイログを分析し，要求されるタイミングと，実際にそのアクションが行われたタイミングを比較すれば，プレイヤ個人のタイミングの正確さが分かる．本研究において，そのタイミング誤差により，プレイヤの行なった個々のアクションを分類する（56ms 以内は Perfect，108ms 以内は Great，160ms 以内は Normal，160ms 以上は MISS とする．これらは「OSU!」で用いられるのと似た，ただし異なる，本研究独自に定めた基準である．「OSU!」は判定種類が多く，難易度によって判定の厳しさも違い，各デザイナーが自らの好みで設定しているため）．

```
Percentage of Perfect:  72.5%    Mean Timing Error:  23.87ms
Percentage of Great:    20.2%    Mean Timing Error:  75.32ms
Percentage of Normal:    4.7%    Mean Timing Error:  126.67ms
Percentage of MISS:      2.7%
Sum of not MISS:        97.3%    Mean Timing Error:  39.45ms
Level Coefficient:  108.27 (less is better)
Mean push down time:  155.37ms (less is better)
```

図 7.3: プレイログのタイミング誤差分析結果

図 7.3 に，プレイヤ一人の（おおよそ 30 分の）プレイログの分析結果を示す．図に各判定（Perfect など）のアクション割合及びタイミング誤差をプレイヤに示す．

Mean Timing Error とは，Perfect/Great/Normal それぞれの判定，およびそれら全体の中で，タイミング誤差の平均値を取ったものである．これが低いほど技量レベルが高いと言えるが，さらにこれに係数をかけて一元化したものとして Level Coefficient を用いる．

Level Coefficient とは，式 7.1 で計算した値である．これは実力の絶対値ではなく，実力の変化を表す相対値に用いる．この値を通して比較すれば，プレイヤが自分のアクションタイミングの正確さが改善したかどうかを簡単に分かるようになる．

$$\begin{aligned} Level\ Coefficient\ &=\ Perfect\ Percentage * Perfect\ Error * 1 \\ &+\ Great\ Percentage * Great\ Error * 2 \\ &+\ Normal\ Percentage * Normal\ Error * 3 \\ &+\ MISS\ Percentage * Miss\ Error * 4 \end{aligned} \tag{7.1}$$

Mean push down time とは，プレイヤが一つのボタンを押してから離すまで時間の経過であり，この値が大きいほどプレイヤがボタンを離す対応が追いつけなく（クリックに離す判定がないため疎かにしがちであり），小さいほど思い切りよく押して離したと考える．優秀なプレイヤならば 80ms から 100ms 程度に収まるため，自分の傾向を知ることで修正を試みることができる．

7.1.2　間違いの種類（ミス種別）

プレイヤはその技量ごとに多少の MISS 判定を受けるが，同じような総合力のプレイヤであっても，その苦手部分はさまざまに異なるし，ミスの種類も異なる．本研究で提案するシステムでは，ミスの種類を 7 種類に分類し，どのようなミスが多いのかを気付けるようにした．

図 7.4 が，ミスの場所や程度（左側）と種類（右側）を表したものである．左側の長い青の横棒は要求されたアクションである．短い横棒が実際にプレイされたアクションであり，黄色は Perfect, 緑色は Great，水色 Normal，赤色は MISS を表している．短い横棒の縦幅はボタンの押されている長さである．

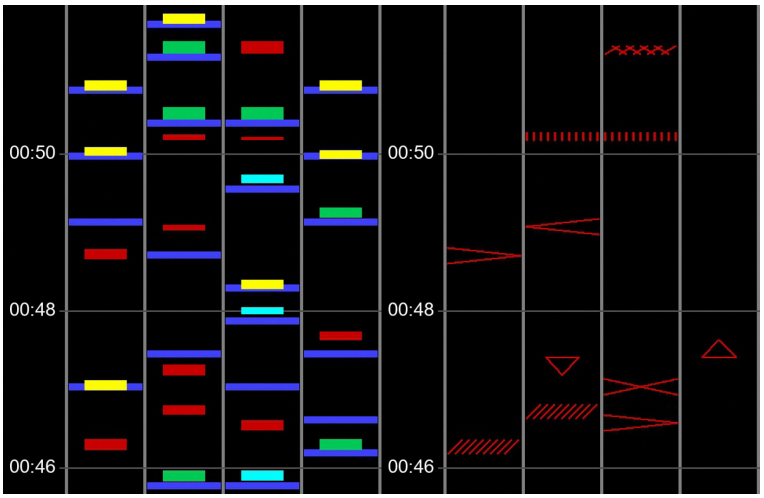


図 7.4: 各間違い種類の例

図 7.4 の右側のマークは，そのタイミングその場所であった MISS に対して，どのような種類なのかを表したものである．

×	押し逃し，ボタンを押しそびれた
△	押し遅れ，遅れてボタンを押した
▽	押し急ぎ，要求されるより早くボタンを押した
＞や＜	ボタンミス，違ったボタンを押した
//////	空押し，アクションのないところに押した
～～～	隣接ボタンの同時押しミス，一つに 2 つのボタン一緒に押した
|||||||||	ボタンの 2 度押し，アクション一つにボタンを早く 2 回押した

これらの種類を用い，同じ間違いでもそのプレイヤの個性や癖が分かる．

7.2 Beatmapのグルーピング

本研究では，前節で述べたプレイヤのミスの分類のほかに，プレイヤがどんな状況を苦手としているか，またどんなアクション配置パターンを苦手としているかを抽出して提示し，さらに複製したい．Beatmapおよびその部分であるパターンは多様であり，全く同じ状況やパターンは何度も現れることは少ないため，似た状況や似たパターンをグルーピングする必要がある．

本節では，そのグルーピングをクラスタリングによって行う．クラスタリングには距離の定義が必要であり，7.2.1節では“状況”の定義と距離の計算法，7.2.2節では“パターン”の定義と距離の計算法を提示する．得られたクラスターの例は7.3節で後述する．

なお，グルーピングの質を保証するために，本節のクラスタリングは予め本研究のデータセットのBeatmapを用いて行い，全プレイヤに対して同じクラスターを用いる．Beatmapのサンプルの数が多いほど良いクラスターが得られる利点がある．一方でこうすると，ある（プレイログの少ない）プレイヤだけについて見れば,「20のクラスターのうち5個程度しか実際には出会っていない」ということもありえて，きめ細かい指導がしにくくなるという課題もある．

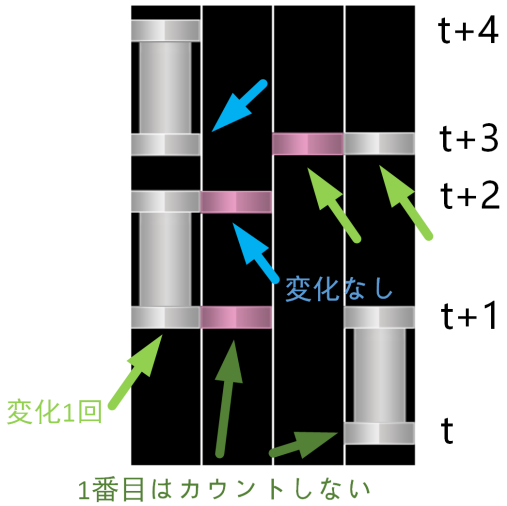
7.2.1 Beatmapスライスを用いた状況の定義

本節では，各プレイヤがどのような状況でミスをしやすいのかを抽出するため，そもそも状況とは何で，どんなものに影響されるのかを定義する．これは例えば，Beatmapの中のある数秒について「アクションは単純だが回数が多い」「クリックと長押しが組み合わされ複雑」「同時押しが多くかつそれが繰り返される」などの特徴を表したものにしたい．そうすれば，プレイヤは自分がどのような状況が苦手なのか分かり，また学習システムは苦手なパートを増やして練習を容易にすることができる．

