Human-in-the-Loop 型適応によるインタラクティブな音楽的拍節解析

山本 和彦 *

概要. 音楽情報処理において拍 (ビート) の抽出は最も基礎的で重要なタスクのひとつである. しかし, この拍の捉え方や解釈は個人によって大きく異なり唯一の正解というものが存在しない. そのため, あらかじめ完璧なアルゴリズムを設計しておくことは非常に困難である. 本稿ではこの問題を解決するため, Human-in-the-Loop 型のビートトラッキングインタフェースを提案する. 本システムではユーザは解析結果の気に入らない箇所をインタラクティブに修正するが, すべての修正箇所を手動で修正する必要はなく, 最初に気付いたごく一部分のみを修正してシステムにフィードバックする. システムはそれをもとに内部の計算モデルをユーザ適応させ, ユーザが直接編集していない残りの部分もユーザの意図に沿うように自動修正する. このユーザとシステムのインタラクションを繰り返すことによってユーザは効率的に所望の結果を得ることができる. これを実現するため, 我々は適応型実行時自己注意機構を導入する. 適応型実行時自己注意機構のアルゴリズムはユーザインタフェースと密接に統合されており, ユーザの局所的な修正の影響を楽曲全体へ効率的に伝搬させることを可能にする. 本稿では提案手法が所望の解析結果を得るためのユーザの労力を劇的に削減することをシミュレーション実験とユーザ実験の両方で示す.

1 はじめに

音楽において拍 (ビート) は最も重要な要素のひとつである. 多くの音楽情報処理システムが最初にビート解析をしたのちそれを後段の処理のための一単位として採用している. しかし, 機械学習の発展によって音楽情報処理アルゴリズムの性能は劇的に向上したにもかかわらず, 完璧なシステムを実現することは困難であり, どうしてもエラーは発生してしまう [1]. さらに, ビートの捉え方や解釈は個人によって異なる. たとえユーザの知覚する拍が楽譜上では作曲者が制作時に想定したものと一致していたとしても, その正確なタイミングは人によってわずかに前や後ろにずれていることも多い. 同一のフレーズであってもどこを拍とすべきかは状況によってどのようにでも変化するということを意図的に示した音楽も多く存在する [2]. このため拍の解析には唯一の正解というものが存在せず, 一意の入力から状況に応じて異なる結果を出力することが求められる. このような問題に事前学習された機械学習システムだけで対応することは原理的に困難である.

本稿ではこの問題を解決するため, Human-in-the-Loop のアプローチによって特定のユーザと楽曲に適応した結果を出力することのできるインタラクティブなビートトラッキングインタフェースを提案する (図 1). 本システムでは, ユーザはシステムの一時的な出力結果に対して気に入らない部分を修正しフィードバックする. システムはそのフィードバック情報を利用して内部のニューラルネットワークモデル (DNN) をこのユーザに対して適応し, 再

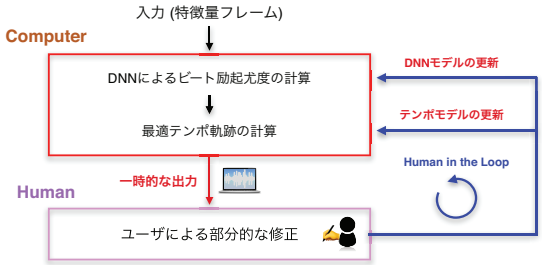


図 1. システム概要. ユーザはシステムの一時的な出力に対して一部分だけを修正してフィードバックする. システムはそれをもとに内部モデルを更新しより所望の結果を出力できるようになっていく.

度出力を更新する. このユーザとシステムのインタラクションを反復することで, システムはよりユーザにとって所望の結果を出力できるようになっていく. ここで重要な点はユーザはすべての修正箇所を手動で修正する必要はなく, ごく少数の最初に気付いた一部分のみの修正をおこなうだけでよいということである. システムはこの少数のユーザ修正から, 残りの直接編集されていない箇所をユーザの意図に沿うように自動で修正できるように内部モデルを適応させる. これにより, ユーザが所望の解析結果を得るための労力を劇的に削減させることができる.

