

2023年度 秋学期

卒業論文

Transformerを用いた対戦格闘ゲームにおける音声特徴量抽出

指導教員: THAWONMAS RUCK

立命館大学 情報理工学部

卒業研究3 (FD)

コース: 知能情報

学生証番号: 2600190411-5

氏名: CHEN JUNZHANG

要旨

背景：背景音や効果音など様々な音は現代のビデオゲームに重要な役割を果たしている。音はプレイヤーに没入感を向上させるとともに、ボーナスアイテムや敵などの位置も音で認識でき、目以外の情報を与えられる。2013年に開発された対戦格闘ゲーム「FightingICE」はAIの研究に活躍し、その拡張版の「DareFightingICE」はさらに2022年のIEEE CoG(IEEE Conference on Games)の公式試合舞台として使われている。「DareFightingICE」は盲人にも遊べる音声のみのモードが存在し、音声のみの入力とする対戦AIの開発も課題となっていた。

目的：TransformerモデルはAttentionの仕組みがあるため、元の方式より時系列のベクトル間の特徴を掴める。本研究では、TransformerモデルがMel-spectrogramやFFTなどの音声解析システムに比べて音声特徴抽出機能が向上しているかどうかを検証することである。従って、元の解析システムで作られたAIより高い勝率が得る対戦AI「Blind-T」を作ることも目指している。

実験方法：トレーニング済みの「Blind-T」は盲人実力標準に近いAI「MctsAI65」（モンテカルロ探索により作る）と対戦し、勝率で「Blind-T」の強さを評価し、元のAIの対戦結果と比べて検討する。また、異なる解析システムで1000回実行の時間を測り、その計算コストを比較する。

結果：自作の音声解析システムと強化学習モデルで「Blind-T」を作った。「Blind-T」は元の「BlindAI」の63%の勝率より高く超え、「MctsAI65」との対戦に96%の勝率を得た。また、2022年のチャンピオンAI「Sounder」にも勝ち、54%の勝率を取った。

結論：Transformerモデルから組み立てた深層学習モデルは元の方法より計算資源が多く消耗するが、以前の方法と比べて、よりよく音声特徴を抽出するも確実である。

目次

第 1 章	はじめに	1
1.1	研究背景	1
1.1.1	「FightingICE」時代	1
1.1.2	「DareFightingICE」時代	2
1.2	研究目的	4
1.3	関連研究	4
1.4	論文構成	5
第 2 章	従来の研究	6
2.1	FPS ゲーム中音声データ入力による強化学習 AI	6
2.2	「DareFightingICE」の深層強化学習 AI 「BlindAI」	8
第 3 章	システムの提案	10
3.1	提案システムの概要	10
3.2	生データの前処理	10
3.3	システムの詳細	12
3.3.1	システムの構造	12
3.3.2	Transformer 部分	13
3.4	異なるエンコーダーの計算コスト比較	17
3.5	ゲーム AI 構造と更新ポリシー	17
3.6	技リストとヒットボックス	18
3.7	評価方式	20
第 4 章	実験	21
4.1	実験環境	21
4.2	実験の流れ	22
4.3	ハイパーパラメーター	22

4.3.1	技リスト (agent.py)	22
4.3.2	モデル訓練 (train.py)	23
4.3.3	音声解析システム (エンコーダー)	23
4.4	実験結果	23
第5章	おわりに	26
5.1	まとめ	26
5.2	プロジェクトのアクセス先	27
5.3	今後の展望	27
謝辞		28
参考文献		29
付録		31
付録.A	提案システムのソースコード	31
付録.A-1	encoder.py	31
付録.B	ゲームA I構造のソースコード	44
付録.B-1	agent.py	44
付録.B-2	model.py	56
付録.C	トレーニングのソースコード	63
付録.C-1	Train.py	63
付録.D	テストのソースコード	92
付録.D-1	PvJ.Py	92
付録.D-2	fight_run.py	94
付録.D-3	fight_agent.py	96
付録.E	結果分析のソースコード	97
付録.E-1	visualize.py	97
付録.E-2	analyze_fight_result.py	100

図目次

1.1	「FightingICE」実況場面	1
1.2	2021年大会優勝結果 [1]	2
1.3	2022年大会優勝結果 [1]	4
2.1	音声解析エンコーダーの構造 [2]	6
2.2	異なるエンコーダーの成功率 [2]	7
2.3	「BlindAI」のエンコーダー構造図 [3]	8
2.4	「BlindAI」強化学習モデルのネットワーク構造 [3]	8
3.1	左右チャンネルの波形図	11
3.2	ゼロパッティングの仕組み	11
3.3	システム構造	12
3.4	Transformer Encoder 構造	13
3.5	Transformer モデルの構造図 [4]	14
3.6	BLEU SOCRE[5]	15
3.7	VIT 構造 [6]	16
3.8	「Blind-T」アーキテクチャ	17
3.9	技 STAND _A	19
3.10	技 CROUCH _{FB}	19
4.1	「Blind-T」学習曲線	24

表目次

2.1	異なるエンコーダーの対戦結果 [3]	9
3.1	異なるエンコーダーの計算コスト	17
3.2	技リスト	18
4.1	「Blind-T」技リスト	22
4.2	Train.py のハイパーパラメーター	23
4.3	音声解析システムのハイパーパラメーター	23
4.4	対戦結果 1 (30 GameSet)	24
4.5	対戦結果 1 (30 GameSet)	25

第1章

はじめに

1.1 研究背景

1.1.1 「FightingICE」時代

2013年に開発された対戦格闘ゲーム「FightingICE」はAIの研究に活躍し、毎年自作のAIで対戦する大会も開催している。「FightingICE」はキャラクターごとが56の動作が存在する簡潔な対戦格闘ゲームであり、JavaまたはPythonでAIを開発でき、アルゴリズムにより独特な攻撃や移動の組み合わせが作られる[1].



図 1.1 「FightingICE」実況場面

実行場面は図 1.1 のように示したものである。設計者は Player1 をコードで操作でき、従来の大会には深層学習の DQN アルゴリズムや強化学習で作られた AI が登場した。訓練途中だけではなく、実際対戦する時にも AI は敵と自分の位置、両方の HP、エネルギーの量、音声情報など全ての情報がもらえる。

オフィシャルのデフォルト AI はモンテカルロ探索により作られた「MctsAI」である。毎年の大会の参加者は「MctsAI」を勝つことが一ステップで、歴代のチャンピオン AI にも勝ち続けることが目標である。大会には一番勝率が高い AI は優勝である。また、AI のアルゴリズムは軽量化に向かい設計しないと、計算コストが高いためオフィシャルのパソコンで対戦する際に性能が落ちる可能性もある。従って、アルゴリズムの設計も工夫する必要がある。

図 1.2 は 2021 年の大会の優勝結果を示したものである。勝つ AI は中国のネットイーズのゲーム AI ラボに生まれる「BlackMamba」である。「BlackMamba」は PPO ポリシーに基づき、強化学習で作られた AI であり、以前の AI と当年度の参加者 AI に比べ、ほぼ完勝の成績で 2021 年の大会に優勝した。従って、完全情報獲得の AI 試合はほぼ「BlackMamba」により定着となり、「FightingICE」は他方向でゲーム AI の道を探し始めた。

2021 Competition Results

Congratulations to

1st Place Winner (AI Name: BlackMamba):

Peng Zhang, Guanghao Zhang, Xuechun Wang, Sijia Xu, Shuo Shen, and Weidong Zhang (NetEase Games AI Lab, China)

2nd Place Winner (AI Name: WinOrGoHome):

Weijun Hong (NetEase Games AI Lab, China)

3rd Place Winner (AI Name: Thunder2021):

Eita Aoki (Japan)

図 1.2 2021 年大会優勝結果 [1]

1.1.2 「DareFightingICE」時代

現代のビデオゲームは解像度が高い画面だけではなく、豊富な背景音や効果音もゲームの没入感に重要な役割を果たしている。ストーリーに合わせる BGM はプレイヤーに没入感が提供し、敵やアイテムの位置を提示する効果音がプレイヤーに目以外の情報を与えられる [7]。

音声機能を特化した「DareFightingICE」は「FightingICE」の拡張版であり、2022 年の IEEE CoG(IEEE Conference on Games) の公式試合舞台と使われている。「Dare-

FightingICE」は目の不自由な人にも遊べる音声のみの盲人モードが存在する。そのため、音声だけを入力とする対戦 AI の開発も課題となった。

2022 年以後の「Fighting Game AI Competition」[1] は従来の AI 設計の試合以外、サウンドデザインの試合も開催した。サウンドデザインは既存の BGM や効果音の変更だけではなく、ゲームのソースコードも改変でき、元が存在しない音声効果を追加することも可能である。AI 設計は音声特化の「DareFightingICE」に合わせて、音声情報しかもらえない盲人 AI を設計することになった。訓練する際に、モデルの報酬やロスを計算するため、自分のアルゴリズムに合わせて音以外の情報をもらうこともできるが、試合中には音声情報しかもらえない。

Khan, I により [7], 「DareFightingICE」のデフォルト AI はモンテカルロ探索による作られた「MctsAI」を調整することで、目の不自由な人と似たような実力になった「MctsAI65」である。「MctsAI65」は「MctsAI」の毎フレームの処理時間を減少することにより弱化した AI である。「MctsAI」の処理時間は 16.5ms であり、「MctsAI65」の処理時間は 6.5ms しかない。

図 1.3 は 2021 年の大会の優勝結果を示したものである。サウンドデザインの優勝者は立命館大学の Adam Kumar である。優勝デザインはソースコードを調整しなく、BGM と音声効果を変更することで対戦の没入感を増加し、被験者の勝率も上昇した。また、盲人 AI の優勝者は株式会社 HEROZ の青木さんである[8]。優勝 AI 「Sounder」は技「キック」と「波動拳」の組み合わせで、一切深層学習モデルを使わずに、「MctsAI65」を勝ち、優勝になった。優勝者は格闘ゲームを深く理解し、簡潔なポリシーで強い“AI”を作った。

2022 Competition Results

Congratulations to the Winners of Sound Design Track

1st: SoundEffectProject
by Adam Kumar, Ritsumeikan Univ., Japan

2nd: D.A_GameSoundDesign
by Dossymova Aiya, Ritsumeikan Univ., Japan

3rd: DefaultSoundDesign
by Team DareFightingICE, Japan

Congratulations to the Winners of AI Track

1st: Sounder
by Eita Aoki, HEROZ.inc, Japan

2nd: MctsAi65
by Team DareFightingICE, Japan

3rd: BlindAI
by Team DareFightingICE, Japan

図 1.3 2022 年大会優勝結果 [1]

1.2 研究目的

強い対戦 AI を訓練するため、音声情報から特徴を取りだすことが必要である。生音声データから特徴抽出する音声解析システムはエンコーダーという。従来のエンコーダーはフーリエ変換やメルスペクトログラムで作られたものである。本研究は近年音声識別領域に活躍している Transformer モデルで音声特徴抽出の音声解析システムを作り、従来のエンコーダーよりよく音声データを解析し、より高い勝率を得る対戦 AI を作る。

1.3 関連研究

生音声を異なるエンコーダーで音声特徴を抽出し、ゲーム AI に訓練する研究がある [2]。また、「DareFightingICE」に音声のみを入力とする深層強化学習 AI の研究も存在する [3]。Transformer モデルの発明の文章も参考し [4]、Transformer モデルのエンコーダー部分で画像の特徴抽出する研究もある [6]。

1.4 論文構成

本論文の構成は以下の通りである。まず、第1章は研究の背景、目的及び関連研究について述べた。そして、第2章では従来の研究について述べた。次に、第3章では、エンコーダーの概要、設計のヒント、エンコーダーの構造、評価方式などの提案手法を述べた。さらに、第4章では、実験環境、流れ、ハイパーパラメーター及び結果を示し、自作のモデルの有用性を検証した。最後に第5章では、本論文のまとめと今後の展望について述べた。

第 2 章

従来の研究

2.1 FPS ゲーム中音声データ入力による強化学習 AI

Hegde, S 等により [2], 従来の強化学習 AI は音声を使用せず, 画面の情報を使用するだけで既に簡単な Atari ゲームと FPS ゲームに人を超える実力を持っていた. しかし, FPS ゲームには画面だけではなく, 音声で自分と相手の位置を理解することも重要な能力であり, 音声情報も AI に与えれば, さらに強い AI が作られると考え, 音声解析システムを設計した.

生の音声データは解析システム(エンコーダー)で音声の特徴を抽出して特徴データ(256 個や 512 個など)になり, 強化学習のモデルに入力する. Hegde, S 等は三つのエンコーダーを設計し, その構造は図 2.1 のように示した.