我々はBeatmapを1〜3秒にランダムにスライスし，以下の16個の特徴量を用いてそれを定量化する．この特徴量は予備実験を通じて決めたものである．Beatmapを同じ長さにスライスしないのは，楽曲によって適切なスライスの長さが異なるためであり，ランダムに何度もスライスすることでその影響を抑えてある．以下の特徴量の中で「時間単位数」とあるのは，Beatmapスライス長を100msで割った値であり，10〜30の整数である．

1. アクション密度： アクション数 / 時間単位数
2. タイムスタンプ密度： タイムスタンプ数 / 時間単位数
3. クリック平均間隔： クリックのあるタイムスタンプ間の時間単位数の平均値
4. クリック間隔の標準偏差

5. クリックの割合： クリック数 / アクション数
6. 長押しの割合： 長押し数 * 2 / アクション数 (長押し 1 回は「押す」と「離す」2 アクションある)
7. 長押しの平均長さ（時間単位長さ）
8. 長押しの長さの標準偏差
9. 長押しの変化率： 変化回数（図 7.5） / 長押し数
10. 長押しとクリックの比例： 長押し数 / クリック数
11. クリックの変化率： 変化回数（図 7.5） / クリック数
12. シングルクリック割合： シングルクリック数 / タイムスタンプ数
13. ダブルクリック割合： ダブルクリック数 / タイムスタンプ数
14. トリプルクリック割合： トリプルクリック数 / タイムスタンプ数
15. 4 つ押しクリック割合： 4 つ押しクリック数 / タイムスタンプ数
16. 長押し・クリック割合： 長押しとクリックが一緒のタイムスタンプ数 / タイムスタンプ数



タイムスタンプを単位に，t+1のボタンがtのボタンと違う場合，ボタン一つが1変化としてカウントする．長押しとクリックは別々カウントする．この場合，長押し変化回数が合計1回，クリックの変化回数が合計2回である．

図 7.5: ボタンの変化回数説明図

この 16 次元の特徴量ベクトルとそのユークリッド距離を用いて，20 カーネルの K-means クラスタリングを行うことでグルーピングを行う．なお，ユークリッド距離を用いているため，各特徴量のうち“本来とても重要であるべきもの”“本来無視すべきもの”が過小評価・過大評価されている恐れはあり，それは今後の課題である．

さらに，我々はこのクラスタリングののち，各カーネルについてそれを特徴づけている“重要特徴量”を 3 つ抜き出す．これにより，特定のグループが苦手なプレイヤに対して,「あなたはクリックと長押しが組み合わさった状況が苦手ですね」などと理解の容易な説明を行うことができる．

7.2.2　アクションパターン

前節の Beatmap スライスが総合的な“状況”を表そうとしていたのに対し，本節では，具体的な指の運びに直接関連する“アクションパターン”に注目する．そのために，Beatmap を 3 ステップ× 4 ボタンの単位で見ることにする（図 7.6）．あまりに時間的に離れたものには意味がないため，1 秒以下の範囲の 3 ステップについて注目する．

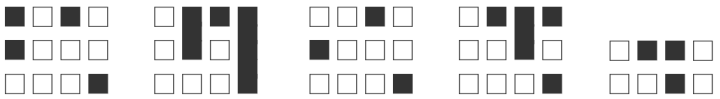


図 7.6: アクションパターン例

ここでは求められるタイミングの情報は削除され，どのボタンを押す必要があるかだけが示されている．アクションパターンは全体ではなく，局所からそのアクション組み合わせで Beatmap の具体的な「形」を測る．アクションパターンを用い，プレイヤの苦手パターンを見つけ，記録する．

アクションパターンは非常に多数あるため，前節の Beatmap 状況のグルーピングと同様，20 カーネルの K-means 法でクラスタリングを行う．アクションパターン間の類似度を表すための距離は以下のように定義したが，あまり似ていないものが同じグループに入ったり，その逆など，これもまだ検討の余地はある．

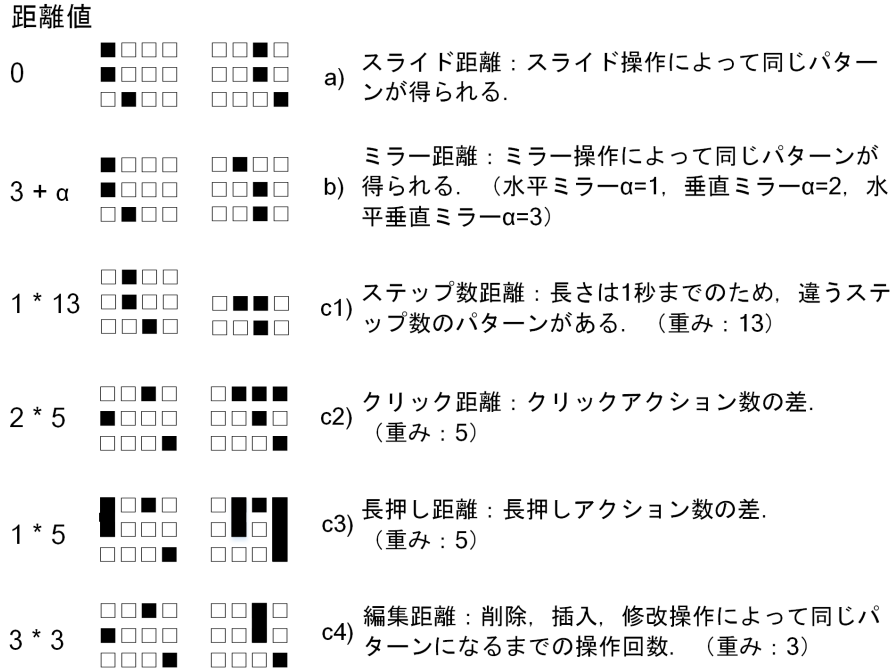


図 7.7: アクションパターンの距離を求める方法

アクションパターン間の距離の計算は以下のように 3 種類に分けて行う．各距離の計算方法は図 7.7 に例示する．

a スライド操作だけで変換できるパターン間の距離は 0 とする．

b ミラー操作だけで変換できるパターン間の距離は $3+\alpha$ とする．

c そうでない場合，以下の 4 つの距離の合計+6 を，パターン間の距離とする．

c1 ステップ数距離．ステップ数が異なる場合は，差× 13 を加算する．

c2 クリック数距離．クリックアクション数の差× 5 を加算する．

c3 長押し数距離．同様．

c4 編集距離．削除，挿入，修改を何回行えばパターンを一致させられるか × 3 を加算する．

7.3　プレイログ分析及び苦手説明

プレイヤ個人のプレイログがある程度溜まったら，統計分析を行う．タイミング誤差の分析，間違い種類の分析，苦手な Beatmap 状況（Beatmap スライスグループ）の分析，苦手なアクションパターン種類の分析である．そして，分析結果をプレイヤに説明する．本研究は，分析にあたるログ量は 30 分以上必要と考える．