このオンラインのユーザ適応を実現するため, 本稿では新しい DNN モデルである適応型実行時自己注意機構 (Adaptive Runtime Self-Attention, ARSA) を導入する. ARSA の内部パラメータはインタラクティブなユーザインタフェースを密接に統合されている (5.3 章). ARSA は一般的な注意機構 [3] の拡張であり, 他の注意機構と同様にある時点の出力を得るために入力系列全体の情報を集約する. ARSA

* ヤマハ株式会社

と従来の注意機構との大きな違いとして．ARSA はまったく事前学習をおこなわない．ARSA は実行時にのみ，別に用意された事前学習済み DNN の中に組み込まれ，実行時ユーザ適応のためだけに利用される．つまり，我々の DNN モデルは事前学習時と実行時でまったく異なる構造をとる．

我々の DNN モデルでは，事前学習時には楽曲の局所的な情報のみをつかってそれぞれの時刻でのビート励起尤度 (その時刻がどのくらいビートらしいか) を学習する．一方，実行時には ARSA を組み込むことによってある時点の出力を得るために楽曲全体の情報を集約して計算をおこないユーザ適応をおこなう．この局所的な事前学習は学習時の計算コストを削減し，かつ時系列全体を利用した実行時適応はユーザの局所的なエラー修正の影響を楽曲全体へ効率的に伝搬させることができるという利点がある．この局所学習-全体適応の戦略は，ユーザが現実的に修正可能な局所的で少量のフィードバック情報量だけでは楽曲全体のユーザ適応をおこなうには不十分である，という仮定にもとづいている．本稿ではシミュレーション環境上での性能評価に加えて，音楽編集ソフトウェアに普段から慣れ親しんでいる被験者によるユーザスタディをおこない，提案手法がユーザの所望する解析結果を得るための労力を劇的に削減できることを示す．

本稿は，国際学会ですでに発表済みの内容 [4] を簡潔にまとめたものである．

2　関連研究

Human-in-the-Loop のコンセプトは，単にインタラクティブであるだけでなく，人間をシステムの反復の一部として取り込むことで，システム自体がより望ましい結果を出力できるように更新していくことにある．中野ら [5] は複数パートがミックスされた音源から歌声を分離するために，事前学習済みの DNN のファインチューニングをインタラクティブにおこなう手法を提案した．このシステムでは，ユーザが一旦推定された歌声の基本周波数 (F0) の軌跡を GUI 上で修正しファインチューニングのための新たな学習データとすることで，楽曲とユーザへの適応をおこなっている．これは我々のアプローチに近いが，前述したようにユーザの労力を考えたときに現実的に与えることのできる局所的かつ少量の修正データだけでは十分なモデル適応をおこなうには不足である可能性が高いという問題がある．

Human-in-the-Loop のシステムでは取り込む人間が個人とは限らない．HumanGAN [6] では音声合成のための機械学習の品質を向上させるために，学習中に生成された音声が機械が生成したものなのか実際の人間の声の録音かどうかをクラウドソーシングを使って実際に人に判別させて評価関数として利用した．しかしながらこうした不特定多数の人