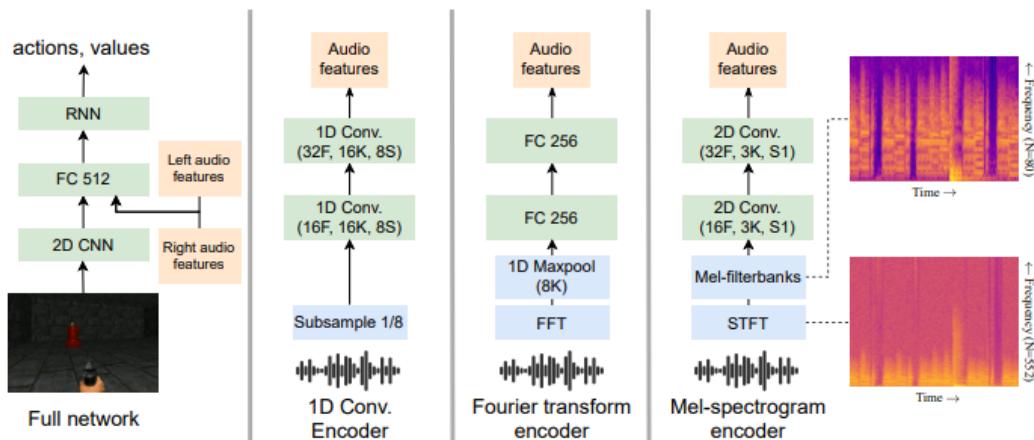


図 2.1 音声解析エンコーダーの構造 [2]

1. 1D-CNN: 生の音声データは八分の一でダウンサンプリングし, 2 つの一次元の畳み込みニューラルネットワークで特徴抽出し, 32×5 の音声特徴ベクトルが得られた.
2. FFT: 生の音声データは高速フーリエ変換で周波数領域のスペクトルを計算し, $sFFT = \log FFT(s) \in R^{\frac{n}{2}}$ により倍率の自然対数 sFFT に変更した. そして, 一次元のマックスプーリングでダウンサンプリングし, 2 つの全接層 (Fully Connected Layer) に通し, 256 個の音声特徴ベクトルが得られた.
3. Mel-spectrogram: 生の音声データはホップで移動する窓付き信号の短期フーリエ変換 (STFT) で周波数領域のスペクトルに変換した. そして, 周波数ごとをメルスケールで処理し, メルスペクトログラムで処理した. ハイパーパラメーターの設定はホップサイズ 10ms, ウィンドウサイズ 25ms, メル周波数成分 80 個を選択した. その後, スペクトログラムデータが 2 つの 2 次元畳み込み層からなるネットワークに与え, 最終的には $32 \times 40 \times 1$ の音声特徴ベクトルが得られた.

次に, この音声解析システムを FPS ゲーム「ViZDoom」の音源探し任務 AI に使用し, 性能を測った. AI は PPO ポリシーによる強化学習で作られたものである. 図 2.2 は異なるエンコーダーで任務に完成する成功率を示したものである. 音声がある際に成功率は明らかに上昇し, 音声解析システムで処理したデータはゲーム AI の強化に対して有用性が証明できた.

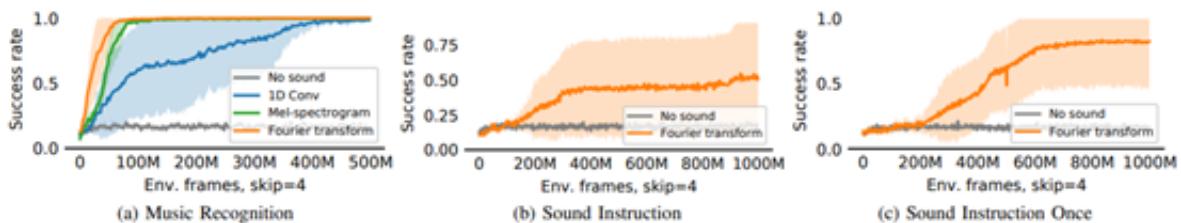


図 2.2 異なるエンコーダーの成功率 [2]

2.2 「DareFightingICE」の深層強化学習AI「BlindAI」

これまでにも、音を使ったゲームAIに注目した研究は数多く存在し、Nguyen, T等[3]は「DareFightingICE」に対し、音声入力のみの深層強化学習を用いる「BlindAI」を作った。「BlindAI」はGAInaとStephenson[9]のビデオゲームAIのフレームワークに基づいて、音だけを入力とするAIである。また、音声の特徴抽出はHegde, S等[2]の三つの音声解析システムを参照し、図2.3に示したように、「DareFightingICE」特有の音声解析エンコーダーで行う。

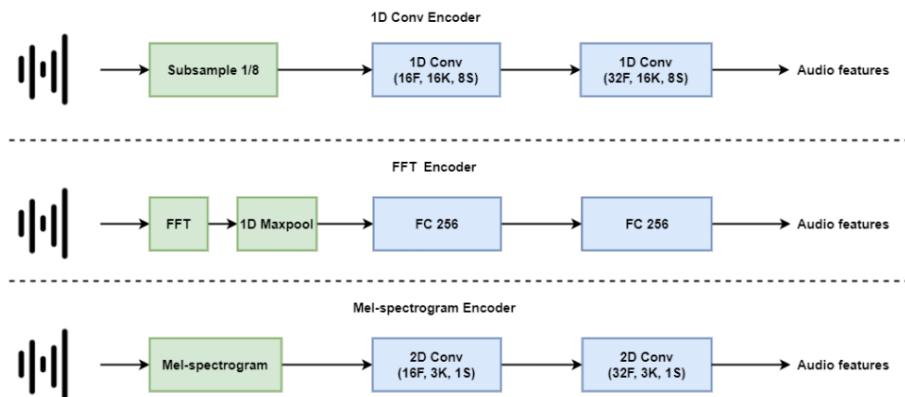


図2.3 「BlindAI」のエンコーダー構造図 [3]

次に、図2.4は強化学習モデルのネットワーク構造を示したものである。図2.3のエンコーダーのように、「DareFightingICE」からもたらされた生音声データを異なるエンコーダーを通して音声特徴データへ変更し、AIの深層強化学習モデルに与える。ネットワークの構造はGRU (Gated Recurrent Unit)を始め、3層の全接続層で組み合わせた。第一層と第二層の全接続層は256個のノードを設定した。第三層はAI操作するキャラクターのアクションリストの数により、40個のノードで構成した。

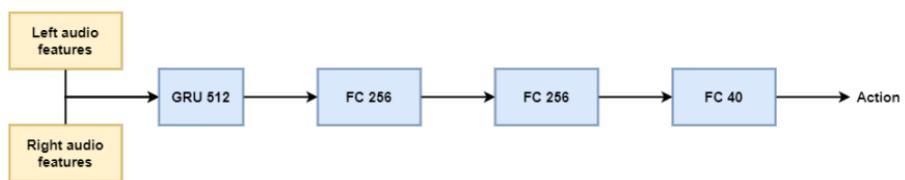


図2.4 「BlindAI」強化学習モデルのネットワーク構造 [3]

また、ネットワークのパラメータは PPO (Proximal Policy Optimization) ポリシーで更新し、ハイパーパラメーターが [10] を参照し、報酬の設定が [11] を設定した。そして、異なるエンコーダーで訓練したAIの対戦結果は表2.1のように示した。Mel-spectrogramのエンコーダーは1D-CNNとFFTのエンコーダーより強く、人以上(50%)の勝率になった。よって、Mel-spectrogramエンコーダーは確実に音声特徴を抽出でき、強化学習モデルに有用なメッセージを与えられる。

表2.1 異なるエンコーダーの対戦結果 [3]

Sound design	Encoder	<i>winratio</i>	<i>avgHPdiff</i>
DareFightingICE	1D-CNN	0.33	-28.83
DareFightingICE	FFT	0.37	-40.5
DareFightingICE	Mel-spectrogram	0.63	37.07

この三つのエンコーダーは Hegde, S 等 [2] のエンコーダーを改変してきたものである。結果から見ると一番よい Mel-spectrogram エンコーダーも 63 %の勝率しかないから、目の不自由な一般人 (MctsAI65) より少し強い AI に過ぎない。

今の時代が計算資源爆発の時代とも言え、研究用のパソコンはさらに高性能なものである。この三つのエンコーダーがかなり簡単な解析方法を用い、コンピュータの計算速度を完全発揮していない。従って、簡単な解析システムの代わりにより複雑な深層強化学習ネットワークで生データを解析する方が計算資源を発揮できるとともに、よりよく音声特徴を抽出することが可能になる。そのため、本研究は深層学習モデル Transformer や他のネットワークを組み立てて音声解析システムを構築した。

第3章

システムの提案

3.1 提案システムの概要

本システムは近年自然言語処理領域やコンピュータビジョンに活躍したニューラルネットワーク Transformer モデルを用い、全接続層と組み合わせて、音声解析システムを構築した。Nguyen, T 等 [3] の「BlindAI」の音声解析システム部分に代わり、本システムで音声データを渡し、訓練することにより、複雑なニューラルネットワークモデルが音声解析の有用性を検討する。

本システムで訓練してきた AI は「Blind-T」と命名した。

3.2 生データの前処理

「DareFightingICE」から提供される生音声データは n 個の正規化ベクトル s であり、 $s \in [-1,1]^n$ の形式である。一フレームごとの生データのサイズは左右 2 チャンネルでそれぞれ 800 個が存在するが、高速フーリエ変換をするため、各チャネルが 1024 サンプルになるようにゼロでパディングされた。自分が作っているエンコーダーはもらった 1024 個のデータのゼロを抜き、メッセージが持つ 800 個のデータのみ利用する。図 3.1 は左右チャネルを合わせて、ゲーム実行中の一フレームの波形を示したものである。

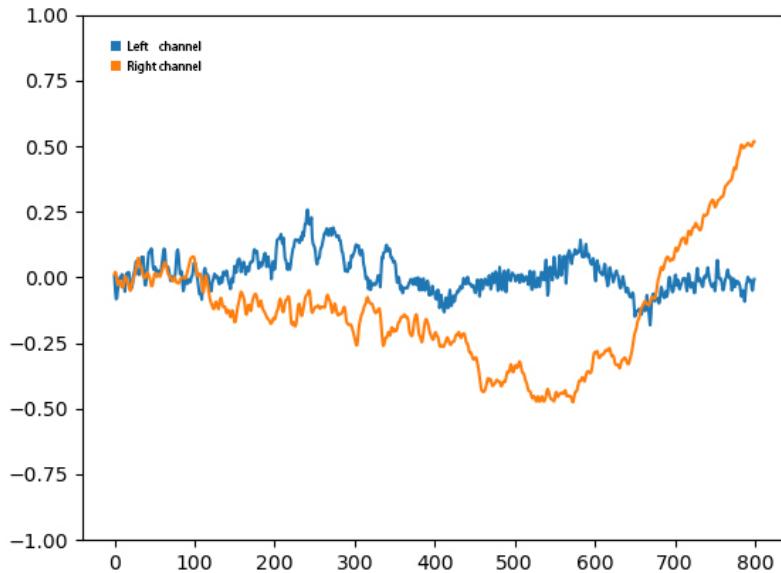


図 3.1 左右チャンネルの波形図

また、快速フーリエ変換を計算する際に、 2^n のデータ量なら計算効率が高くなるため、ゼロパディングが必要になる。図 3.2 はゼロパッティングの仕組みを示したものである。左の処理前の図のように、データ量は 800 であり、高速フーリエ変換に対して不効率であるため、ゼロパッティングを行う。方法はデータ量に合わせて、元のデータの後ろに総数 2^n まで 0 を追加するものである。また、データの前に 0 を追加する方法もある。従って、右の処理後の図のように、データの量は 800 から 1024 個になり、 2^n 個の総量になる。そして、高速フーリエ変換に導入し、効率よく計算できる。