苦手状況や苦手パターンを提示するためには，20 のクラスターに属する状況やパターンがそれぞれ何回登場し，そのうち何回プレイヤがミスするのかをカウントする必要がある．そして，単純にミス回数ではなくて，ミスの割合に注目する．例えば 100 回登場する状況に対して 10 回のミス（0.1）と，200 回登場する状況に対して 15 回のミス（0.075）であれば，前者の状況のほうが苦手であると判断する．ただし，あまりにも稀な状況（出現回数 50 以下）は指摘することに価値が少ないため，例えば 10 回登場する状況に対して 3 回のミスであれば，単純なミス率 0.3 に「出現回数/50」の重みをかけて 0.06 と補正して優先度を下げた．

続いて，4 つの項目について何をどのように説明するかを述べる．

- 【総合的な技量について】　図 7.3 に，総合的技量の提示例を示す．ここでは，プレイヤは自分がどの程度のミスをしたのか，MISS と判定されなくともどの程度求められたタイミングでアクションを取れていたのかなどを知ることができる．これらから直接的に改善の方針を見つけることは難しいが，Level Coefficient などは総合的な技量を示しているため，練習を続けることでこれらの数値が改善していくことはプレイヤの励みになる．

- 【ミス種別について】　7.1.2 節図 7.4 に，ミス種別を個別に示した提示例を示す．これは具体的に自分がどの部分でどのようなミスをしたのかを知るには役立つが，それだけではどんなミスが多いかを明確に知るのは難しいので，これとは別に他のプレイヤを含めた全体のミス種別の割合を示す（図 7.8）．これにより，例えば自分は隣接ボタンの同時押しが普通の人よりも多いな，といったことを知ることができる．

```
Amount of each mistake type:
              amount  percentage <-> players' mean percentage
leaky click   :  8        (  2.6% <-> 21.1% ) 押し逃し
time late     :  50       ( 16.8% <-> 9.3%  ) 押し遅れ
time early    :  22       (  7.3% <-> 8.0%  ) 押し急ぎ
empty click   :  29       (  9.6% <-> 8.3%  ) 空押し
sticky click :  107      ( 35.3% <-> 9.3%  ) 隣接ボタンの同時押しミス
wrong key     :  49       ( 16.5% <-> 19.0% ) ボタンミス
double key    :  35       ( 11.9% <-> 25.0% ) ボタンの 2 度押し
```

図 7.8: ミス種別分析結果の出力例

- 【苦手状況について】 20 のクラスター全てについて登場回数とミス回数を提示するのでは，プレイヤにとって情報が多くなりすぎる．従って，補正後のミス割合によってソートして，その上位 3 つだけを提示することにする（図 7.9 上側，3 つのうち 2 つのみ例として表示）．各状況（Beatmap スライス群）の説明については，7.2.1 節の最後で説明したように，カーネルの支配的な特徴量の上位 3 つを示す．最後に，苦手な状況に共通する特徴量を示すことで，全体的にどんな状況が苦手なのかを提示する（図 7.9 下側）．

```
Beatmap cluster No.1
Characteristic:
|---percent of double click
|---percent of quadruple click
|---percent of clicks    クリックの割合
Miss rate(adjusted):0.144 Total occurrences:1255 Total miss:181
================================================================
Beatmap cluster No.16
Characteristic:
|---std-length of press
|---percent of presses
|---rate of presses and click    長押しとクリックの比例
Miss rate(adjusted):0.089 Total occurrences:1563 Total miss:139
================================================================
                              ......
Frequent characteristic of mistake Beatmap attribute:
|---confusion of press    長押しの変化率
|---std-length of press
|---percent of press-click    長押し-クリック割合
|---percent of single click
|---rate of presses and click    長押しとクリックの比例
```

図 7.9: 苦手な Beatmap 状況の出力例

- 【苦手パターンについて】 苦手状況と同様，20 のクラスター全てについて提示するのは適切ではないため，補正後のミス割合によってソートして，上位 3 つだけを提示する．各パターングループから，実際に登場した典型的な 5 つのサンプルを示す（図 7.10）．

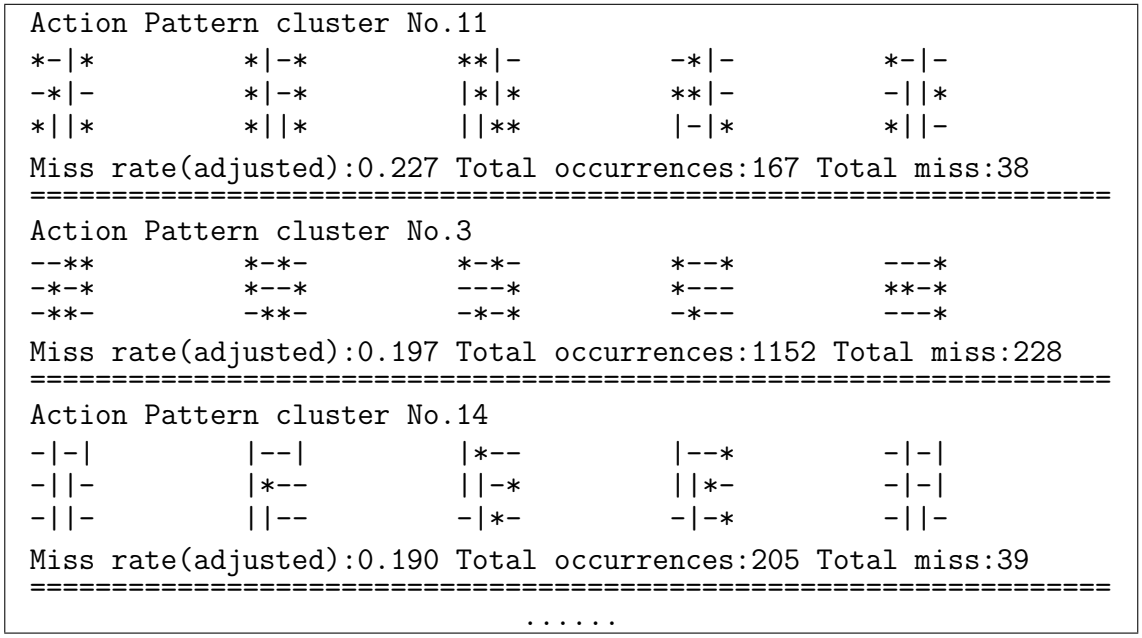


図 7.10: 苦手なアクションパターンの出力例

さて，ここで一つ考えるべきことがある．登場した状況やパターンそれぞれについて，ミス割合によってソートして上位のものを提示することが適切かどうかということである．同じ難易度の Beatmap 群の中にも，また 1 曲の Beatmap の中にも，比較的簡単なところと比較的難しいところというのは当然存在する．ということは，状況グループやパターングループの中にも，比較的簡単なものや難しいものがあってもおかしくない．具体的には，

- A グループ： 比較的簡単なはず．平均的プレイヤは 2%くらいミスをする．このプレイヤは 7%のミスをした．
- B グループ： 比較的難しいはず．平均的プレイヤは 10%くらいミスをする．このプレイヤは 8%のミスをした．