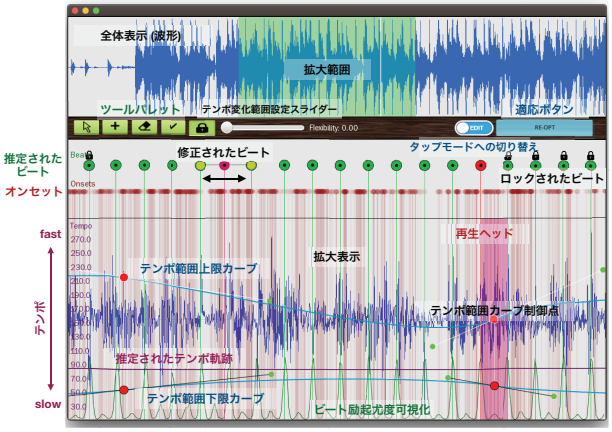


図 2. システム GUI. 楽曲が入力されると波形とともに推定されたビート情報を表示する．システムはユーザに複数種類のビート修正ツールを提供する．

間を利用するアプローチでは多くの人々の平均的な「正解」は得られるが，本稿の目的とする特定個人の所望の結果を得ることには向かない．

Bongjun ら [7] は音響イベントを効率的にアノテーションするための手法を提案した．ユーザがある音響イベントを一箇所指定してシステムに与えると，システムは時系列全体の中から類似した音響イベントの可能性が高い場所を候補場所としてユーザにいくつか提示する．ユーザはそれをチェックし，実際に求めている音響イベントかどうかを判断してシステムにフィードバックする．システムはそれを利用して内部の類似探索モデルを更新し，より精度良くユーザの求めている候補点を提示できるようになっていく．このアプローチは一部我々の提案手法にも取り入れられており，ユーザの局所的な修正の影響が高い可能性が高い場所を推薦するために利用している．

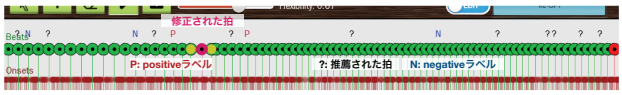
3　ワークフロー

解析対象の楽曲が入力されると，システムはまず事前学習済みの DNN で拍節の解析をおこない，初期状態として画面に結果を表示する (図 2)．楽曲を再生すると，システムは再生ヘッドとともに推定された拍の位置でクリック音を重ねて鳴らすことでユーザに拍位置を提示する．ユーザはそれを聴き，その結果が望ましいかどうかを判断し，もし望ましくなかった場合にはそれを修正する．この修正作業のためには内容に応じて何種類かのツールが用意されている．特定の拍の位置をマウスで移動したり，追加/削除できる基本的な操作に加えて，DNN の出力から最終的なビート位置を確定するためのテンポ軌跡推定 (4 章) をする際の上下限の範囲をベジェ曲線で設定したり，テンポ変動がどれだけ許容されるかをスライダーで設定することができる．テンポ上下限の設定は，推定された拍が明らかに所望のものか

ら外れている場合 (例えば 2 倍や半分のテンポだったとき) にまとめて全体を修正するのに有効である. またテンポ変動率はロックやポップスなどほぼテンポが一定の場合は低く, 逆にクラシックのように大きく変動する楽曲に対しては高く設定すると所望の結果が得られやすくなる. また, ひとつずつ拍位置を修正するのではなく, 再生をしながら同時にキーボードをタップすることでその部分の拍をまとめて直接指定するツールも用意されている.

ユーザはこれらのツールを使ってごく少数の気に入らない箇所のみを修正し, 残りの部分に修正箇所があるかどうかは関知する必要はない. ユーザが修正したのち, 図 2 右上の最適化ボタンを押すことで内部モデルの適応プロセスが開始され, 残りのユーザが直接編集していない箇所にも自動修正がおこなわれていく. この適応は反復ごとに経過が画面に表示されるので, ユーザは任意のタイミングで適応プロセスをストップし, 結果を聴いて確認する. この反復作業をユーザは楽曲全体の推定結果が望ましいものになるまで繰り返す.

ここで想定し得る問題として, 単純にモデル適応をおこなった場合, 本当は修正してほしくないところまで修正されてしまう可能性がある. これを防ぐため, 提案手法には修正が必要な可能性が高い場所には大きな影響を, 逆にそうではない場所にはより小さな影響のみを及ぼすようなインタラクティブな適応プロセスを組み込まれている (5.3 章). ユーザがある拍を修正すると, システムはその修正によって影響が大きい可能性がある場所を何箇所か提示する (下図の？マーク). ユーザはその場所を再生して,



本当に修正が必要かどうかを判断し, 必要な場合は positive, そうでない場合は negative ラベルを割り当てる. もし提示された拍が修正が必要だったとしてもユーザはその可否を判断するだけでよく, 直接手で修正する必要はない. これはユーザの労力を削減するうえで重要な点である. なぜなら拍の位置の修正作業は, 単に音を聴いて可否を判断するだけよりずっとコストが高いからである. このラベリング作業は必須ではないが, 適応ボタンを押す前にわずかな追加タスクとしてこれをおこなうことでユーザは不都合なモデル適応を避けることができる.