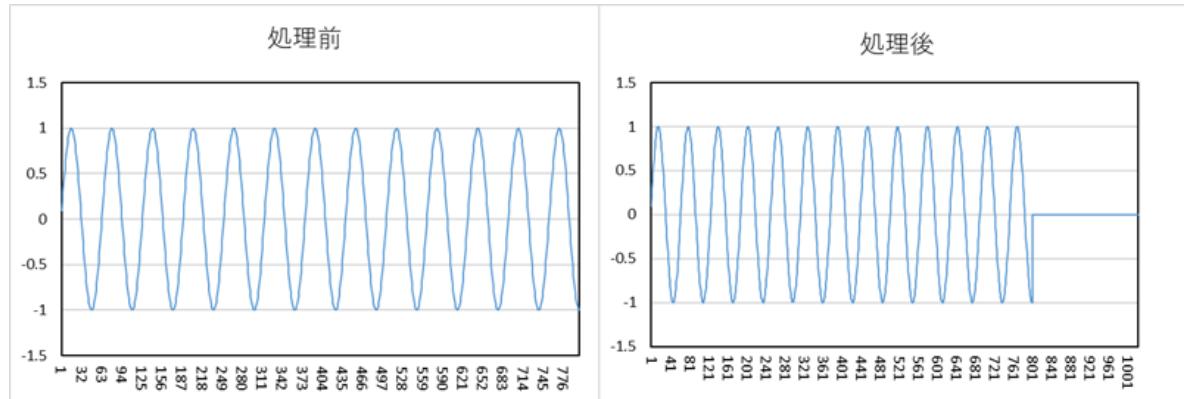


図 3.2 ゼロパッティングの仕組み

3.3 システムの詳細

これからは本システムの詳細を紹介し、システム内のパートごとを分解し、工夫する部分も細かく紹介を行う。

3.3.1 システムの構造

音声解析システム全体の構造図は図 3.3 のように示したものである。まず、生の音声データは左右二チャンネル計(800,2)のベクトルであり、チャンネルごとを分けて Transformer モデルのポジションコーディングを行い、Transformer モデルのエンコーダーに導入する。そして、Transformer モデルのエンコーダー部分は図 3.3 の右の図のように示したものである。エンコーダーを通したデータは(800,1,4)のベクトルであり、Flatten 層で 3200 個パラメータの 1 次元のベクトルに展開する。次に、全接続層を通して 256 個の特徴パラメータを出し、左右チャンネルのパラメータを合成した後、512 個の音声特徴データになる。活性化関数はすべて Relu 関数で活性化する。

Transformer モデルの利用は全体なモデルではなく、エンコーダー部分だけを利用し、「Attention」の仕組みを利用して時系列の音声データの間の関連性を探し、特徴抽出の役割を果たせる。

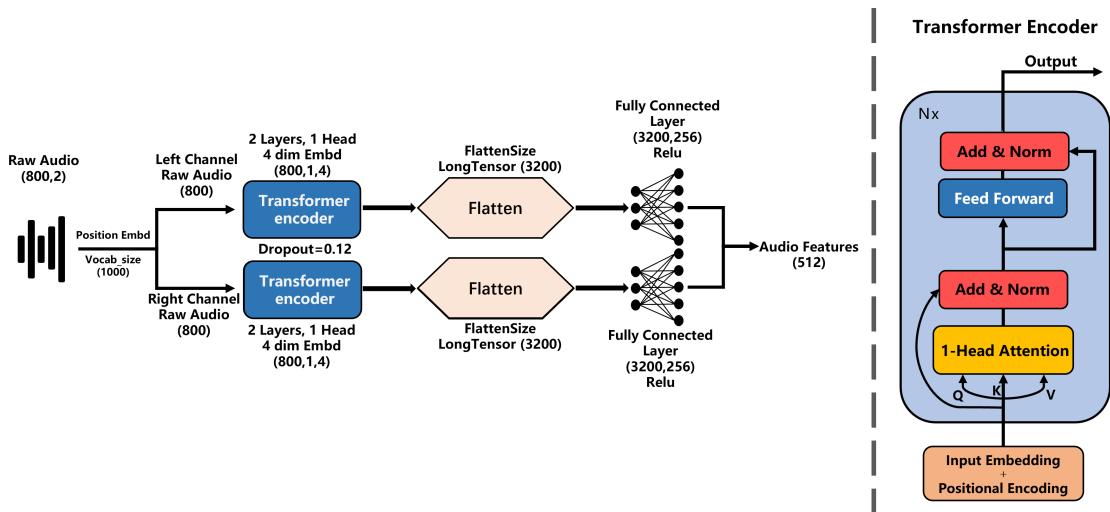


図 3.3 システム構造

3.3.2 Transformer 部分

本システムは Transformer モデルのエンコーダー部分を利用している。その構造は図 3.4 のように示したものである。このエンコーダー部分は Transformer モデルを提出する Vaswani, A.[4] 等のモデルを参照して作られたものである。提出者はグーグルのチームで、Python の Tensorflow の上で構築したものである。自分の「Blind-T」は Python の Pytorch で深層学習モデルを構築するため、ハーバード大学の自然言語処理チーム Rush, A[12] 等のプロジェクトを参考し、「DareFightingICE」に合わせて構築されたものである。次に、Transformer モデルの役割となぜ Transformer を利用して本システムを構築することを紹介する。

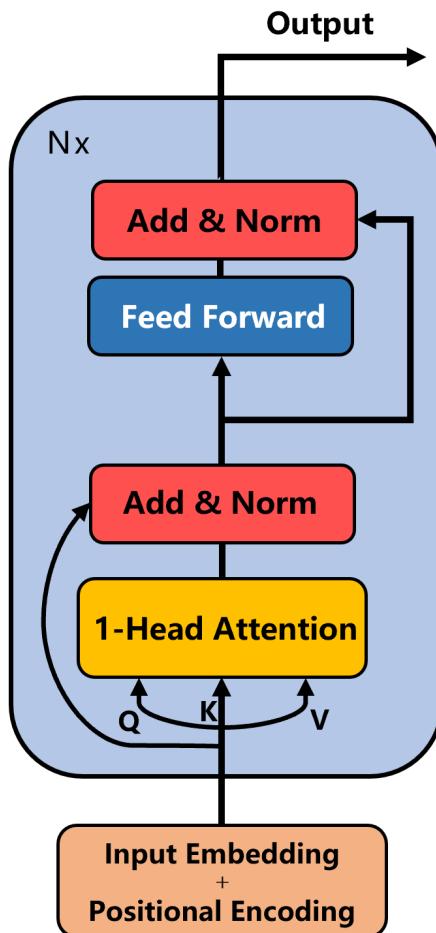


図 3.4 Transformer Encoder 構造

Attention と Transformer モデル

Transformer モデルは Vaswani, A.[4] 等で開発した機械翻訳任務に向かう新たなニューラルネットワーク構築である。その構造は図 3.5 を示したように、エンコーダーとデコーダで組み合わせたものである。また、「Attention」仕組みで言葉にある単語とすべての単語の関連性を計算できる仕組みである。それに、「Attention」により複雑な「Multi-Head-Attention」が組み立てるだけではなく、エンコーダー、デコーダの数も複数設定できる。これで計算資源と任務の難易度によりモデルの大きさを簡単に調整できる。

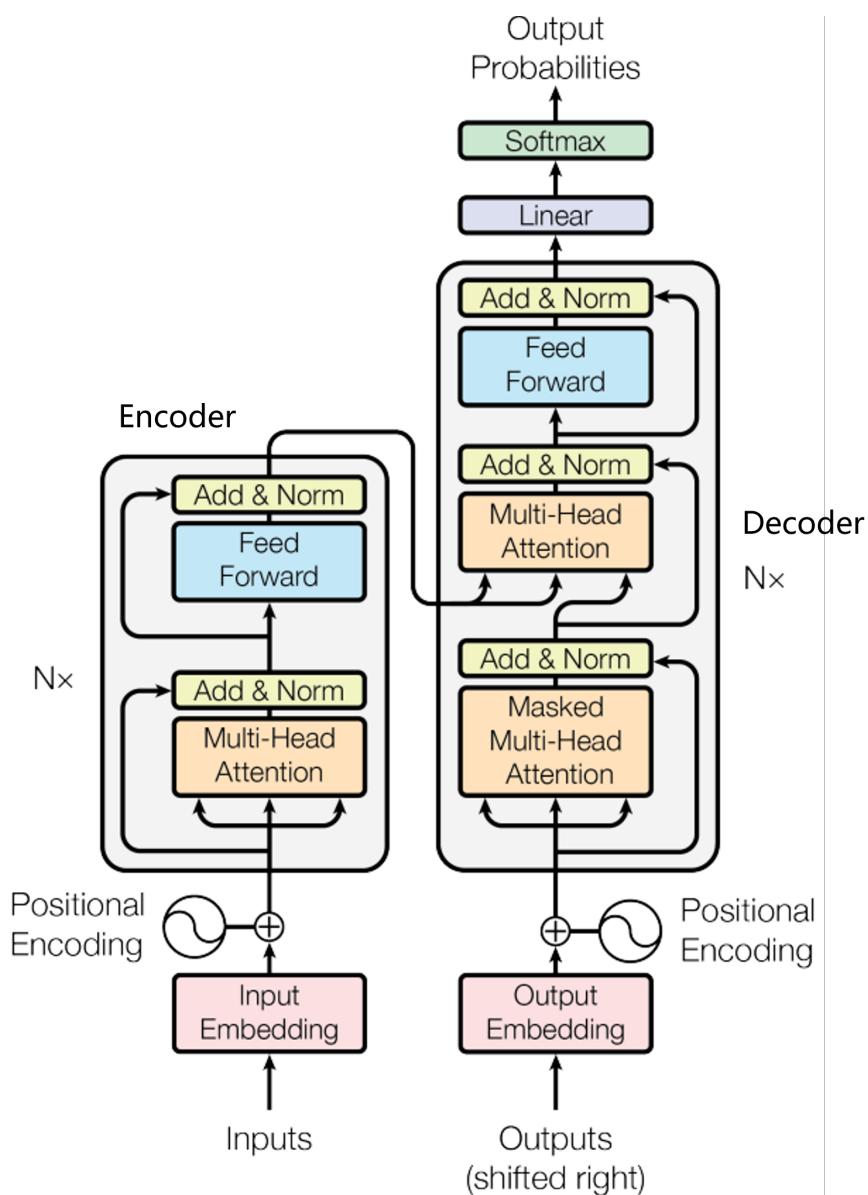


図 3.5 Transformer モデルの構造図 [4]

入力データはまずポジションコーディングで単語ごとの位置を決める。その公式は以下のように示す。

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

次に、「Attention」の計算公式は以下のように示した。式から見ると相関関数と似たような考え方で構造してきたものである。

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

NLP と ViT の応用

Transformer モデルを生みれた以来、機械翻訳を含む自然言語処理領域に活躍し、多くの任務に SOTA (state-of-the-art model) アルゴリズムとして支配している。図 3.5 は機械翻訳領域の各深層学習アルゴリズムの強さを示したものであり、その強さは BLEU (bilingual evaluation understudy) 点数で表せる。「Attention」仕組みを持つ Transformer モデルはないと比べて大きな差が存在し、言葉の長さに関わらず最尤なアルゴリズムである。

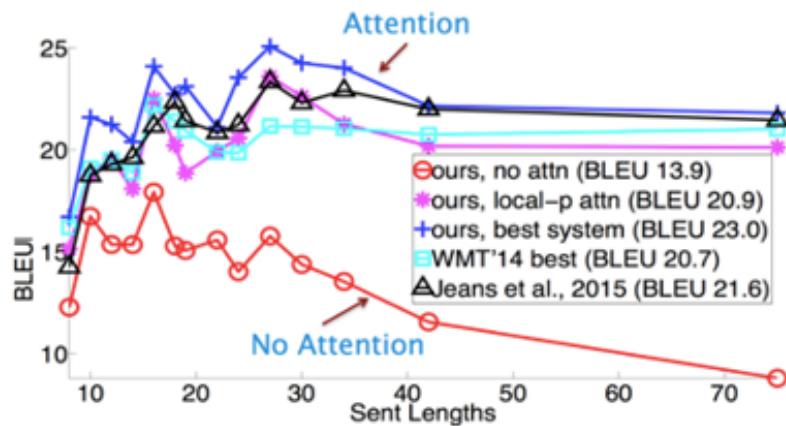


図 3.6 BLEU SOCRE[5]

また、近年の研究から見ると、自然言語処理領域だけではなく、コンピュータビジョン(CV)の領域にもTransformerモデルを出現し、畳み込みニューラルネットワークの支配位置を挑んでいる。Dosovitskiy, A. 等[6]はTransformerモデルをCVに合わせて、画像分類の任務に使うViT(Vision Transformer)を開発した。その構造は図3.7のように示した。