というような二つのグループがあるとする．このとき，絶対的な視点からは B グループがより苦手と判定されるが，総体的な視点からは A グループのほうが苦手とすべきであろう．本研究ではこの点については比較を保留して補正ミス割合でソートすることにしたが，これも今後の検討課題である．

7.4　上達支援ためのBeatmap調整

プレイを繰り返していれば，どのアクションパターンが得意で，どのアクションパターンは（偶然ではなくて）苦手としているのかが分かるようになる．プレイログに基づき，苦手なアクションパターン種類を特定し，3 章図 3.3 で提示したようにそのパターンやそれに似たパターンを追加挿入することは上達を支援するために有効であると考える．

本来は，Beatmap 生成・アクション選択そのものと組み合わせて曲調にあわせ苦手パターンを多めに生成するモデルが理想である．本論文ではそこまで高度なことはできなかったが，苦手パターンを増やすシステムを提案し，評価した．

7.4.1　間違い確率の予測

Beatmap を調整する前に，まずプレイヤにとって Beatmap の簡単なところ及び苦手なところを予測する必要がある．簡単なところを苦手なパターンを入れ換えるためである．

7.2.1 節では Beatmap スライスを用いて苦手な状況を，7.2.2 節では 3 ステップのアクションパターンを用いて苦手なパターンを抽出する方法を示した．このとき，各状況についての間違い率を計測しているから，その簡単な部分を難しい部分に置き換えるのが自然な考え方である．我々は今回，タイムスタンプそのものは変更せずに，アクションの種類のみを変更するアプローチをとる．タイムスタンプそのものまで変えてしまって Beatmap スライスの入れ換えを行うと，曲調を損なう可能性が高いと考えたためである．

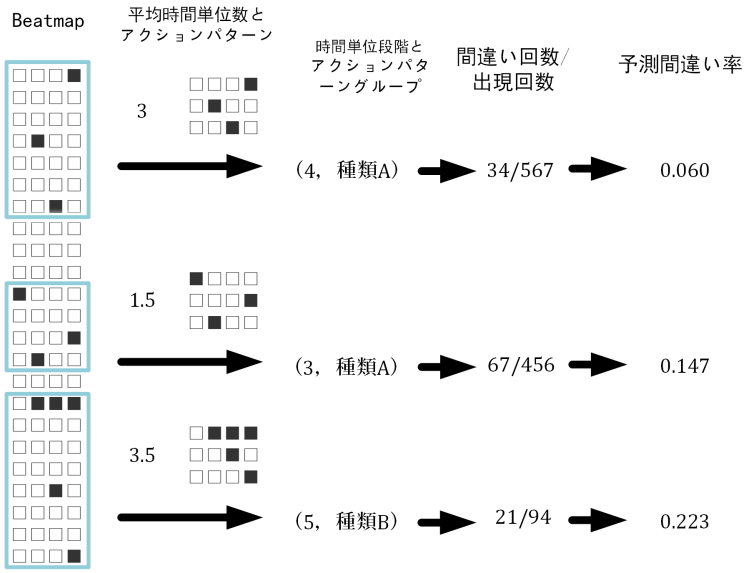


図 7.11: 間違い確率の予測例

一方で，アクションパターンのみを見ていると，それが「離れた簡単なもの（例は図 7.11 左上）」なのか,「詰まった難しいもの（例は図 7.11 左中）」なのか分かりにくい．そこで，アクション間の平均間隔を考慮に入れ,「アクションパターンのグループ 20 種類」と「平均間隔の長短 6 種類+1 ステップ間隔なし」の組み合わせにより，それらについてどの程度そのプレイヤが間違っているのかを調べることにする．

まずは，時間単位数を {1, (1,2], (2,3], (3,4], (4, 5], (5, ∞), 0(1 ステップの場合)} の 7 段階に分けてタイミング情報とし，7.2.2 節に予めグルーピングしたアクションパターンのクラスターと合わせ，140 セルのテーブルを作る．次に，Beatmap を切り分けてその出現回数と間違い回数を記録する．最後は，間違い率を計算し，テーブルを埋める．図 7.12 に例を示す．

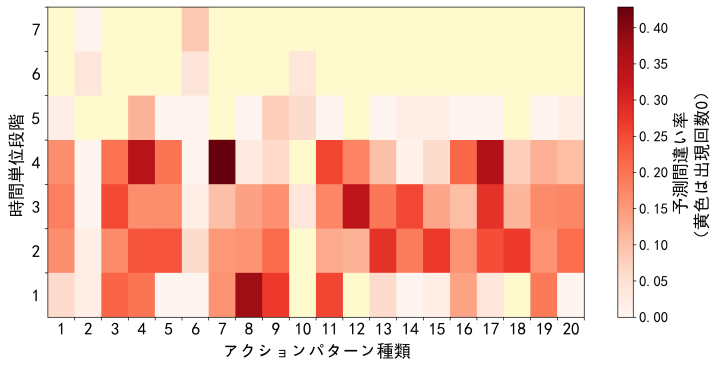


図 7.12: あるプレイヤの 30 分のログで作成した実力テーブル

図 7.12 に例を示す．例えばアクションパターン 8 を見ると，間隔が短いほどミス率が高く，これは納得できる結果である．一方でアクションパターン 7 を見ると，間隔が長いところに多くの間違いがあり，まだ原因を特定することができないが，プレイ時間が少ないためのラッキーアンラッキーがある可能性はある．

7.4.2　苦手パターンの挿入

プレイヤ個人に対する難易度（間違い率）を求めた後，苦手パターンの挿入が行なわれる．本研究においての苦手パターンの挿入は，アクション種類だけを入れ換えて実現する．タイムスタンプは変更しない．なお，現段階では，挿入の効果を反映しやすくするため，挿入する苦手パターンはプレイヤのプレイログに出現したものだけとする．

次は苦手パターンの挿入の手順を述べる．

1. Beatmap の平均間違い率を計算する．
2. 平均間違い率より低い Beatmap 区間を簡単アクションパターンとし，そのタイムスタンプを記録する．
3. 実力テーブルから，1.5 倍の平均間違い率より高いセルのアクションパターンを候補苦手アクションパターンとする．(従って，全体が難しいほど，候補アクションパターンは少なくなる)
4. ランダムで苦手パターンを選出し，2. で選ばれた簡単パターンの部分に入れ換える．もしこの時間間隔について対応する苦手パターンの候補がない場合は変更しない．

なお，簡単なパターンをどの程度苦手パターンに置き換えるかは自由であるが，後述の実験では，置き換える対象が見つからない場合を除いて全て置き換えている．

さらに，長押しアクションが関係する場合には追加的なルールを採用する．付録 A.2 では，音楽から主旋律を抽出し，同音が長く続く場所“長押し区間”を定める方法を提案している．そこでそれを用いて，