4 ビートトラッキング

本稿で扱うビートトラッキングのアルゴリズムは基本的に Temporary Convolutional Network (TCN) をもちいた手法に基づいている [1]. この手法は 2 段階のプロセスから成り立つ. まず, 楽曲を細かな時間フレームに分割 (サンプリング周波数 44.1kHz で

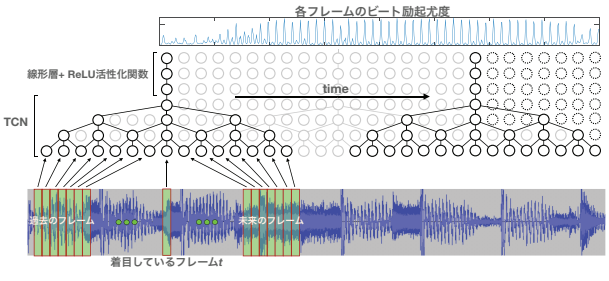


図 3. 事前学習時にはある時刻周辺の局所的な情報を入力してその時刻のビート励起尤度を出力する.

窓長 4096 サンプル, Hanning 窓, ホップサイズ 2048 サンプルを採用) しフレームごとの特徴量 (24 次元 MFCC 時間差分を採用) を計算し、これを DNN に入力する. この DNN は各時刻の前後約 2.4 秒分を含めた局所的な 96 フレームを入力として, その時刻におけるビート励起尤度 $\in [0, 1]$ を出力する. 次に 2 段階目として, 求まったビート励起尤度の系列をもとに楽曲全体を通して最も尤もらしいテンポ軌跡を計算することで実際の離散的なビート位置を確定する [8]. 我々の DNN アーキテクチャは 3 段の TCN ブロックと直列に接続された 3 層の ReLU 活性層付き線形層から成る (図 3). また, ビート励起尤度系列からの最尤テンポ推定には隠れセミマルコフモデル [9] による Viterbi アルゴリズムによって計算をおこなう.

5 適応アルゴリズム

5.1 ユーザ修正に対する適応目的関数

ユーザが拍位置を修正すると, システムは現在の解析結果との誤差からモデル適応をおこなうための目的関数を作成し, それを確率的勾配法で最小化する. ここでは例としてある拍の位置を移動した場合を想定して説明するが, 他のツールでの修正についても同様の考え方を適用できる. 適応後に目指すべき状態として, ユーザが修正後のビートを b_i , その前後のビートを b_{i-1}, b_{i+1} とすると, DNN の出力が b_i の位置で 1 に近く (ビート励起尤度が高く), 逆に拍と拍の中点 $b_i^- = \frac{b_i+b_{i-1}}{2}$, $b_i^+ = \frac{b_i+b_{i+1}}{2}$ で 0 に近く (ビート励起尤度が低く) なるようにする (図 4). よって最小化すべき損失 $L(t)$ は修正位置周辺の各フレーム t (図 4 のオレンジのフレーム) 上で,

$$L(t) = \left| DNN(t) - \left(1 - min(1.0, \frac{|b^c - t|}{W})\right)\right|_2^2 \quad (1)$$

のように定義される. ここで, b^c と W はそれぞれフレーム t から最近傍のビートと窓長, $DNN(t)$ はフレーム t での DNN の出力である.

5.2 適応型実行時自己注意機構 (ARSA)

我々の DNN モデルは事前学習時にはある時間フレームでのビート励起尤度を出力するためにその

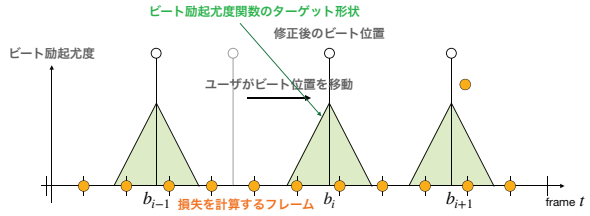


図 4. ユーザ修正から DNN へ与える損失の作成.