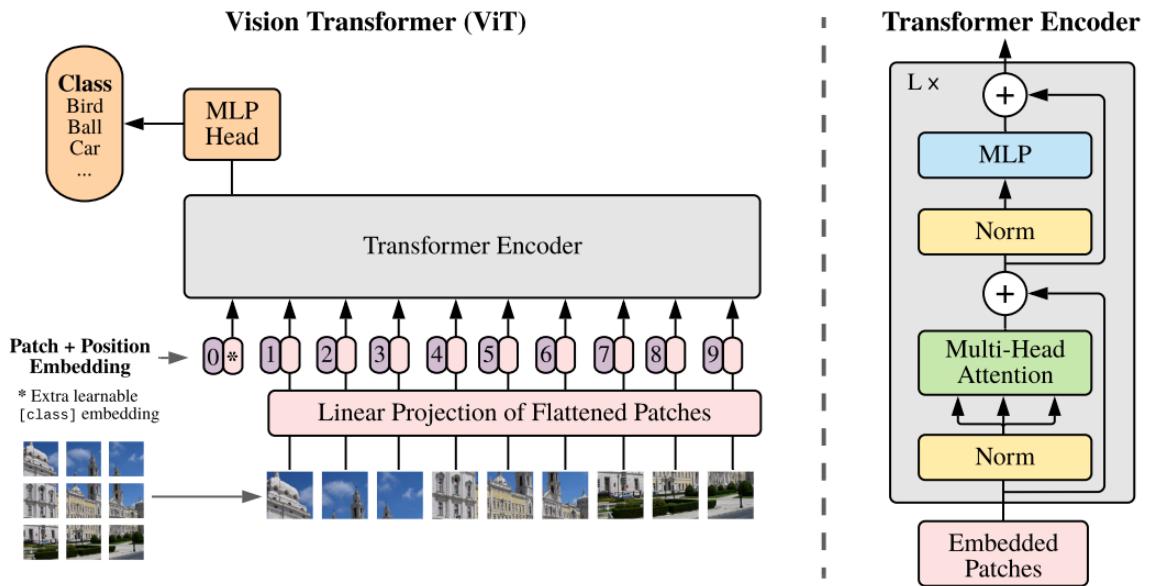


図3.7 ViT構造[6]

ViTの構造を分析すると、TransformerモデルのEncoder部分のみ利用し、画像ごとの特徴を抽出することとした。また、ViTは画像分類により結果を取るから、TransformerのEncoder部分は特徴抽出の有用性が存在することを考えられる。従って、TransformerモデルのEncoder部分は同様に音声の特徴を取られることを推測し、TransformerモデルのEncoderで「Blind-T」の音声解析システムを構築した。

3.4 異なるエンコーダーの計算コスト比較

提案システムは Transformer モデルを含む複数なニューラルネットワークで構成するため、計算コストが膨大と予測できる。表 3.1 は異なるエンコーダーが生音声データを 1000 回解析する時間を示したものである。

表 3.1 異なるエンコーダーの計算コスト

モデル	1000回 (秒)	平均1回
1d-cnn	0.244	0.00024
FFT	0.420	0.00041
Mel	0.722	0.00072
Transformer (Muti_head=2,d_model=32)	54.05	0.05400
Transformer (Muti_head=1,d_model=16)	27.91	0.04700

表 3.1 から見ると Transformer モデルは昔一番解析効果よい Mel-spectrogram の 74 倍以上のコストになり、モデルの軽量化へハイパーパラメーターを調整してもコストが半分になるしかない。従って、Transformer で組み立てた特徴解析システムの計算コストは元のものより激増し、高性能なパソコンしか遅延なく計算ができない。

3.5 ゲーム AI 構造と更新ポリシー

ゲーム AI 「Blind-T」の構造は図 3.8 のように示した。「Blind-T」は Encoder 部分が提案した音声解析システムであり、全体の構成が Nguyen, T 等 [3] の「BlindAI」を参考して作られたものである。

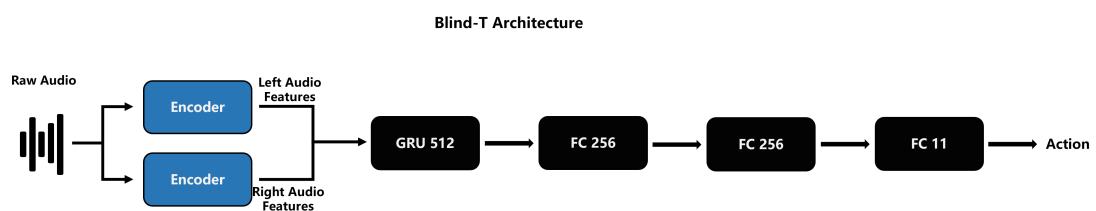


図 3.8 「Blind-T」アーキテクチャ

アクションの選択は生音声データがエンコーダーを通して音声特徴データになり、512個のパラメータの GRU モデルと二つの 256 個パラメータ持つ全接続層を通す。最後、全接続層 11 個（アクションリストの数）の出力によりアクションを選択する。

ネットワーク内のパラメータは PPO ポリシーで更新する。PPO ポリシーの報酬は [11] を参考し、以下の式のように実行した。

$$Reward = Reward_t^{offense} + Reward_t^{defense}$$

$$Reward_t^{offense} = HP_t^{opp} - HP_{t+1}^{opp}$$

$$Reward_t^{defense} = HP_{t+1}^{self} - HP_t^{self}$$

3.6 技リストとヒットボックス

表 3.2 は地上と空中に使えるすべての技とその詳細な属性を示すものである。技ごとはダメージと消耗するエネルギーが異なり、攻撃範囲またはヒットボックスも異なっている。

表 3.2 技リスト

- On Ground -								
Skill	Command	Damage	Attack Type	Special	StartUp	Active	Recovery	Energy
THROW_A	4_A	10	throw	-	5F	1F	24F	-5+2
THROW_B	4_B	20	throw	-	20F	1F	9F	-10+5
STAND_A	A	5	high	-	4F	3F	11F	0+2
STAND_B	B	10	high	-	5F	3F	15F	0+5
CROUCH_A	2_A	5	low	-	3F	3F	12F	0+3
CROUCH_B	2_B	10	low	-	11F	3F	6F	0+5
STAND_FA	6_A	8	high	-	5F	3F	28F	0+4
STAND_FB	6_B	12	middle	-	12F	3F	28F	0+10
CROUCH_FA	3_A	8	low	-	4F	3F	31F	0+4
CROUCH_FB	3_B	12	low	-	12F	3F	45F	0+10
STAND_D_DF_FA	2 3 6_A	5	high	projectile	20F	-	40F	-5+3
STAND_D_DF_FB	2 3 6_B	20	high	projectile	15F	-	45F	-30+10
STAND_D_DF_FC	2 3 6_C	120	high	projectile	15F	-	33F	-150+30
STAND_F_D_DFA	6 2 3_A	10	high	-	10F	20F	40F	0+5
STAND_F_D_DFB	6 2 3_B	40	low	-	10F	10F	40F	-55+20
STAND_D_DB_BA	2 1 4_A	10	middle	-	31F	5F	21F	0+5
STAND_D_DB_BB	2 1 4_B	25	middle	-	5F	10F	45F	-50+15

- In Air -								
AIR_A	A	8	middle	-	4F	4F	6F	0+5
AIR_B	B	10	middle	-	6F	9F	8F	0+10
AIR_DA	2_A	8	middle	-	5F	10F	33F	0+8
AIR_DB	2_B	10	middle	-	15F	10F	25F	0+10
AIR_FA	6_A	10	middle	-	11F	15F	16F	0+5
AIR_FB	6_B	12	middle	-	7F	13F	15F	0+10
AIR_UA	8_A	10	middle	-	5F	10F	33F	0+8

「BlindAI」は40個の技を選べ、その中に攻撃力が低くエネルギーも消耗するコスパが低い技が多くある。例えば、図3.9のような技はヒットボックスが小さく、攻撃力も低く技であり、コスパが低い。

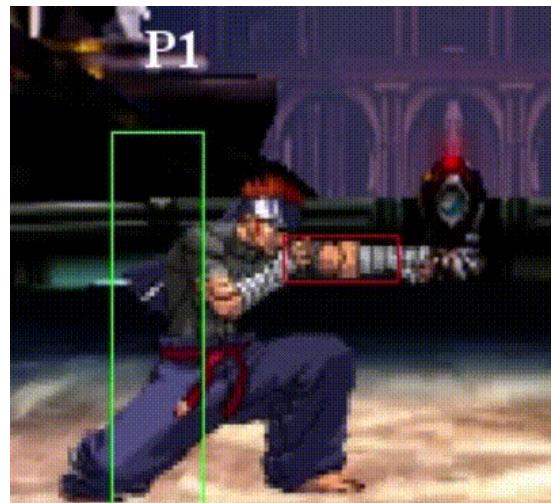


図3.9 技 $STAND_A$

逆に、図3.10のような技はヒットボックスが広く、エネルギーの消耗も少ない技であり、コスパが高いと考える。また、2022年チャンピオンAI「Sounder」は蹴ると波動拳二つの技だけで優勝になれる。その開発理念は防御を専念するとともに、ヒットボックスが長い技で相手にダメージを与える。よって、「Blind-T」はコスパが高い技しか残らない。



図3.10 技 $CROUCH_F B$

3.7 評価方式

対戦ゲームセット：	30 局 (90 Round)
対戦相手 1：	MctsAI65
対戦相手 2：	Sounder (2022 年チャンピオン AI)
結果評価：	勝率&平均 HP 差 (自分と相手の差)

まず、対戦の局数は Nguyen, T 等 [3] の「BlindAI」を評価する時に使う 30 ゲームセット、総計 90Round である。そして、盲人の基準「MctsAI65」と戦う。次に、2022 年のチャンピオン AI を挑み、結果は対戦の勝率と平均 HP の差で評価する。

第4章

実験

4.1 実験環境

- OS : Windows 10 professional
- ハードディスク
 - ◊ GPU : NVIDIA GeForce GTX 1080 8GBytes
 - ◊ CPU : Intel Core i7-6950X @ 3.00GHz
 - ◊ メモリ : 128 GBytes DDR4
 - ◊ マザーボード : ASUS X99-E WS
- ゲーム : DareFightingICE[13]
- CUDA 12.0.89
- CUDA Toolkit
- Conda の環境 [13]
 - ◊ Python 3.8.13
 - ◊ ライブライ
◦ environment.yml[13]
 - altair
 - spacy
 - GPUUtil
 - warnings

4.2 実験の流れ

トレーニング段階：

コードが完成した AI「Blind-T」をゲームに導入、「MctsAI65」より強い「MctsAI」と対戦し、2200Epoch 訓練した。1Epoch は 1 ゲームセット（3Round）のゲーム対戦情報を収集し、2 回パラメータを更新する仕組みである。

テスト段階：

「MctsAI65」と 30 ゲームセットを対戦し、結果分析する。次に、2022 年のチャンピオン AI「Sounder」と 30 ゲームセットを対戦し、結果分析する。

4.3 ハイパーパラメーター

4.3.1 技リスト (agent.py)

「Blind-T」は元の 40 種類の技から 11 種類の技まで圧縮し、コスパが高い技しか残らない。表 3.3 は「Blind-T」の技リストを示したものである。

表 4.1 「Blind-T」技リスト

技リスト			
STAND_D_DB_BA	STAND_D_DF_FC	BACK_STEP	FOR_JUMP
CROUCH_GUARD	AIR_GUARD	STAND_GUARD	CROUCH_B
CROUCH_FB	AIR_DB	AIR_FA	

4.3.2 モデル訓練 (train.py)

フィルタ「Train.py」のハイパーパラメーターは PPO の学習率などの値であり、隠れ層の出力数などを決める。表 4.2 はフィルタ「Train.py」のハイパーパラメーターを示したものである。

表 4.2 Train.py のハイパーパラメーター

Hyperparameter										
TRAINING_ITERATION	GAME_NUM	TRAIN_EPOCH	HIDDEN_SIZE	RECURRENT_LAYERS	LEARNING_RATE	GAMMA	C1	LAMBDA	ROLL_OUT	STATE_DIM: trans
2000	1	2	512	1	0.0002	0.99	0.95	0.95	3600	512

4.3.3 音声解析システム (エンコーダー)

音声解析システムは Transformer モデルと全接続層、Flatten 層などのネットワークから組み立てた深層学習モデルである。また、Transformer モデルの「Attention」の数とエンコーダーの数により、性能と計算コストが異なる。表 4.3 は音声解析システムのハイパーパラメーターである。

表 4.3 音声解析システムのハイパーパラメーター

Hyperparameter								
Transformer	vocab_size	d_model	dropout	max_len	mask	head	d_ff	Num of Encoder
	1000	4	0.12	800	ones(1,1,800)	1	32	1
FullyConnectedLayer	input	output						
	3200	256						

4.4 実験結果

まず、訓練結果は図 4.1 のように示した。横軸は Epoch の回数であり、縦軸は 1 ゲームセット (3Round) の報酬と示す。強い「MctsAI」と戦って訓練するから負ける場合が