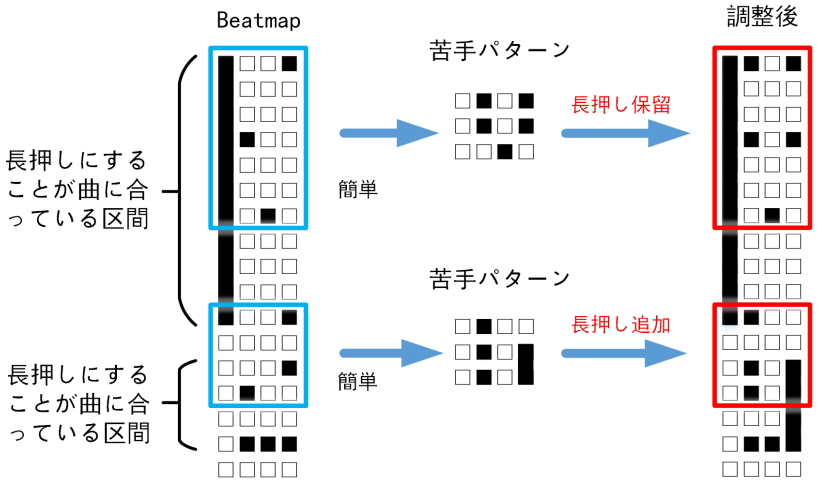


図 7.13: 苦手パターンの挿入ルールの説明図

- 置き換えたい（削除したい）簡単区間が長押し区間であり，実際に長押しが採用されているならば，そこ以外を置き換える（図 7.13 上）．
- 置き換えたい（追加したい）苦手パターンに長押しが含まれている場合で，それが長押し区間にかかっているならば，置き換え範囲を越えて長押しアクションを追加する（図 7.13 下）

という工夫を行う．

苦手パターンの挿入にあたり，候補アクションパターンと現在のアクションパターンの類似度を考慮していないので，調整した Beatmap は不自然になったり，急に種類が変わったようにプレイヤに違和感を感じさせたりする可能性がある．そして，タイムスタンプも変更しないため，変更できる幅が小さい．これらは，今後に改善すべき課題とする．

図 7.14 では，あるプレイヤの苦手データを用いた調整例である．

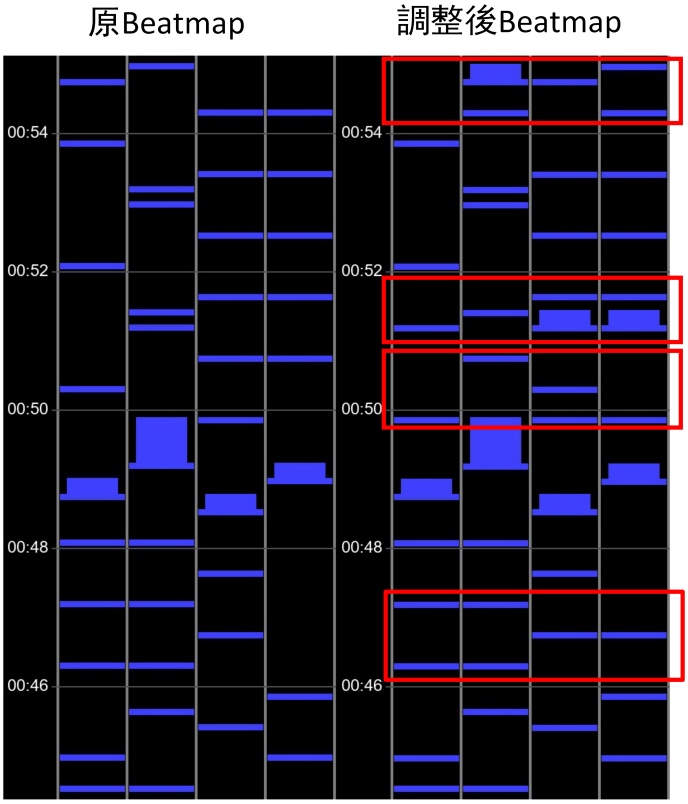


図 7.14: 実際の調整例
苦手なアクションパターンの挿入は確認された

7.5　上達支援機能についてのアンケート調査

本来ならば，このような教育システムの評価では，事前に技量が似通った 2 つ以上のグループを準備し，1 つのグループには提案手法による訓練，別のグループには既存手法（例えば通常の Beatmap を繰り返すだけなど）をさせて技量の伸びを比較するのが望ましい．しかし本研究では，多数の被験者を集めることの困難さと時間の関係でそれを諦め，数人の被験者によるアンケート調査に留めた．

【実力分析の有効性】

まず，3 人の被験者に，それぞれ 40 分（20 Beatmap 前後）ほど,「OSU!」のプレイを行ってもらった．そしてそのプレイログを集め，7.3 節で述べた総合技量・ミス種別・苦手状況・苦手パターンについての指摘を行った．この結果を被験者に提示したうえで，以下の選択肢（図 7.15）から感想を選んでもらった．

1. 実力分析は適切でしたか.
    1a 適切
    1b 不適切
2. 苦手なポイントが当てられましたか.
    2a 結構当てられた
    2b 大体当てられた
    2c ほとんど当たっていない

図 7.15: アンケートの一部 1/3

結果として，全員が (1a) (2a) を選び，実力分析について肯定的な結果が得られた．

【苦手パターン追加の有効性】

続いて，7.4 節の技術を用いて，苦手なアクションパターンを挿入したものを 15 分（6 Beatmap 程度）ほどプレイしてもらった．そのうえで，以下の選択肢（図 7.16）から感想を選んでもらった．

3. 自分の苦手パターンが増えたと感じましたか.
    3a 感じた
    3b 感じない
    3c 感じないけど難しくなったと感じる
4. 苦手パターンの入れ方に不自然と感じましたか.
    4a 感じない
    4b 感じた

図 7.16: アンケートの一部 2/3

結果として，全員が (3a) を選んだという意味では肯定的な結果であったが，同時に (4a) を選んだ被験者が 2 名，(4b) を選んだ被験者が 1 名おり，まだ改善の余地があると考える．

【総合評価】

最後に，このシステム全体の有用性を以下の選択肢（図 7.17）から 1 つ選んでもらった．

5. こんな上達支援機能は使いたいと思いましたか．
   5a 有用なので使いたい
   5b 面白いので使いたい
   5c 別に使わなくても大丈夫

図 7.17: アンケートの一部 3/3

結果として，2 人が (5a) を，1 人が (5b) を選び，(5c) は選ばれなかった．

これとは別に自由記述をお願いしたところ,「苦手パターンを入れるときに，全体のバランス（時に簡単，時に困難という必要な段取り）が少し良くない」「苦手なパターンは増えたが，自分がほしい苦手パターンがこない」など，我々も不十分と考えている部分のコメントをもらった．

以上の結果から，上達支援機能は概ねうまく作動したといえる.「遊んでもらって」「ログを分析して」「苦手なところを見つけてうまく説明して」「苦手なところを効率よく練習させる」という基本的な流れの基礎部分を実現することはできた．しかし，研究として見た際にそれぞれの提案や設定がどの程度結果に結びついているのかは不明瞭であり，また十分でないことが分かっていても行えていない部分があり，それは今後の課題である．

第8章　結論及び今後の課題

我々は，ディープラーニングを主な手法として，音楽素材からコンテンツを自動的に生成することを目的として研究を行った．既存研究で用いていたゲームよりも利用の便利な「OSU!」を用い，その際に現れた課題に対処した．難易度がさまざま・正例が少ないといった学習データの特性に対し，曖昧ラベルを提案し，その有用性を実験で実証した．双向 LSTM による性能の向上も実証した．