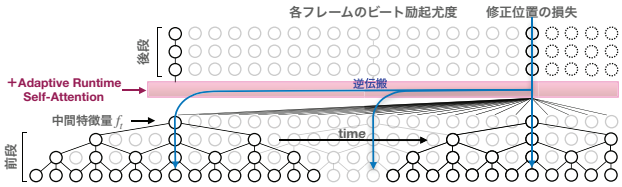


図 5. 実行時には事前学習された DNN を分割し, 新たに ARSA ブロックを組み込む.

周辺の局所的な情報のみを利用する構造をしている (図 3). 一方実行時には, その事前学習した DNN を前半ブロック (3 ブロックの TCN) と後半ブロック (TCN ブロック以降の出力段側) の 2 つに分割して, その間に ARSA を挿入する (図 5). つまり我々のモデル構造は学習時と実行時で異なる. ARSA は学習時の局所的な視野の適用とは異なり, 楽曲全体を通した中間特徴量 (前半ブロックの出力) を集約し, 後半ブロックのための入力を出力する. この学習時と実行時で異なる構造をとる新しいタイプの DNN モデルは, 学習時の計算コストの削減とユーザの局所的なフィードバックの影響を楽曲全体へ効率良く分配する実行時適応をバランス良く両立する.

前半ブロックの出力である中間特徴量を $f^t \in \mathbb{R}^D$ とすると, ARSA はまず通常の注意機構と同様に各時刻 t のキュー $Q^t = \mathbf{W}_Q \cdot f^t$, キー $K^t = \mathbf{W}_K \cdot f^t$, バリュー $V^t = \mathbf{W}_V \cdot f^t$, $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V \in \mathbb{R}^{D\times D}$ を計算する. これらのベクトルから ARSA のフレーム t におけるフレーム i への注意マップは,

$$d(t \to i) = (w^t \otimes Q^t) \cdot (P_e(|t-i|) \otimes K^i), \quad (2)$$

で計算される. ここで, $w^t \in \mathbb{R}^D$ はユーザのラベリングによって決まる重み, $P_e(\delta t) \in \mathbb{R}^D$ は相対位置埋め込みである [3]. この $d(t \to i)$ を i についてソートし値の大きなものから N フレーム (ここでは N=512 を採用) サンプリングした集合を $\mathbb{C}_t$ とすると ARSA の出力は,

$$y_t = (1-\alpha) \cdot V^t + \alpha \sum_{i \in \mathbb{C}_t} \frac{e^{d(t\to i)}}{\sum_{j\in\mathbb{C}_t} e^{d(t\to j)}} \cdot V^i, \quad (3)$$

となる. α は重み係数である ($\alpha = 0.5$ を使用).

5.3 インタラクティブな注意マップ最適化

式 (2) の w^t は ARSA 独特のパラメータでありユーザのインタラクティブなラベリングによって決まる. 3 章で述べたように, ユーザが拍を修正するとシステムはそこからモデル適応をおこなったときに影響が大きい可能性が高い場所がいくつか提示する. この提示箇所は注意機構本来の仕組みに基づいて決定される. 注意マップ $d(t \to i)$ が大きいということはフレーム t はよりフレーム i に対して注目しているということである. 逆に言えば, フレーム t 上で与えられた損失はフレーム i により多く逆伝搬するということを示している. よって単純に注意マップの値が大きな箇所付近の拍が影響を受けやすいとみなすことができるため, それらを提示する.

提示された拍に対してユーザは本当に修正が必要かどうかを判断し, positive/negative ラベルを割り当てる. ラベル自体は拍ごとに割り当てられるが, システム内部ではそれらの拍周辺のフレームに対してラベル付けがおこなわれる. これらの関係から w^t は Fisher's criterion [7] と同様に,

$$w^t = \frac{(avg(A^{t,pos}) - avg(A^{t,neg}))^2}{std(A^{t,pos})^2 + std(A^{t,neg})^2}, \quad (4)$$

のように求められる. ここで, $A^{t,pos}$ と $A^{t,neg}$ はそれぞれフレーム t に対する positive/negative フレーム i での $Q^t \otimes P_e(|t-i|) \otimes K^i$ である. この w^t はユーザの修正によって影響を受けてほしくない場所への影響を減らし, 逆に影響を受けてほしい場所には増やす効果がある.