多くあり、報酬はほぼ 0 以下になるが、Epoch の増加により、報酬も上昇することが明らかである。

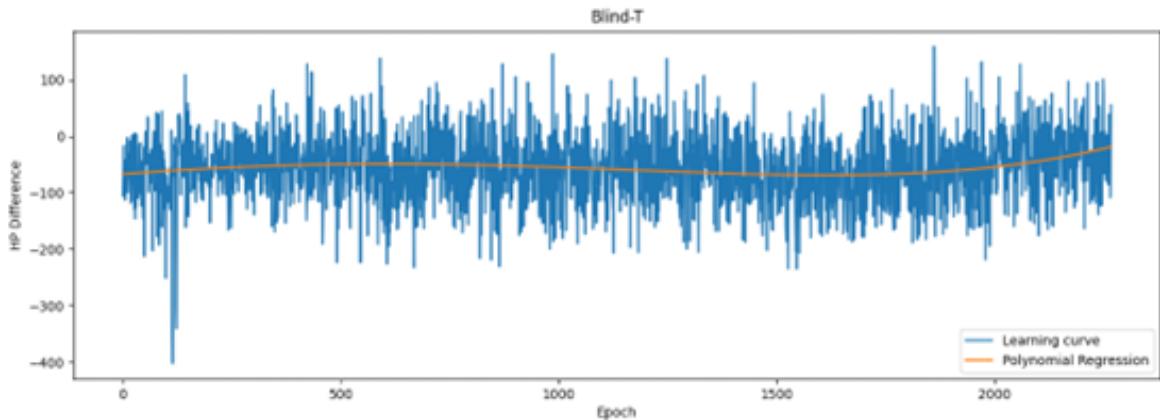


図 4.1 「Blind-T」学習曲線

報酬は自分と相手の HP の差を取り、1 ゲームセット（3Round）の平均値を計算する。公式は以下のように示した。

$$HP_{diff} = \frac{\Sigma(HP_{Blind-T} - HP_{MctsAI})}{3 \cdot GameSet}$$

従って、訓練した「Blind-T」がテスト階段のように「MctsAI65」と戦い、その対戦結果と BlindAI の対戦結果は表 4.4 のように示した。一番よい音声解析システムの「BlindAI」は Mel-spectrogram エンコーダーを使うものであり、その勝率は 63 % しか達していない。本研究の「Blind-T」は 96 % の勝率にもなり、平均報酬も 104.04 に達した。ほぼ完勝の成績で「MctsAI65」を勝った。

表 4.4 対戦結果 1 (30 GameSet)

Sound design	Encoder	win_{ratio}	$avgHP_{diff}$
DareFightingICE	1D-CNN	0.33	-28.83
DareFightingICE	FFT	0.37	-40.5
DareFightingICE	Mel-spectrogram	0.63	37.07
DareFightingICE	Blind-T	0.96	104.04

また, Nguyen, T 等 [3] は既に異なるエンコーダーが AI の性能を影響することを証明した。「Blind-T」は「Mel-spectrogram」エンコーダーで訓練した AI より強い。そのため、元のエンコーダーより Transformer モデルで組み立てた深層学習モデルの方は対戦格闘ゲームの音声特徴をよりよく抽出できることを証明できた。

それに、2022 年のチャンピオン AI 「Sounder」と対戦し、その結果は表 4.5 のように示した。「Blind-T」の勝率は 54 % になり、平均値報酬が 16.13 になる。「Sounder」より勝率が高いが、大会の時なら必ず勝てるとは言えない。

表 4.5 対戦結果 1 (30 GameSet)

Enemy	Encoder	win _{ratio}	avgHP _{diff}
Mcts65	Blind-T	0.96	104.04
Sounder	Blind-T	0.53	16.13

第5章

おわりに

5.1まとめ

BGM や効果音は現代のビデオゲームの没入感に重要な役割を果たしている。音で任務の手がかりを提示するなど様々な効果を発揮している。また、対戦格闘ゲーム「FightingICE」は機械学習の研究として使われ、その拡張版の「DareFightingICE」はさらに IEEE CoG(IEEE Conference on Games) の公式試合舞台として使われている。「DareFightingICE」は音声のみの入力とする対戦 AI の開発も課題となっていた。

本研究では Transformer モデルが元の Mel-spectrogram や FFT で作られた音声解析システムより音声特徴抽出機能が向上することを検討することである。

提案手法として、Transformer から組み立てる深層学習モデルで音声解析システムを構築し、元の音声解析システムで作った AI と同じ構造のゲーム AI 「Blind-T」を作る。その後、同じ敵と戦い、勝率が高い方は音声特徴がよりよく抽出したことを認める。実際、トレーニング済みの AI は盲人実力標準に近い「MctsAI65」と対戦し、勝率でどのシステムがよりよく音声特徴を抽出できることを判断する。また、異なる解析システムで 1000 回実行の時間を測り、その計算コストを比較する。

結果として、「Blind-T」は元の「BlindAI」の 63 % の勝率より高く超え、「MctsAI65」との対戦に 96 % の勝率を得た。それに、2022 年のチャンピオン AI 「Sounder」にも勝ち、54 % の勝率を取った。そのため、Transformer モデルから組み立てた深層学習モデルは以前の方法と比べて、計算資源が多く消耗するが、よりよく音声特徴を抽出するも確実である。

5.2 プロジェクトのアクセス先

<https://github.com/ChaoxJz/FightingICE/tree/master/DareFightingICE/SampleAI/BlindAI>

5.3 今後の展望

現在の「Blind-T」は音声解析の部分に Transformer モデルを使っている。実際、AI 全体の構造図から見ると、解析したパラメータは GRU モデルに導入することが分かる。Transformer も GRU も時系列問題によく使うモデルであり、Transformer はさらに全面的に特徴を抽出できるものである。そのため、GRU の部分も Transformer に変更し、「Blind-T」をより強い方向で進化することが可能である。

謝辞

本論文の執筆にあたり、日頃から研究の進み具合を気にかけていただき、終始暖かく見守って下さった THAWONMAS Ruck 教授に深く感謝いたします。プロジェクトのコードに途方に暮れる私に的確な助言と指導をくださった Thai Van Nguyen 先輩には、感謝の念が絶えません。ゼミ時間外にも何度もミーティングを設けていただき、ご指導と励ましをいただきました。そして、同期の TaoLiyongくんと我谷英俊くんを始め、知能エンターテインメント研究室の同窓生の皆さんには常に刺激的な議論を頂き、精神的にも支えられました。本当にありがとうございました。

参考文献

- [1] FightingGame AI Competition Homepage, 2022.12.3 Accessed.
<https://www.ice.ci.ritsumei.ac.jp/~ftgaic/index-1.html>.
- [2] Hegde,S.,Kanervisto,A., Petrenko,A..(2021).Agents that listen: high-throughput reinforcement learning with multiple sensory systems.
- [3] Nguyen,T.V.,Dai,X.,Khan,I.,Thawonmas,R., Pham,H.V..(2022). A Deep Reinforcement Learning Blind AI in DareFightingICE.
- [4] Vaswani,A.,Shazeer,N.,Parmar,N.,Uszkoreit,J.,Jones,L., Gomez,A.N.,et al. (2017). Attention is all you need. arXiv.
- [5] CS224n: Natural Language Processing with Deep Learning1. , 2023.1.9 Accessed.
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/lecture_notes/cs224n-2017-notes6.pdf.
- [6] Dosovitskiy,A.,Beyer,L.,Kolesnikov,A.,Weissenborn,D., Houlsby,N..(2020). An image is worth 16x16 words: transformers for image recognition at scale.
- [7] Khan,I.,Nguyen,T.V.,Dai,X.,Thawonmas,R..(2022). Darefightingice competition: a fighting game sound design and ai competition. arXiv e-prints.
- [8] HEROZ AI 「DareFightingICE:AI Track」 優勝報告, 2023.1.3 Accessed.
https://heroz.co.jp/info/2022/08/24_heroz01.
- [9] R.D.Gaina and M.Stephenson, ““Did You Hear That?” Learning to Play Video Games from Audio Cues,” in Proceedings of the 2019 IEEE Conference on Games (CoG), pp. 1-4, 2019.
- [10] Y.Takano,W.Ouyang,S.Ito,T.Harada and R.Thawonmas, “Applying Hybrid Reward Architecture to a Fighting Game AI,” in Proceedings of IEEE Conference on Computational Intelligence and Games (CIG), pp. 433-436, August 2018.
- [11] K.Cho,B.Van Merriënboer,C.Gulcehre,D.Bahdanau,F.Bougares,H.Schwenk, and Y.Bengio, “Learning phrase representations using rnn encoder-decoder for statis-

- tical machine translation,” in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1724–1734, 2014.
- [12] Rush,A.. ”The Annotated Transformer.” Proceedings of Workshop for NLP Open Source Software (NLP-OSS) 2018.
- [13] DareFightingICE Project Transformer Encoder -GitHub, 2022.1.27 Accessed.
<https://github.com/ChaoxJz/FightingICE>.