そのうえで，アクション生成機能・上達支援機能を作成し,「OSU!」と統合することでプレイヤが「音楽を入れ」「Beatmap を遊び」「実力が分析され」「苦手を練習できる」統合システムを作った．特に実力を分析する部分は重要であり,「プレイヤが行ったアクションがどれくらい上手なのか」「プレイヤがどんな間違い方をするのか」「どんな状況が苦手なのか」「どんなアクションパターンが苦手なのか」,それぞれの評価尺度を定めて提示することに成功した．被験者実験の結果からこの部分の成果は肯定的だと考える．

さらに，コンテンツの中にプレイヤ個人にとって簡単なアクションパターンと苦手なアクションパターンがあることを踏まえ，これを検出し，簡単な部分を苦手なパターンで置き換えることを提案，実装した．現段階ではまだ自然さが不十分であるが，苦手な部分が増えたという評価は得られた．

全体的に見ると，まだ足りない点が様々残っている．例えば，現時点のアクション種類生成は，音楽の特徴を入力特徴量としていない．上達支援では，苦手パターンの挿入機能だけ実装しており,「プレイヤの実力に合わせて少しずつ難易度を上げる」などの機能がまだである．これらを解決し，またできるだけ楽しく有効に訓練するための支援技術の開発実装もしていきたい．

付 録A章　長押しに関する改良の試み

6 章では，Beatmap 生成の後半，タイムスタンプからアクション種類の生成を説明した．だが，その後半では，音の高さは入力に含めていない．これは明らかに不自然な設定であり，多くのゲームコンテンツでは音の高さも考慮してデザイナーが作成していると考えられる．

A.1　既存手法の問題点と解決策

タイムスタンプは音の始まりや変化タイミングを表しており，ドラムの打撃やボーカルの歌声などを混ぜ合わしたプレイヤが聞こえる“ リズム ”と言える．だが，音がするタイミングだけで音メロディ情報なしでは，単なるリズムであり，メロディとは言えない．例えば，長い音の始まりと終わりのタイミングがともにタイムスタンプに選ばれたとしても，後半の Beatmap 生成ではどれが始まりかどれが終わりか，あるいは単なる短い音の始まりか，それらの判別はできないということである．ゆえに，長い音に合わせて適切な長押しを生成することはできない．

論文 [5] に提出された音の高さをアクション種類生成の入力とする手法は，現在のアクションタイミングにおける音の高さを特徴量として入力している．それである程度メロディの変化を分かるようになった．だが，前後のタイムスタンプの音の高さが同じだとしても，その間にどれほどの音の高さ変化があったか，本当に長い音なのかを判別するに不十分である．本研究では，それを踏まえ論文 [7] の手法を使ってメロディ抽出をし，自然な長押しアクションを生成できる手法を実現し，検証する．(図 A.1 に参考)

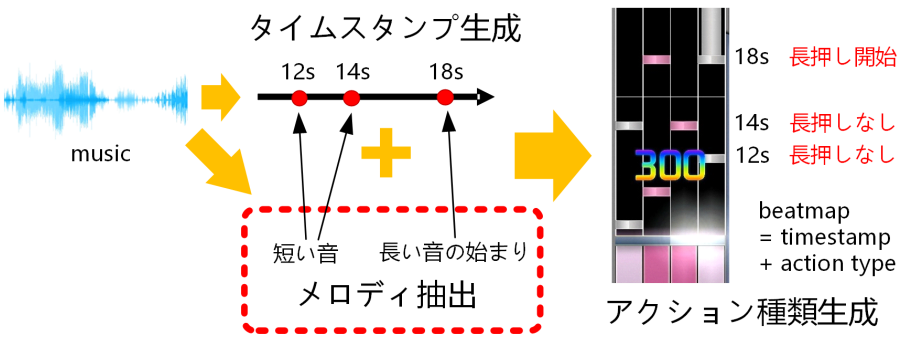


図 A.1: メロディ情報を用いた Beatmap 生成

A.2　メロディ抽出

予測されたタイムスタンプは長押しすべき始まりと終わりタイミングも含めているが，アクション予測の段階にその情報が欠落した．本研究は，それらタイムスタンプの種類（長押しすべきタイミングであるかどうか）を別途で求める．

基本的な流れは，長い音を検出して，長押しにすべき長い音の始まりと終わりのタイミングに，長押しのアクションを入れる．長押し予測の流れは図 A.2 に示す（説明は上から下へ対応している）．

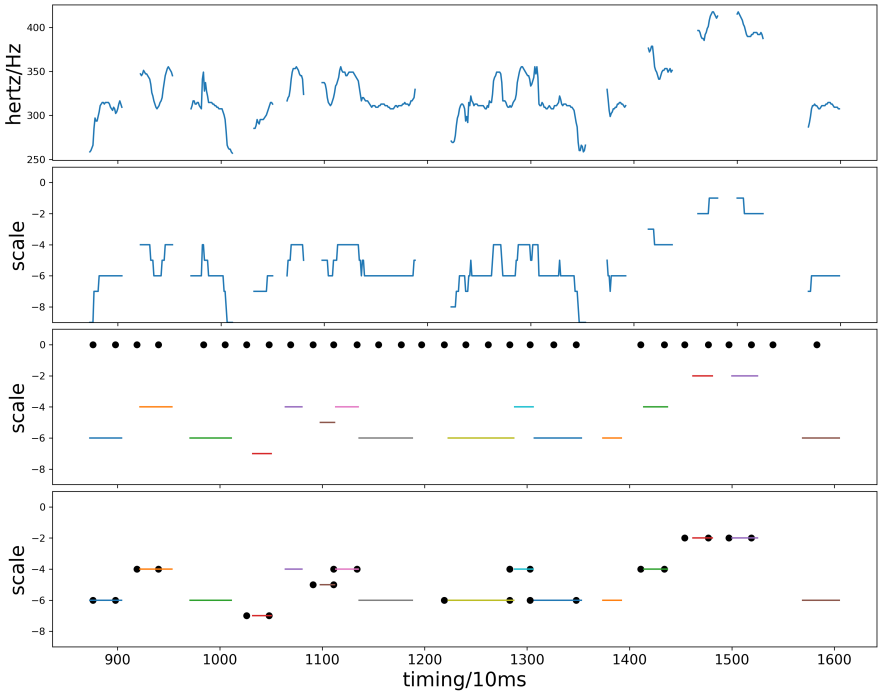


図 A.2: 長押し予測の流れ

1) 論文 [7] の手法を使って，主メロディを抽出する．横軸は時間フレーム（10ms 単位），縦軸はその時刻の主メロディの周波数である．この時間フレームは 5 章のタイムスタンプ生成に用いる時間フレームと一致している．

2) 周波数は音の高さを表しているが，人間の聴覚では，1Hz の差まで聞き分けることができない，必要もない．従って，得た主メロディを式 A.1 を用いて人間が馴染む音楽の音階に変換する．横軸は同じ時間フレームである．縦軸は音階であり，値を四捨五入した整数である．（式は 440Hz を基準音として，12 平均律を用い，音階を計算している．例えば，440Hz 周波数の音階は 0）

$$Scale = 12 * \log_2 \frac{hertz}{440}. \tag{A.1}$$

3) 得た時間順の音階列をスムーズし，予測誤差による急な音階変化をなくす.（例えば前後の音階は-3 であって，中に 30ms の間だけが-2 であった場合は，全部-3 にスムーズされる）．そして本研究では，60ms 長さ以上音を“長い音”とし，ほかの区間は放棄される（それは「長押しの最短長さは 60ms ぐらい」と本研究のデータセットを統計した結果である）.