6 評価

6.1 シミュレーション実験

提案手法の性能を評価するため, まずシミュレーション環境下での実験をおこなった. この実験では ARSA 有りと無しのときのモデル適応を比較する. ここで ARSA 無しの場合とは, 単純に事前学習した DNN 構造をそのまま使い, ユーザが修正したデータを新たな学習データとしてファインチューニングすることを意味し, これは中野ら [5] の手法とほぼ同様のものとなる. 検証用データとしては事前学習データには一切含まれていない SMC MIREX [10] を使用した. このデータセットにはビート位置のアノテーション情報とともに様々なジャンルの 217 曲が収録されている. またすべての曲の長さは 40 秒に揃えられている.

実験の手続きとしては, まずそれぞれの曲において解析をおこない, 最も大きなエラー (拍周辺のフレームの平均二乗誤差) を観測する拍を曲中から探し出し, それを自動的にアノテーションされている正解位置へ修正する. 次にシステムに対して 100 反復のモデル適応を実行する. この自動での拍修正とモデル適応の組み合わせを 5 回繰り返し, 計 500 回の適応反復を得る. ARSA のラベリングについては自動で各修正ごとにランダムに選択された 5 箇所ず

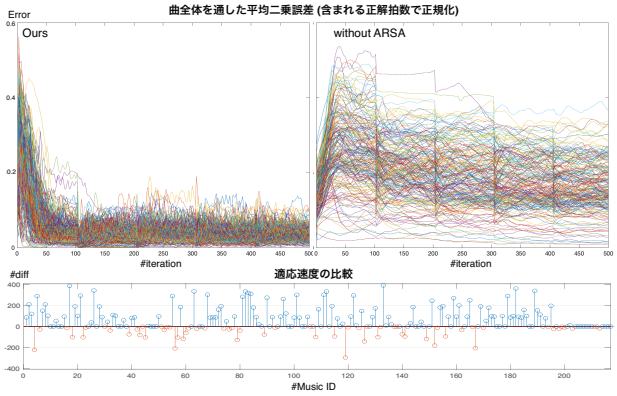


図 6. シミュレーション実験結果．上段：反復回数にともなう各楽曲全体を通した平均二乗誤差の推移．下段：提案手法と比較手法間での収束速度の差分．

つおこなう．ここで可否の判断については，提示された拍がアノテーションから ±70 ms 以内に入っていれば positive，それ以外を negative とみなした．

図 6 に結果を示す．上段に各楽曲全体を通した平均二乗誤差 (各曲ごとに含まれる正解拍の数で正規化後) を示している．明らかに提案手法はほとんどの楽曲において小さなエラーへすばやく収束しているのがわかる．一方，比較手法では 100 反復ごとにおこなわれる拍修正のタイミングで階段上にエラーは下がるものの、曲全体としてはなかなか誤差が小さくならないことがみてとれる．これは局所的な修正でのファインチューニングだけでは楽曲全体に対して適応をおこなうには不十分であり，ARSA の導入によってそれを大きく改善できることを示している．図 6 下段はどちらの手法のほうがどれだけ適応が速く進行するかの比較を示しており，F1 (曲全体を通した正解率) の値が初めて 0.8 を超えたときの反復回数を N^A (提案手法) と N^B (比較手法) とすると，それらの差 $N^B - N^A$ を示している．つまり大きな正の値 (青) になるほど提案手法のほうがより速くほとんどの拍が正解の状態に到達したことを示しており，ほとんどの曲において提案手法のほうが速く高い正解率に到達していいることがみてとれる．実際，統計的にも 156/217 曲においてカイ二乗判定で有意 ($p<0.05$) に提案手法のほうが速く収束している結果となった．また，計算コストとしては，このシミュレーション実験において 217 曲すべてを処理するのに提案手法で 5.5 時間，比較手法で 5 時間を要した．つまり平均して一曲あたり 8 秒の差であり，これはユーザが実際に修正をおこなう時間に比べて実用的に無視できるオーバーヘッドと言える．