付録

付録.A 提案システムのソースコード

付録.A-1 encoder.py

```
1 import torch
2 import math
3 from torch.autograd import Variable
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import copy
7 import torch.nn.functional as F
8 import os
9 from os.path import exists
10 import math
11 import time
12 from torch.optim.lr_scheduler import LambdaLR
13 import pandas as pd
14 import altair as alt
15 import spacy
16 import GPUUtil
17 import warnings
18 from torch.utils.data.distributed import
19     DistributedSampler
20 import torch.distributed as dist
21 import torch.multiprocessing as mp
22 from torch.nn.parallel import DistributedDataParallel as
23     DDP
24 from abc import ABC, abstractmethod
25 from torch import nn
```

```
24 import torchaudio
25
26 class BaseEncoder(nn.Module, ABC):
27     def __init__(self, sampling_rate=48000, fps=60,
28                  frame_skip=4):
29         super(BaseEncoder, self).__init__()
30         self.sampling_rate = sampling_rate
31         self.FPS = fps
32         self.frame_skip = frame_skip
33
34     def forward(self, x):
35         # left side
36         left = x[:, :, 0]
37         left = self.encode_single_channel(left)
38         # right side
39         right = x[:, :, 1]
40         right = self.encode_single_channel(right)
41         return torch.cat((left, right), dim=1)
42
43     @abstractmethod
44     def encode_single_channel(self, data):
45         pass
46
47     class SampleEncoder(BaseEncoder):
48         def encode_single_channel(self, data):
49             return data
50
51     class RawEncoder(BaseEncoder):
52         def __init__(self, sampling_rate=48000, fps=60,
53                      frame_skip=4):
54             super(RawEncoder, self).__init__(sampling_rate,
55                                             fps, frame_skip)
55             self.num_to_subsample = 8
56             self.num_samples = (self.sampling_rate / self.FPS
57                               ) * self.frame_skip
58             assert int(self.num_samples) == self.num_samples
```

```
57     # Encoder (small 1D conv)
58     self.pool = torch.nn.MaxPool1d(2)
59     self.conv1 = torch.nn.Conv1d(1, 16, kernel_size
60                               =16, stride=8)
61     self.conv2 = torch.nn.Conv1d(16, 32, kernel_size
62                               =16, stride=8)
63
64     def encode_single_channel(self, data):
65         """Shape of x: [batch_size, num_samples]"""
66         # Subsample
67         x = data[:, ::self.num_to_subsample]
68
69         # Add channel dimension
70         x = x[:, None, :]
71         x = F.relu(self.conv1(x))
72         x = self.pool(x)
73         if x.shape[2] >= 24:
74             x = self.conv2(x)
75             x = self.pool(x)
76         return x
77
78 class NewEncoder(BaseEncoder):
79     def __init__(self, sampling_rate=48000, fps=60,
80                  frame_skip=4):
81         super(NewEncoder, self).__init__(sampling_rate,
82                                         fps, frame_skip)
83         self.num_to_subsample = 8
84         self.num_samples = (self.sampling_rate / self.FPS
85                            ) * self.frame_skip
86         self.num_frequencies = self.num_samples / 2
87         assert int(self.num_samples) == self.num_samples
88         self.num_samples = int(self.num_samples)
89         self.num_frequencies = int(self.num_frequencies)
90
91         # Encoder (small MLP)
92         # self.linear1 = torch.nn.Linear(int(self.
93                                         num_frequencies / self.num_to_subsample), 256)
```

```
88         dim = 800 * frame_skip
89         self.linear1 = torch.nn.Linear(dim, 256)
90         self.linear2 = torch.nn.Linear(256, 256)
91         self.flatten = torch.nn.Flatten()
92
93     def encode_single_channel(self, data):
94         #raw_audio = np.random.rand(1,800,2)
95
96         # Add and remove "channel" dim...
97         x = F.relu(self.linear1(self.flatten(data)))
98         x = F.relu(self.linear2(x))
99         return x
100
101
102
103 class MelSpecEncoder(BaseEncoder):
104     def __init__(self, sampling_rate=48000, fps=60,
105                  frame_skip=4):
106         super(MelSpecEncoder, self).__init__(
107               sampling_rate, fps, frame_skip)
108         self.window_size = int(self.sampling_rate *
109                               0.025)
110         self.hop_size = int(self.sampling_rate * 0.01)
111         self.n_fft = int(self.sampling_rate * 0.025)
112         self.n_mels = 80
113
114         self.mel_spectrogram = torchaudio.transforms.
115                               MelSpectrogram(
116                                 sample_rate=self.sampling_rate,
117                                 n_mels=80,
118                                 n_fft=self.n_fft,
119                                 win_length=self.window_size,
120                                 hop_length=self.hop_size,
121                                 f_min=20,
122                                 f_max=7600,
123                                 )
124
```



```
151
152     # Encoder (small MLP)
153     self.linear1 = torch.nn.Linear(int(self.
154         num_frequencies / self.num_to_subsample), 256)
155     self.linear2 = torch.nn.Linear(256, 256)
156
157     def _torch_1d_fft_magnitude(self, x):
158         """Perform 1D FFT on x with shape (batch_size,
159             num_samples), and return magnitudes"""
160         # Apply hamming window
161         if x.device != self.hamming_window.device:
162             self.hamming_window = self.hamming_window.to(
163                 x.device)
164         x = x * self.hamming_window
165         # Add zero imaginary parts
166         x = torch.stack((x, torch.zeros_like(x)), dim=-1)
167         c = torch.view_as_complex(x)
168         fffts = torch.fft.fft(c)
169         fffts = torch.view_as_real(fffsts)
170         # Remove mirrored part
171         fffts = fffts[:, :(fffsts.shape[1] // 2), :]
172         # To magnitudes
173         mags = torch.sqrt(fffsts[..., 0] ** 2 + fffsts[...,
174             1] ** 2)
175
176         return mags
177
178     def encode_single_channel(self, data):
179         """Shape of x: [batch_size, num_samples]"""
180         #data=data.to("cuda:0")
181         mags = self._torch_1d_fft_magnitude(data)
182         mags = torch.log(mags + 1e-5)
183
184         # Add and remove "channel" dim...
185         x = self.pool(mags[:, None, :])[:, 0, :]
186         x = F.relu(self.linear1(x))
187         x = F.relu(self.linear2(x))
188         #print(x.device)
```

```
184         return x
185
186     """
187
188 Transformer_encoder_model_class&function
189
190 """
191
192 class Embeddings(nn.Module):
193     def __init__(self, d_model, vocab):
194         super(Embeddings, self).__init__()
195         self.lut = nn.Embedding(vocab, d_model)
196         self.d_model = d_model
197
198     def forward(self, x):
199         return self.lut(x) * math.sqrt(self.d_model)
200
201
202 class PositionalEncoding(nn.Module):
203     def __init__(self, d_model, dropout, max_len=5000):
204         super(PositionalEncoding, self).__init__()
205         self.dropout = nn.Dropout(p=dropout)
206         pe = torch.zeros(max_len, d_model)
207         position = torch.arange(0, max_len).unsqueeze(1)
208         div_term = torch.exp(torch.arange(0, d_model, 2) *
209                             -(math.log(10000.0) /
210                               d_model))
211         pe[:, 0::2] = torch.sin(position * div_term)
212         pe[:, 1::2] = torch.cos(position * div_term)
213         pe = pe.unsqueeze(0)
214         self.register_buffer('pe', pe)
215
216     def forward(self, x):
217         x = x + self.pe[:, : x.size(1)].requires_grad_(
218             False)
219         return self.dropout(x)
```

```
218
219
220 class MultiHeadedAttention(nn.Module):
221     def __init__(self, head, d_model, dropout=0.1):
222         super(MultiHeadedAttention, self).__init__()
223         assert d_model % head == 0
224         self.d_k = d_model // head
225         self.head = head
226         self.linears = clones(nn.Linear(d_model, d_model)
227                             , 4)
228         self.attn = None
229         self.dropout = nn.Dropout(p=dropout)
230
231     def forward(self, query, key, value, mask=None):
232         if mask is not None:
233             mask = mask.unsqueeze(1)
234             nbatches = query.size(0)
235             query, key, value = [
236                 lin(x).view(nbatches, -1, self.head, self.d_k
237                             ).transpose(1, 2)
238                 for lin, x in zip(self.linears, (query, key,
239                                             value))]
240
241         x, self.attn = attention(query, key, value, mask=
242             mask, dropout=self.dropout)
243         x = x.transpose(1, 2).contiguous().view(nbatches,
244                                         -1, self.head * self.d_k)
245         del query
246         del key
247         del value
248         return self.linears[-1](x)
249
250
251     class PositionwiseFeedForward(nn.Module):
252         def __init__(self, d_model, d_ff, dropout=0.1):
253             super(PositionwiseFeedForward, self).__init__()
254             self.w1 = nn.Linear(d_model, d_ff)
255             self.w2 = nn.Linear(d_ff, d_model)
```

```
250         self.dropout = nn.Dropout(dropout)
251
252     def forward(self, x):
253         return self.w2(self.dropout(F.relu(self.w1(x))))
254
255 class LayerNorm(nn.Module):
256     def __init__(self, features, eps=1e-6):
257         super(LayerNorm, self).__init__()
258         self.a_2 = nn.Parameter(torch.ones(features))
259         self.b_2 = nn.Parameter(torch.zeros(features))
260         self.eps = eps
261
262     def forward(self, x):
263         mean = x.mean(-1, keepdim=True)
264         std = x.std(-1, keepdim=True)
265         return self.a_2 * (x - mean) / (std + self.eps) +
266             self.b_2
267
268 class SublayerConnection(nn.Module):
269     def __init__(self, size, dropout=0.1):
270         super(SublayerConnection, self).__init__()
271         self.norm = LayerNorm(size)
272         self.dropout = nn.Dropout(p=dropout)
273
274     def forward(self, x, sublayer):
275         return x + self.dropout(sublayer(self.norm(x)))
276
277 class EncoderLayer(nn.Module):
278     def __init__(self, size, self_attn, feed_forward,
279                  dropout):
280         super(EncoderLayer, self).__init__()
281         self.self_attn = self_attn
282         self.feed_forward = feed_forward
283         self.sublayer = clones(SublayerConnection(size,
284                                              dropout), 2)
285         self.size = size
```

```
284     def forward(self, x, mask):
285         x = self.sublayer[0](x, lambda x: self.self_attn(
286             x, x, x, mask))
287         return self.sublayer[1](x, self.feed_forward)
288
289 class Encoder(nn.Module):
290     def __init__(self, layer, N):
291         super(Encoder, self).__init__()
292         self.layers = clones(layer, N)
293         self.norm = LayerNorm(layer.size)
294
295     def forward(self, x, mask):
296         for layer in self.layers:
297             x = layer(x, mask)
298         return self.norm(x)
299
300 def subsequent_mask(size):
301     attn_shape = (1, size, size)
302     subsequent_mask = torch.triu(torch.ones(attn_shape),
303                                 diagonal=1).type(torch.uint8)
304     return subsequent_mask == 0
305
306 def attention(query, key, value, mask=None, dropout=None):
307     d_k = query.size(-1)
308     scores = torch.matmul(query, key.transpose(-2, -1)) /
309             math.sqrt(d_k)
310     if mask is not None:
311         scores = scores.masked_fill(mask == 0, -1e9)
312     p_attn = F.softmax(scores, dim = -1)
313     if dropout is not None:
314         p_attn = dropout(p_attn)
315     return torch.matmul(p_attn, value), p_attn
316
317 def clones(module, N):
```

```
317     return nn.ModuleList([copy.deepcopy(module) for _ in
318                           range(N)])
319
320 class TransformerEncoder(BaseEncoder):
321     def __init__(self, sampling_rate=48000, fps=60,
322                  frame_skip=4):
323         super(TransformerEncoder, self).__init__(
324             sampling_rate, fps, frame_skip)
325
326         # add some transformer stuffs here
327         #self.input_data = torch.zeros((1,800,2),dtype=
328                                         torch.long) #vocab_size=1*800*2 + 1
329         #initialization of input_data let vocab_size to
330             # know numbers of data.
331         # size is 1000/Tensor+1
332         #self.vocab_size = self.input_data.numel()+1
333         self.vocab_size = 1000
334         self.d_model = 4
335         self.dropout = 0.12
336         self.max_len=800
337         self.mask = torch.ones(1, 1, self.max_len)
338         #self.mask=self.mask.to("cuda:0")
339         self.head = 1
340         #Neuron
341         self.d_ff = 32
342         #Deepcoopy
343         self.c = copy.deepcopy
344         self.embedding = Embeddings(self.d_model, self.
345                                     vocab_size)
346         self.pe = PositionalEncoding(self.d_model, self.
347                                     dropout)
348         self.attn = MultiHeadedAttention(self.head, self.
349                                         d_model)
350         self.ff = PositionwiseFeedForward(self.d_model,
351                                         self.d_ff, self.dropout)
352         self.layer = EncoderLayer(self.d_model, self.c(
```

```
            self.attn), self.c(self.ff), self.dropout)
345    self.N = 1
346    self.Enc = Encoder(self.layer, self.N)
347
348
349    #finished the transformer_encoder
350    #pass to 2-linear
351    self.flatten = torch.nn.Flatten()
352    # Encoder_linear
353    self.linear2 = torch.nn.Linear(3200, 256)
354
355    def encode_single_channel(self, data):
356        # add some transformer stuffs here
357        #data=torch.Tensor(data)
358
359        self.mask = self.mask.to(data.device)
360        #self.mask=self.mask.to("cuda:0")
361        #print(self.mask.device)
362        input_data=(100*((data+2)/3)).long()
363        #input_data=input_data.to("cuda:0")
364        #input_data=torch.from_numpy(100*((data+2)/3)).
365        #print(input_data)
366        #token index (0~1500)
367        #audio_data is float(-1,1) ->(0,2)---> (0,200).
368        longtensor data
369
370        #Embedding
371        x=self.embedding(input_data)
372
373        #Positional Coding
374        x = self.pe(x)
375
376        x = F.relu(self.linear2(self.flatten(self.Enc(x,
377            self.mask))))
378        #x = F.relu(self.Enc(x,self.mask))
379        #Transformer_Model_Encoder_data=self.Enc(x,mask)
```

```
378     #finished the transformer_encoder
379     #pass to 2-linear
380     #Transformer_Model_Encoder_data = torch.flatten(
381         Transformer_Model_Encoder_data)
382     #Transformer_Model_Encoder_data = F.relu(self.
383         linear1(Transformer_Model_Encoder_data))
384     #print(x.device)
385
386     return x
387 if __name__ == '__main__':
388     # TODO: compare data value vs pytorch version
389     encoder = TransformerEncoder(frame_skip=1)
390     np.random.seed(0)
391     # 1*3200*2
392     #raw_audio = np.random.rand(1,800,2)
393     data = np.random.rand(1,800,2)
394     #a=encoder(data)
395     print(encoder(data).shape)
```