4) そして,「長い音」の選出は 5 章のタイムスタンプ生成結果に従い，長い音の始まりと終わりがともにタイムスタンプ（黒点）と当てはまるものだけが選ばれる．本研究では当てはまる時の誤差許容範囲は 50ms としており，当てはまったタイムスタンプはつまり「長押しにすることが曲に合っている区間」である.

抽出された「長押しにすることが曲に合っている区間」（長押し候補区間）は，Beatmap 調整するときにも用いられる.

A.3　長押し数の割合確率モデル

得た長押し候補区間は，全部使うのではない．それは，音楽が複数の音より合わせた音であり，長い音のボーカルとともにほかの楽器が短い音をしている場合はよくある．ゆえに，ゲームコンテンツの作成において，長い音があれば必ず長押しにすることはない．そういうことで,「どれくらいの長押しアクションをとらせる」を決める必要がある．

図 A.3 に，横軸は長押しのあるタイムスタンプ数が全タイムスタンプ数においての割合，図 A.3 上の部分の縦軸は Beatmap の数，図 A.3 下の部分の縦軸は確率密度である．これは本研究のデータセット用いて統計した結果である．図 A.3 では，難易度（密度段階）1 の Beatmap の長押しタイムスタンプの割合の平均値はおおよそ 11%，そして，長押しタイムスタンプと全部のタイムスタンプの割合は概ねに t 分布に従う．

そして，各難易度の割合確率分布（t 分布）をフィッティングし，Beatmap 生成において，該当難易度の割合確率分布に従って乱数を生成し，その値を該当 Beatmap の長押し割合とし，長押しの数を決める．

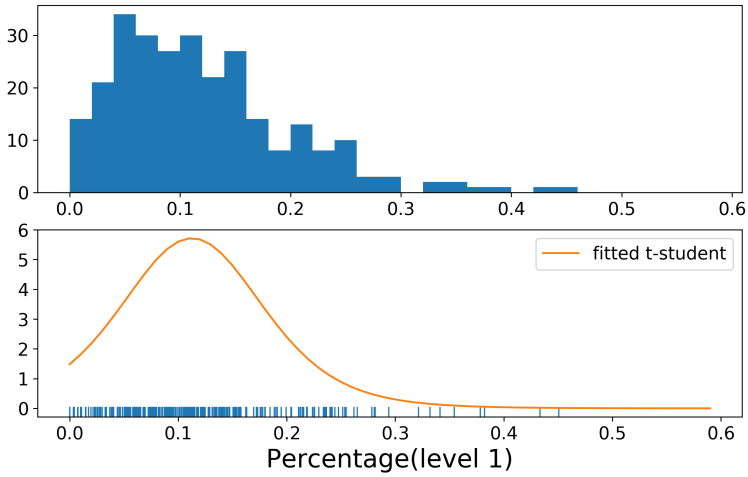


図 A.3: 難易度 1 の長押し数の割合

A.4　長押し長さの割合確率モデル

長押しの数量を決めた後，決めた数量の長押し候補区間を選出する．Beatmap 生成において，長押しの長さはばらばらで，一般的に長いほど少なくなる．図 A.4 では，これは本研究のデータセット用いて統計し，各長押し長さが該当難易度においての割合を示す．横軸は長押しの長さ，縦軸は難易度ごとに分けている．難易度が高いほど，長さの平均値（バイオリン図の縦線）が小さくなり，長い方の長押しが少なくなる（バイオリン図の縦の幅）．

図 A.4 の統計結果に従って，アクション組み合わせのバランスを保つために,「どの長さの長押し候補区間をどれくらい選出するか」を考慮する必要がある．本研究は Beatmap 生成において，該当難易度の長さ割合に従って，長押し候補区間をランダムで選択する．それで，選出された長押し区間は，Beatmap 生成のアクション種類予測に用いられる．

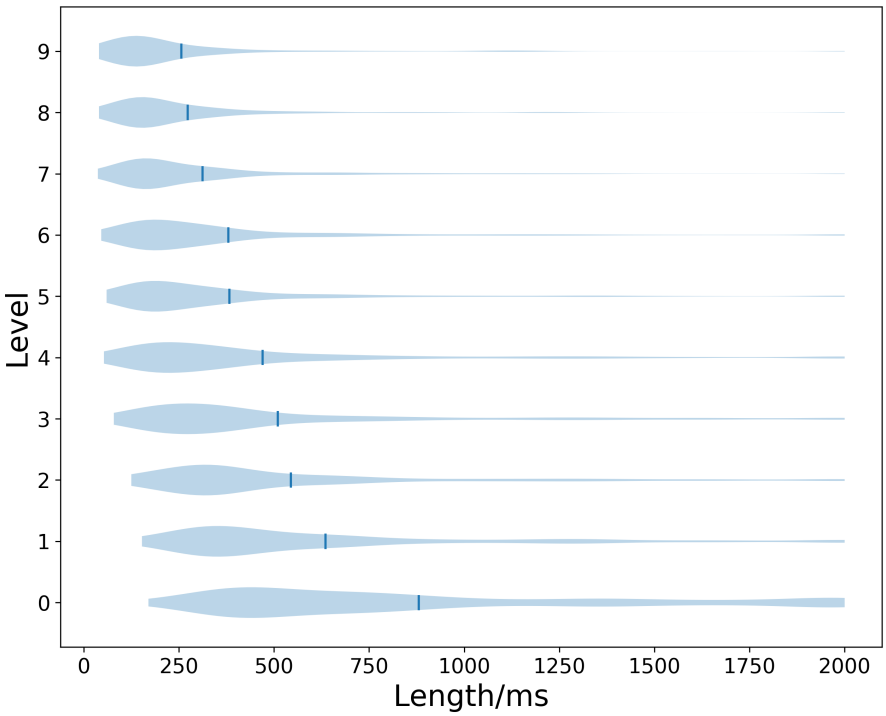


図 A.4: 各難易度長押し長さの割合

A.5　アクション種類の選択確率の調整

アクション種類予測では，Sotfmaxの出力である各アクションの確率を用い，ルーレット式でランダム選択している．だが，その出力はメロディを考慮せず，前のアクション種類の次に各アクションの出現確率だけである．その中に長押しアクションも含めているが，適切な確率とは言えない．そこで，本研究は前述の手法で求めた長押し区間を用い，Sotfmaxの出力である各アクションの確率を調整する．

その前，長押しタイムスタンプや長押しでないタイムスタンプなどを決める必要がある．例として，ゲームボタン各列のアクション種類（アクションなし，クリック，長押し開始，長押し終わり）を0，1，2，3に置き換えたら（長押し維持，つまり2と3の間は全部0にする），256種類の出力を以下のように4グループに分けることができる．そして各タイムスタンプをグルーピングする．

長押しなし（A）0001, 0010, 0100, 1010 .....

長押し開始（B）0201, 0012, 2000, 2020 .....

長押し終わり（C）0301, 0013, 0300, 3030 .....