6.2 ユーザスタディ

6.2.1 セットアップ

提案手法がどれだけユーザの修正効率を向上させるかを評価するため，ユーザスタディをおこなった．この実験では，対象楽曲の解析結果に含まれるエラーを修正するタスクでの作業時間を計測する．この作業時間の基準として，我々は対象楽曲の総再生時間を設定した．これは以下の理由によるものである．既存の方法で最も速くビート位置をアノテーションする方法は，曲全体を通して一度で正確にタップすることであり，この最短時間は曲の長さ分である．よってこの時間より短時間で被験者が修正を終えることができれば，提案手法の有効性を示すことができると考えられる．

被験者として普段から音楽編集ソフトウェアに慣れ親しんでいる 5 名を集めた．まず最初にそれぞれの被験者に我々のインタフェースの使い方を説明し，実際に触って練習してもらった (45 分)．次に本番タスクとして，できる限りすばやくテスト楽曲に含まれる解析エラーを提案インタフェースを使って修正してもらった．このテスト楽曲には RWC Jazz データセット [11] から 2 曲 (No.40 - 7 分 41 秒，正解ビート数 481 個，No.41 - 6 分 6 秒，正解ビート数 637 個) を選んだ．どちらの楽曲も初期状態での解析では手動ですべて修正するには現実的ではない多くの数のエラーが含まれていた．すべての被験者がこの 2 曲を修正した．また，被験者依存による楽曲への解釈の差異を避けるため，我々は正解ビートの位置でクリック音を重ねた「正解音楽ファイル」をあらかじめ作成しておき，このタスクを始める直前に必ずすべての被験者に聴いてもらい，その「正解」を目指して修正をおこなうように指示をした．修正するためには 3 章で述べたあらゆるツールを使用可能としたが，タップ指定のツールだけは連続して 5 秒以上使用することを禁じた．また，一箇所修正するごとに必ず適応ボタンを押すことも義務付けた．このタスクをおこなっている間，我々は時間を計測するとともに F1 値の時間変化を記録した．それぞれのタスクは F1 値が 0.9 を超えた段階でたとえ被験者が曲全体を聴いて最終チェックをしていなかったとしても打ち切って終了させた．最後にこの本番タスクを終えたのち，インタフェースの使用感についての簡単なインタビューをおこなった (10 分)．

6.2.2 結果

図 7 に本番タスクでの各被験者の F1 値の推移結果を示す．垂直の点線は楽曲の総再生時間を示している．それぞれの楽曲において 4/5 人と 3/5 人の被験者が曲の再生時間より早くタスクを終了することができた．また，残りの被験者もほぼ再生時間と同等の時間でタスクを終えている．これは，ほとんどの被験者が曲全体を聴いてはおらず，タスクが完了したことを気付く前 (全体をチェックしていないため) に所定の精度に到達して打ち切られているということを意味する．もちろん，現実的に提案インタフェースを使用する場合は最後に曲全体を聴いて

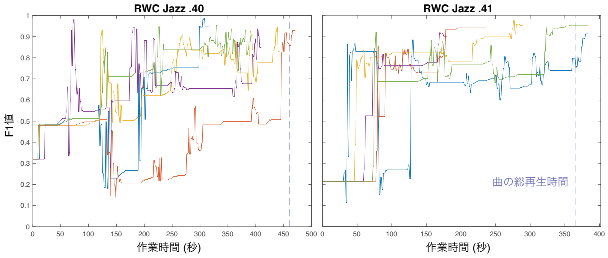


図 7. 作業時間経過 (横軸) にともなう各被験者ごとのビート正解率 (縦軸, F1 値). の推移.