付録.B ゲームA I構造のソースコード

付録.B-1 agent.py

```
1 # game agent (implements aiinterface.AIInterface)
2 # TODO check the actions from RHEA_PPO bot (currently 56
   by default)
3 # TODO collect data
4 import time
5
6 import numpy as np
7
8 import torch
9
10 GATHER_DEVICE = 'cpu'
11
12 import logging
13
14 #updata 12.10.20.17
15
16 class SoundAgent:
17     def __init__(self, gateway, **kwargs):
18         self.gateway = gateway
19         self.actor = kwargs.get('actor')
20         self.critic = kwargs.get('critic')
21         self.device = kwargs.get('device')
22         self.logger = kwargs.get('logger')
23         self.collect_data_helper = kwargs.get(
24             'collect_data_helper')
25         self.rnn = kwargs.get('rnn')
26         self.trajectories_data = None
27
28         '',
29         self.actions = "STAND_D_DB_BA", "STAND_D_DF_FC", "
   STAND_D_DF_FC", \
                           "BACK_STEP", "FOR_JUMP", "
```

```
30             "CROUCH_GUARD", "AIR_GUARD", "  
31             STAND_GUARD", \  
32             "CROUCH_GUARD", "AIR_GUARD", "  
33             STAND_GUARD"\  
34             "CROUCH_B", "CROUCH_B", "CROUCH_FB  
35             ", "CROUCH_FB", "CROUCH_FB", "  
36             AIR_DB", "AIR_FA", \  
37             "CROUCH_GUARD", "AIR_GUARD", "  
38             STAND_GUARD"  
39             ""  
40             self.actions = "STAND_D_DB_BA", "STAND_D_DF_FC", \  
41             "BACK_STEP", "FOR_JUMP", "  
42             CROUCHGUARD", "AIR_GUARD", "  
43             STAND_GUARD", \  
44             "CROUCHB", "CROUCHFB", "AIR_DB", "  
45             AIR_FA"  
46             self.audio_data = None  
47             self.raw_audio_memory = None  
48             self.just_initiated = True  
49             self.pre_framedata = None  
50             self.nonDelay = None  
51  
52             # data of 1 rounds  
53             # self.round_state_list = []  
54             # self.round_value_list = []  
55             # self.round_action_list = []  
56             # self.round_action_probs_list = []  
57             # self.round_true_reward_list = []  
58             # self.round_reward_list = []  
59             # self.round_terminal_list = []  
60             # self.round_actor_hidden_state_list = []  
61             # self.round_actor_cell_state_list = []  
62             # self.round_critic_hidden_state_list = []  
63             # self.round_critic_cell_state_list = []
```

```
58      # data over all rounds
59      # self.state_list = []
60      # self.value_list = []
61      # self.action_list = []
62      # self.action_probs_list = []
63      # self.true_reward_list = []
64      # self.reward_list = []
65      # self.terminal_list = []
66      # self.actor_hidden_state_list = []
67      # self.actor_cell_state_list = []
68      # self.critic_hidden_state_list = []
69      # self.critic_cell_state_list = []
70      # self.episode_lengths = []
71      if self.rnn:
72          self.actor.get_init_state(GATHER_DEVICE)
73          self.critic.get_init_state(GATHER_DEVICE)
74      self.round_count = 0
75      self.n_frame = kwargs.get('n_frame')
76      # self.collect_data_helper = CollectDataHelper()
77
78      def initialize(self, gameData, player):
79          # Initializng the command center, the simulator
80          # and some other things
81          self.inputKey = self.gateway.jvm.struct.Key()
82          self.frameData = self.gateway.jvm.struct.
83              FrameData()
84          self.cc = self.gateway.jvm.aiinterface.
85              CommandCenter()
86          self.player = player # p1 == True, p2 == False
87          self.gameData = gameData
88          self.simulator = self.gameData.getSimulator()
89          self.isGameJustStarted = True
90          return 0
91
92      def close(self):
93          pass
```

```
92     def getInformation(self, frameData, inControl,
93                         nonDelay):
94         # Load the frame data every time getInformation
95         # gets called
96         self.frameData = frameData
97         self.cc.setFrameData(self.frameData, self.player)
98         # nonDelay = self.frameData
99         self.pre_framedata = self.nonDelay if self.
100            nonDelay is not None else nonDelay
101         self.nonDelay = nonDelay
102         self.isControl = inControl
103         self.currentFrameNum = nonDelay.getFramesNumber()
104             # first frame is 14
105
106         def roundEnd(self, x, y, z):
107             self.logger.info(x)
108             self.logger.info(y)
109             self.logger.info(z)
110             self.just_initiated = True
111             obs = self.raw_audio_memory
112             if obs is not None:
113                 self.collect_data_helper.put([obs, 0, True,
114                     None])
115             # final step
116             # state = torch.tensor(obs, dtype=torch.float32)
117             terminal = 1
118             true_reward = self.get_reward()
119             self.collect_data_helper.finish_round()
120             self.raw_audio_memory = None
121             self.round_count += 1
122             self.logger.info('Finished {} round'.format(self.
123                 round_count))
124
125             # please define this method when you use FightingICE
126             # version 4.00 or later
127             def getScreenData(self, sd):
128                 pass
```

```
122      # start_time = time.time()
123      # data = sd.getDisplayByteBufferAsBytes()
124      # # tmp = np.frombuffer(data)
125      # end_time = time.time()
126      # print('screen', (end_time - start_time) * 1000)
127
128      def input(self):
129          return self.inputKey
130
131      @torch.no_grad()
132      def processing(self):
133          # start_time = time.time()
134          # First we check whether we are at the end of the
135          # round
136          start_time = time.time() * 1000
137          if self.frameData.isEmptyFlag() or self.
138              frameData.getRemainingFramesNumber() <= 0:
139              self.isGameJustStarted = True
140              return
141
142          if self.cc.getSkillFlag():
143              self.inputKey = self.cc.getSkillKey()
144              return
145
146          self.inputKey.empty()
147          self.cc.skillCancel()
148          # if not self.isGameJustStarted:
149          #     pass
150          # else:
151          #     # initialize the argument at 1st frame of
152          #     # round
153          #     self.isGameJustStarted = False
154          #
155          # if self.cc.getSkillFlag():
156          #     self.inputKey = self.cc.getSkillKey()
157          #     return
158          # self.inputKey.empty()
```

```
156     # self.cc.skillCancel()
157     # same as env.reset()
158     # if self.just_inited:
159
160     # for lstm
161     # self.actor_hidden_state_list.append(self.actor.
162         hidden_cell[0].squeeze(0).cpu())
163     # self.actor_cell_state_list.append(self.actor.
164         hidden_cell[1].squeeze(0).cpu())
165     # self.critic_hidden_state_list.append(self.
166         critic.hidden_cell[0].squeeze(0).cpu())
167     # self.critic_cell_state_list.append(self.critic.
168         hidden_cell[1].squeeze(0).cpu())
169
170     # if self.currentFrameNum == 14:
171     # self.set_last_hp()
172
173     # for gru
174     obs = self.raw_audio_memory
175     if self.just_inited:
176         self.just_inited = False
177         if obs is None:
178             obs = np.zeros((800 * self.n_frame, 2))
179             self.collect_data_helper.put([obs])
180             terminal = 1
181     elif obs is None:
182         obs = np.zeros((800 * self.n_frame, 2))
183         self.collect_data_helper.put([obs])
184     else:
185         terminal = 0
186         reward = self.get_reward()
187         self.collect_data_helper.put([obs, reward,
188             False, None])
189
190     # get action
191     # self.round_actor_hidden_state_list.append(self.
192         actor.hidden_cell.squeeze(0).to(self.device))
193     state = torch.tensor(obs, dtype=torch.float32)
194     action_dist = self.actor(state.unsqueeze(0).to(
```

```
        self.device), terminal=torch.tensor(terminal).
        float())
187    action = action_dist.sample()
188    # put to helper
189    self.collect_data_helper.put_action(action)
190    if self.rnn:
191        self.collect_data_helper.
192            put_actor_hidden_data(self.actor.
193                hidden_cell.squeeze(0).to(self.device))
194    #
195    self.cc.commandCall(self.actions[action])
196
197    self.inputKey = self.cc.getSkillKey()
198    # end_time = time.time() * 1000
199
200    def get_reward(self):
201        offence_reward = self.pre_framedata.getCharacter(
202            not self.player).getHp() - self.nonDelay.
203            getCharacter(
204                not self.player).getHp()
205        defence_reward = self.nonDelay.getCharacter(self.
206            player).getHp() - self.pre_framedata.
207            getCharacter(
208                self.player).getHp()
209        return offence_reward + defence_reward
210
211    def set_last_hp(self):
212        self.last_my_hp = self.nonDelay.getCharacter(self.
213            player).getHp()
214        self.last_opp_hp = self.nonDelay.getCharacter(not
215            self.player).getHp()
216
217    def getAudioData(self, audio_data):
218        self.audio_data = audio_data
219        # process audio
```

```
214     try:
215         byte_data = self.audio_data.getRawDataAsBytes()
216         np_array = np.frombuffer(byte_data, dtype=np.
217             float32)
218         raw_audio = np_array.reshape((2, 1024))
219         raw_audio = raw_audio.T
220         raw_audio = raw_audio[:800, :]
221     except Exception as ex:
222         raw_audio = np.zeros((800, 2))
223     if self.raw_audio_memory is None:
224         # self.logger.info('raw_audio_memory none
225         # {}'.format(raw_audio.shape))
226         self.raw_audio_memory = raw_audio
227     else:
228         self.raw_audio_memory = np.vstack((raw_audio,
229             self.raw_audio_memory))
230         # self.raw_audio_memory = self.
231         # raw_audio_memory[:4, :, :]
232         self.raw_audio_memory = self.raw_audio_memory
233         [:800 * self.n_frame, :]
234
235     # append so that audio memory has the first shape
236     # of n_frame
237     increase = (800 * self.n_frame - self.
238         raw_audio_memory.shape[0]) // 800
239     for _ in range(increase):
240         self.raw_audio_memory = np.vstack((np.zeros
241             ((800, 2)), self.raw_audio_memory))
242
243     class Java:
244         implements = ["aiinterface.AIInterface"]
245
246     def reset(self):
247         self.collect_data_helper = CollectDataHelper(self
248             .logger)
```

```
241 class CollectDataHelper:  
242     total_round_data = []  
243     total_round_action_data = []  
244     total_round_action_dist_data = []  
245     total_round_actor_hidden_data = []  
246  
247     current_round_data = []  
248     current_round_action = []  
249     current_round_action_dist_data = []  
250     current_round_actor_hidden_data = []  
251     # curr_idx = 0  
252  
253     def __init__(self, logger) -> None:  
254         self.total_round_data = []  
255         self.total_round_action_data = []  
256         self.total_round_action_dist_data = []  
257         self.total_round_actor_hidden_data = []  
258  
259         self.current_round_data = []  
260         self.current_round_action = []  
261         self.current_round_action_dist_data = []  
262         self.current_round_actor_hidden_data = []  
263         self.logger = logger  
264         self.logger.info('create new data helper')  
265     def put(self, data):  
266         if(len(data) == 1):  
267             self.logger.info('put data at game reset')  
268             if len(self.current_round_data) > 0 and len(  
269                 self.current_round_data[-1]) == 1:  
270                 self.logger.info('game reset data exists'  
271                         )  
272             self.current_round_data.append(data)  
273  
274     def put_action(self, action):  
275         self.current_round_action.append(action)  
276  
277     def put_action_dist(self, action_dist):
```

```
276         self.current_round_action_dist_data.append(
277             action_dist)
278
279     def put_actor_hidden_data(self, hidden_data):
280         self.current_round_actor_hidden_data.append(
281             hidden_data)
282
283     def finish_round(self):
284         self.total_round_data.append(self.
285             current_round_data)
286         self.current_round_data = []
287
288         self.total_round_action_data.append(self.
289             current_round_action)
290         self.current_round_action = []
291
292         self.total_round_action_dist_data.append(self.
293             current_round_action_dist_data)
294         self.current_round_action_dist_data = []
295
296         self.total_round_actor_hidden_data.append(self.
297             current_round_actor_hidden_data)
298         self.current_round_actor_hidden_data = []
299
300         self.total_round_action_data = []
301         self.current_round_action = []
302
303         self.total_round_action_dist_data = []
304         self.current_round_action_dist_data = []
305
306         self.total_round_actor_hidden_data = []
307         self.current_round_actor_hidden_data = []
```

```
307
308 class SandboxAgent:
309     def __init__(self, gateway, **kwargs):
310         self.gateway = gateway
311         self.nonDelay = None
312         # self.logger = kwargs.get('logger')
313
314     def initialize(self, gameData, player):
315         # Initializng the command center, the simulator
316         # and some other things
317         self.inputKey = self.gateway.jvm.struct.Key()
318         self.frameData = self.gateway.jvm.struct.
319             FrameData()
320         self.cc = self.gateway.jvm.aiinterface.
321             CommandCenter()
322         self.player = player # p1 == True, p2 == False
323         self.gameData = gameData
324         self.simulator = self.gameData.getSimulator()
325         self.isGameJustStarted = True
326         return 0
327
328     def close(self):
329         pass
330
331     def getInformation(self, frameData, inControl,
332                         nonDelay):
333         # Load the frame data every time getInformation
334         # gets called
335         self.frameData = frameData
336         self.cc.setFrameData(self.frameData, self.player)
337         # nonDelay = self.frameData
338         self.pre_framedata = self.nonDelay if self.
339             nonDelay is not None else nonDelay
340         self.nonDelay = nonDelay
341         self.isControl = inControl
342         self.currentFrameNum = nonDelay.getFramesNumber()
343         # first frame is 14
```

```
337
338     def roundEnd(self, x, y, z):
339         print(x)
340         print(y)
341         print(z)
342
343     def getScreenData(self, sd):
344         pass
345
346     def input(self):
347         return self.inputKey
348     def processing(self):
349         self.inputKey.empty()
350         self.cc.skillCancel()
351
352     def getAudioData(self, audio_data):
353         pass
354
355     class Java:
356         implements = ["aiinterface.AIInterface"]
```