長押し開始+終わり（D）0302, 0032, 3200, 3032 .....

全体的に，生成されたタイムスタンプを用い，Beatmap生成後半のアクション選択確率調整の流れは次になる：

1) 音楽から，メロディを抽出し，長い音のタイミングをタイムスタンプに当てはまり，長押し候補区間を選別する．

2) その上，A.3節とA.4節の確率モデルで，1) 長押しの数を決める，2) 長さの割合に従って長押し区間をランダムで選別する．長押しの数以上の長押し候補区間は放棄する．

3) 5章で生成されたタイムスタンプを上述のABCD，4種類に割り当てる．

4) 次のタイムスタンプのアクション種類を予測し，出力数は256．

5) タイムスタンプの種類によって，該当グループのアクション種類とその値を全部取り出す．

6) 取り出した値をもって，ルーレット式でアクション種類をランダム選択する．

確率調整を通し，長い音のないタイムスタンプを長押しにしなくなり，長い音のタイムスタンプに長押しに確保することができる．（図A.1に参照）

A.6　メロディ抽出による改善の評価

メロディ抽出による改善の有効性を検証するために，人間手作り及び自動生成における自然さのチューリングテストを行った．被験者は 13 人．自然さとは，リズムゲームにおいて，アクションのタイミングと音楽のリズムの間にズレがあるかどうか，長押しが音楽のメロディに合わせているかどうかなどについての評価である．

実験に用いる Beatmap は 32 本，そのうち，16 本は人間が作った Beatmap，直接生成と確率調整生成は 8 本ずつにする．「手作り 8 − 8 直接生成」と「手作り 8 − 8 確率調整生成」にセットし，全部で 16 セットである．同じセットの Beatmap は同じ歌と同じ難易度とする．

Beatmap は「OSU!」の中に自動プレイをし，プレイ実況を 1 分まで録画する．セット中の 2 本の動画の順番はランダムする．さらに，各被験者のセットの再生順番はランダムで決める．

録画を被験者に再生させ，Beatmap の自然さについて 1〜5 の点数を付けさせる（1 不自然すぎる，2 不自然，3 不自然なところがある，4 そこそこ，5 なかなか）．実験結果は図 A.5 に示す．

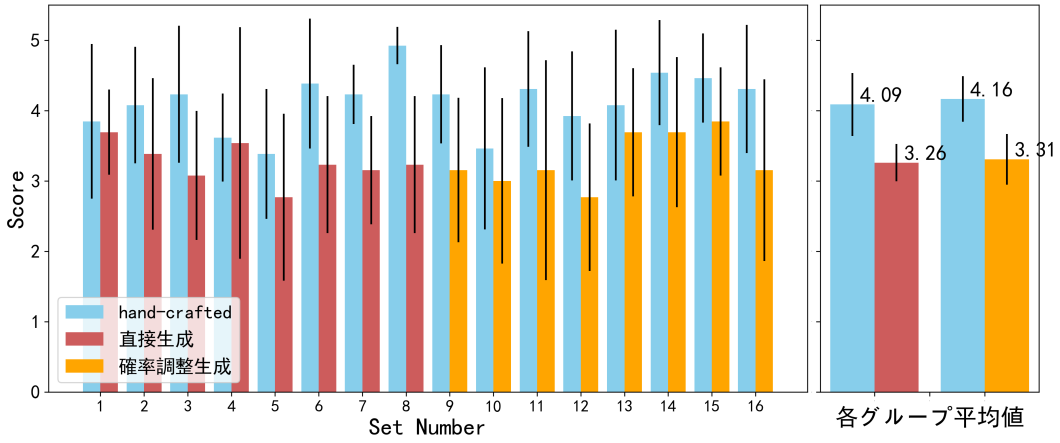


図 A.5: チューリングテスト結果

結果から見ると，確率調整生成手法による著しい改善は得られなかった．一つの原因としては，メロディ抽出手法の正確率が足りない，及びその後の処理において幾度のスムーズによる精度ロスなどと考えられる．この点については，本研究ではメロディ抽出手法の正確率を評価できる手段はなく，それを今後の課題とする．

その他も，不自然であった原因がいくつ存在すると考えられる．例えば，音階の変化に合わせてボタンを変化させるとか，同じメロディのところに同じアクションの組み合わせを繰り返すなど，人間デザイナーがよく用いる手法が，本システムはまだ対応できていない．これらも今後の課題である．

参考文献

[1] リズムゲーム: https://ja.wikipedia.org/wiki/音楽ゲーム

[2] OSU! homepage: https://osu.ppy.sh/

[3] Beatmap download page: https://osu.ppy.sh/p/Beatmaplist? アクセス時間 : 2017 年 6 月

[4] Schluter, Jan, and Sebastian Bock. "Improved musical onset detection with convolutional neural networks.", Acoustics, speech and signal processing(icassp), 2014 ieee international conference on. IEEE, 2014.

[5] Donahue, Chris, Zachary C. Lipton, and Julian McAuley. "Dance Dance Convolution." arXiv preprint arXiv:1703.06891 (2017).

[6] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[7] Salamon, Justin, and Emilia Gómez. "Melody extraction from polyphonic music signals using pitch contour characteristics." IEEE Transactions on Audio, Speech, and Language Processing 20.6 (2012): 1759-1770.

[8] Ikeda, Kokolo, Simon Viennot, and Naoyuki Sato. "Detection and labeling of bad moves for coaching go." Computational Intelligence and Games (CIG), 2016 IEEE Conference on. IEEE, 2016.

[9] Stevens, Stanley Smith, John Volkmann, and Edwin B. Newman. "A scale for the measurement of the psychological magnitude pitch." The Journal of the Acoustical Society of America 8.3 (1937): 185-190.

[10] Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." IEEE Transactions on Signal Processing 45.11 (1997): 2673-2681.

[11] Librosa documentation page: https://librosa.github.io/librosa/

[12] TensorFlow homepage: https://www.tensorflow.org/

[13] Sokolova, Marina, Nathalie Japkowicz, and Stan Szpakowicz. "Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation." Australasian joint conference on artificial intelligence. Springer, Berlin, Heidelberg, 2006.

[14] Z-Score: https://en.wikipedia.org/wiki/Standard_score

[15] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

謝辞

本論文は，筆者が北陸先端科学技術大学院大学 先端科学技術研究科 池田心研究室に在籍していた期間に行った研究をまとめたものです．本研究を遂行するにあたり，多くの方のご支援とご指導を賜りました．

特に池田心准教授は終始暖かくご指導して下さり，誠にお礼を申し上げます．研究の仕方や発表の仕方等々において，厳しくご指摘して頂き，大変勉強になりました．これからも，この経験を糧として活かしたいと思います．

他にも，副テーマ研究において，赤木正人教授から音声処理について色々ご指導を頂きまして，誠に感謝いたします．研究の早期では，Nguyen Duc Tang Tri と Ou Wei 先輩からもたくさんご指導して頂きまして，誠に感謝いたします．

また，筆者の渡日留学を支持し支えていただいた家族，筆者を励ましてくださった友人達に，深く心より感謝しています．

本論文は皆様のご協力無くして完成することは無く，本論文を執筆するに当たり，多大なご支援とご協力を頂いた皆様にこの場を借りて感謝を申し上げます．