チェックをする必要があるので曲の総再生時間より作業時間が短くなることはありえない．それゆえあくまで条件付きではあるが, 提案手法は効率的なエラー修正をおこなうことができたと結論付けることのできる結果を得たと言える．

インタビューについては．我々はあらかじめ 2 つの質問を用意した．ひとつめは「音楽制作ソフトにこのインタフェースが実装されたら使ってみたいですか？」である．この質問に対してはすべての被験者が Yes と返答した．2 つめの質問は,「このシステムを使用していた上で何か問題がありましたか？」である．この質問に対しては, 何人かの被験者は **A**:「様々な種類の修正ツールが用意されているため, 発見したエラーに対してどのツールを使えばよいのかわからなかった」と答えた．また, **B**:「適応ボタンを押したあとの反復を自分で止めなければいけないが, どのくらい待って止めればよいのか分からなかった」という回答もあった．

6.2.3 議論

インタビューにおけるコメント **A** に対しては, 被験者がインタフェースにより時間をかけて慣れ親しむことで解決していく問題だと考えられる．しかしながら, もっとより良い解決法としてシステムが現在のユーザのおかれている状況を認識し, 最適な修正ツールを推薦することは今後の課題である．コメント **B** に対しては, 数値計算の反復という概念自体が一般のユーザには馴染みが薄く, そうしたユーザに反復回数を任意で決定させるのは現実的ではないと考えられる．それゆえにシステムが自動的に反復をストップさせることができるような指標を設計することが求められるが, 我々の適応プロセスは一意に収束するものではないためそれは簡単ではなく今後の課題である．似たような問題として, 我々の解いている最適化問題では楽曲全体のエラー率が単調に減少せず, ときに反復の最中に増大することもある．これはユーザにとってみればエラーの修正作業をしたにもかかわらず曲全体のエラーが一時的にも増えたようにみえる, という直感に反する事態となり, 見せ方には今後工夫が必要だと考えられる．あああ

7 まとめと今後の展望

本稿では特定のユーザと楽曲へ Human-in-the-Loop のアプローチで適応できるインタラクティブなビートトラキングインタフェースを提案した．これを実現するため, 学習時の計算コストを削減でき, かつユーザの局所的なフィードバックの影響を入力系列全体へ伝搬させて効率的な実行時適応をおこなうことのできる適応型実行時自己注意機構を導入した．我々はいくつかの実験をおこない提案手法の実行可能性と性能を検証した．こうした機械学習モデルの事前学習は局所的におこない, 逆に特定対象への実行時適応は系列全体の文脈を集約するアプローチは, ビートトラッキングだけではなく, コード認識, 歌声合成, 音源分離などといった他の音楽情報処理, さらには 3D アニメーションの生成や動画編集, といった多くの時系列データを扱う領域で有効だと期待される．

参考文献

[1] M. E. P. Davies and S. Bock. Temporal convolutional networks for musical audio beat tracking. *EUSIPCO*, 2019.

[2] S. Reich. Clapping music (1972), Pianophase (1967)

[3] . Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *NeurIPS*, 2017.

[4] K. Yamamoto. Human-in-the-Loop Adaptation for Interactive Musical Beat Tracking. *ISMIR*, 2021.

[5] T. Nakano, Y. Koyama, M. Hamasaki, and M. Goto. Humangan: Generative adversarial network with human-based discriminator and its evaluation in speech perception modeling. *ICASSP*, 2020.

[6] K. Fujii, Y. Saito, S. Takamichi, Y. Baba, and H. Saruwatari. Interactive deep singing-voice separation based on human-in-the-loop adaptation. *IUI*, 2020.

[7] B. Kim and B. Pardo. A human-in-the-loop system for sound event detection and annotation. *IUI*, 2018.

[8] F. Krebs, S. Bock, and G. Widmer. An efficient state-space model for joint tempo and meter tracking. *ISMIR*, 2017.

[9] R. Chen, W. Shen, A. Srinivasamurthy, and P. Chordia. Chord recognition using duration-explicit hidden markov models. *ISMIR*, 2012.

[10] A. Holzapfel, M. E. P. Davies, J. R. Zapata, and J. L. Oliveira, A. Srinivasamurthy, and P. Chordia. Selective sampling for beat tracking evaluation. *EEE Trans. Audio, Speech, and Language Processing*, 2012.

[11] M. Goto, H. Hashiguchi, T. Nishimura, R. Oka. RWC Music Database: Popular, Classical, and Jazz Music Databases. *ISMIR*, 2002.