付録.B-2 model.py

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 from torch import distributions
5 import numpy as np
6 HIDDEN_SIZE_1 = 256
7 HIDDEN_SIZE_2 = 256
8
9 class RecurrentActor(nn.Module):
10     def __init__(self, state_dim, hidden_size,
11                  recurrent_layers, encoder=None, action_num=11):
12         super().__init__()
13         self.hidden_size = hidden_size
14         self.recurrent_layers = recurrent_layers
15         self.gru = nn.GRU(state_dim, hidden_size,
16                           recurrent_layers)
16         self.fc1 = nn.Linear(hidden_size, HIDDEN_SIZE_1)
17         self.fc2 = nn.Linear(HIDDEN_SIZE_1, HIDDEN_SIZE_2
18                             )
19         self.action_layer = nn.Linear(HIDDEN_SIZE_2,
20                                       action_num)
21         self.hidden_cell = None
22         self.encoder = encoder
23         self.flatten = nn.Flatten()
24         self.relu = nn.ReLU()
25
26     def get_init_state(self, device):
27         # self.hidden_cell = torch.zeros(self.
28             recurrent_layers, batch_size, self.hidden_size
29             ).to(device)
30         self.hidden_cell = torch.zeros(self.
31             recurrent_layers, self.hidden_size).to(device)
32
33     def forward(self, state, terminal=None):
```

```
28     # batch_size = state.shape[1]
29     device = state.device
30
31     # perform audio encoder
32     if self.encoder is not None:
33         state_encoded = self.relu(self.flatten(self.
34             encoder(state)))
35
36     if self.hidden_cell is None: # or batch_size != self.hidden_cell[0].shape[1]:
37         self.get_init_state(device)
38     if terminal is not None:
39         self.hidden_cell = (self.hidden_cell * (1. -
40             terminal)).reshape(-1, )
41
42     try:
43         self.hidden_cell = self.hidden_cell.to(device
44             )
45         rnn_output, self.hidden_cell = self.gru(
46             state_encoded, self.hidden_cell)
47     except Exception as ex:
48         raise ex
49     # hidden1 = F.elu(self.fc1(self.hidden_cell
50         [0][-1]))
51     # hidden1 = F.elu(self.fc1(self.hidden_cell[-1]))
52     hidden1 = F.elu(self.fc1(rnn_output))
53     hidden2 = F.elu(self.fc2(hidden1))
54     policy_logits_out = self.action_layer(hidden2)
55     policy_dist = distributions.Categorical(F.softmax
56         (policy_logits_out, dim=1).to(device))
57
58     return policy_dist
59
60
61     def act(self, state, terminal=None):
62         # batch_size = state.shape[1]
63         device = state.device
64         # perform audio encoder
65         if self.encoder is not None:
```

```
58         state_encoded = self.relu(self.flatten(self.
59                                     encoder(state)))
60
61         if self.hidden_cell is None: # or batch_size != self.hidden_cell[0].shape[1]:
62             self.get_init_state(device)
63
64         if terminal is not None:
65             self.hidden_cell = (self.hidden_cell * (1. -
66                                         terminal))#.reshape(-1,)
67
68     try:
69         self.hidden_cell = self.hidden_cell.to(device
70             )
71         rnn_output, self.hidden_cell = self.gru(
72             state_encoded, self.hidden_cell)
73
74     except Exception as ex:
75         raise ex
76
77     # hidden1 = F.elu(self.fc1(self.hidden_cell
78     # [0][-1]))
79
80     # hidden1 = F.elu(self.fc1(self.hidden_cell[-1]))
81     hidden1 = F.elu(self.fc1(rnn_output))
82     hidden2 = F.elu(self.fc2(hidden1))
83
84     policy_logits_out = self.action_layer(hidden2)
85     return F.softmax(policy_logits_out, dim=1)
86
87     # policy_dist = distributions.Categorical(F.
88     # softmax(policy_logits_out, dim=1).to(device))
89
90     # return policy_dist
91
92
93
94 class FeedForwardActor(nn.Module):
95
96     def __init__(self, state_dim, hidden_size,
97                  recurrent_layers, encoder=None, action_num=11):
98
99         super().__init__()
100
101         self.hidden_size = hidden_size
102         self.fc1 = nn.Linear(state_dim, HIDDEN_SIZE_1)
103         self.fc2 = nn.Linear(HIDDEN_SIZE_1, HIDDEN_SIZE_2
104             )
```

```
86         self.action_layer = nn.Linear(HIDDEN_SIZE_2,
87                                         action_num)
88
89         self.encoder = encoder
90
91         self.flatten = nn.Flatten()
92
93         self.relu = nn.ReLU()
94
95
96     def forward(self, state, terminal=None):
97
98         # batch_size = state.shape[1]
99
100        device = state.device
101
102        # perform audio encoder
103
104        if self.encoder is not None:
105
106            state_encoded = self.relu(self.flatten(self.
107                                         encoder(state)))
108
109        else:
110
111            state_encoded = state
112
113            # print(state_encoded.shape)
114
115            hidden1 = F.elu(self.fc1(state_encoded))
116
117            hidden2 = F.elu(self.fc2(hidden1))
118
119            policy_logits_out = self.action_layer(hidden2)
120
121            policy_dist = distributions.Categorical(F.softmax
122
123                                         (policy_logits_out, dim=1).to(device))
124
125            # print(F.softmax(policy_logits_out, dim=0))
126
127            return policy_dist
128
129
130
131    def act(self, state):
132
133        # print(state.sum())
134
135        device = state.device
136
137        # perform audio encoder
138
139        if self.encoder is not None:
140
141            state_encoded = self.relu(self.flatten(self.
142                                         encoder(state)))
143
144        else:
145
146            state_encoded = state
147
148            # print(state_encoded.shape)
149
150            hidden1 = F.elu(self.fc1(state_encoded))
151
152            hidden2 = F.elu(self.fc2(hidden1))
153
154            policy_logits_out = self.action_layer(hidden2)
```

```
119      # print(hidden2)
120      # policy_dist = distributions.Categorical(F.
121          softmax(policy_logits_out, dim=0).to(device))
122      # print(policy_logits_out)
123      policy_logits_out = F.softmax(policy_logits_out,
124          dim=1)
125      # print(policy_logits_out)
126      # action = torch.argmax(policy_logits_out)
127      return policy_logits_out
128
129  class RecurrentCritic(nn.Module):
130      def __init__(self, state_dim, hidden_size,
131          recurrent_layers, encoder=None):
132          super().__init__()
133          self.hidden_size = hidden_size
134          self.recurrent_layers = recurrent_layers
135          self.gru = nn.GRU(state_dim, hidden_size,
136              recurrent_layers)
137          self.fc1 = nn.Linear(hidden_size, HIDDEN_SIZE_1)
138          self.fc2 = nn.Linear(HIDDEN_SIZE_1, HIDDEN_SIZE_2
139              )
140          self.value_layer = nn.Linear(HIDDEN_SIZE_2, 1)
141          self.hidden_cell = None
142          self.encoder = encoder
143          self.flatten = nn.Flatten()
144          self.relu = nn.ReLU()
145
146      def get_init_state(self, device):
147          # self.hidden_cell = (torch.zeros(self.
148              recurrent_layers, batch_size, self.hidden_size
149              ).to(device),
150              torch.zeros(self.
151                  recurrent_layers, batch_size, self.hidden_size
152                  ).to(device))
153          self.hidden_cell = torch.zeros(self.
154              recurrent_layers, self.hidden_size).to(device)
155
```

```
146     def forward(self, state, terminal=None):
147         batch_size = state.shape[0]
148         device = state.device
149         # perform audio encoder
150         if self.encoder is not None:
151             state_encoded = self.relu(self.flatten(self.
152                                         encoder(state)))
153
154             if self.hidden_cell is None:# or batch_size != self.hidden_cell[0].shape[1]:
155                 self.get_init_state(device)
156             if terminal is not None:
157                 self.hidden_cell = (self.hidden_cell * (1. -
158                                                 terminal)).reshape(-1, )
159             rnn_output, self.hidden_cell = self.gru(
160                 state_encoded, self.hidden_cell)
161
162             # hidden1 = F.elu(self.fc1(self.hidden_cell
163             [0][-1]))
164
165             hidden1 = F.elu(self.fc1(self.hidden_cell[-1]))
166             hidden1 = F.elu(self.fc1(rnn_output))
167             hidden2 = F.elu(self.fc2(hidden1))
168             value_out = self.value_layer(hidden2)
169             return value_out
170
171
172     class FeedForwardCritic(nn.Module):
173         def __init__(self, state_dim, hidden_size,
174                      recurrent_layers, encoder=None):
175             super().__init__()
176             self.hidden_size = hidden_size
177             self.fc1 = nn.Linear(state_dim, HIDDEN_SIZE_1)
178             self.fc2 = nn.Linear(HIDDEN_SIZE_1, HIDDEN_SIZE_2
179                                 )
180             self.value_layer = nn.Linear(HIDDEN_SIZE_2, 1)
181             self.encoder = encoder
182             self.flatten = nn.Flatten()
183             self.relu = nn.ReLU()
```

```
176
177     def forward(self, state, terminal=None):
178         batch_size = state.shape[0]
179         device = state.device
180         # perform audio encoder
181         if self.encoder is not None:
182             state_encoded = self.relu(self.flatten(self.
183                                         encoder(state)))
184         else:
185             state_encoded = state
186             hidden1 = F.elu(self.fc1(state_encoded))
187             hidden2 = F.elu(self.fc2(hidden1))
188             value_out = self.value_layer(hidden2)
189             return value_out
```

付録.C トレーニングのソースコード

付録.C-1 Train.py

```
1 import argparse
2 import math
3 from msilib import sequence
4 import time
5 import os
6 import re
7 import psutil
8 import numpy as np
9 import platform
10 import signal
11 import subprocess
12 import sys
13 import torch
14 import torch.nn.functional as F
15 from dataclasses import dataclass
16 from dotmap import DotMap
17 from py4j.java_gateway import JavaGateway,
    GatewayParameters, CallbackServerParameters, get_field
18 from torch import optim
19 from torch.utils.tensorboard import SummaryWriter
20
21 from agent import SoundAgent, CollectDataHelper,
    SandboxAgent
22 from model import RecurrentActor, RecurrentCritic,
    FeedForwardActor, FeedForwardCritic
23 from encoder import SampleEncoder, RawEncoder, FFTEncoder
    , MelSpecEncoder, TransformerEncoder, NewEncoder
24 import pickle
25 import tqdm
26 import pathlib
27 import logging
28 logger = logging.getLogger(__name__)
```

```
29 logger.setLevel(logging.DEBUG)
30 # create console handler and set level to debug
31 ch = logging.StreamHandler()
32 ch.setLevel(logging.DEBUG)
33
34 # create formatter
35 formatter = logging.Formatter('%(asctime)s - %(name)s -
36 %(levelname)s - %(message)s')
37 # add formatter to ch
38 ch.setFormatter(formatter)
39
40 # add ch to logger
41 logger.addHandler(ch)
42 # logging.basicConfig(format='%(asctime)s %(message)s')
43 # torch.set_num_threads(2)
44 TRAINING_ITERATION = 900000
45 GAME_NUM = 1
46 TRAIN_EPOCH = 2
47 JAVA_GATEWAY_PORT = 4242
48 HIDDEN_SIZE = 512
49 RECURRENT_LAYERS = 1
50 LEARNING_RATE = 0.0002
51 GAMMA = 0.99
52 C1 = 0.95
53 LAMBDA = 0.95
54 ENCODER_NAME = 'conv1d'
55 # EXPERIMENT_NAME = 'experiment_{}'.format(args.encoder)
56 BASE_CHECKPOINT_PATH = f'ppo-pytorch/checkpoints'
57 ROLL_OUT = 3600
58 # BATCH_SIZE = 512
59 BATCH_SIZE = 8
60 STATE_DIM = {
61     1: {
62         'conv1d': 160,
63         'fft': 512,
64         'mel': 2560,
```

```
65         'trans':512,
66         'new':512
67     },
68     4: {
69         'conv1d': 64,
70         'fft': 512,
71         'mel': 1280,
72         'trans':512,
73         'new':512
74     }
75 }
76 GATHER_DEVICE = 'cuda' if torch.cuda.is_available() else
77     'cpu',
78 #GATHER_DEVICE = "cpu"
79 # GATHER_DEVICE = 'cuda' if torch.cuda.is_available()
80     else 'cpu'
81 TRAIN_DEVICE = 'cuda' if torch.cuda.is_available() else
82     'cpu',
83 SEQUENCE_LEN = 32
84 PPO_CLIP = 0.2
85 ENTROPY_FACTOR = 0.01
86 VF_FACTOR = 1
87 MAX_GRAD_NORM = 1.0
88 ACTION_NUM = 11
89 N_FRAME = 1
90
91 def kill_proc_tree(pid, including_parent=True):
92     parent = psutil.Process(pid)
93     children = parent.children(recursive=True)
94     for child in children:
95         child.kill()
96     gone, still_alive = psutil.wait_procs(children,
97                                           timeout=5)
98     if including_parent:
99         parent.kill()
```

```
98         parent.wait(5)
99
100    @torch.no_grad()
101    def process_game_agent_data(actor_, critic_,
102        data_collect_helper, recurrent):
103        logger.info('process game agent data')
104        number_round = len(data_collect_helper.
105            total_round_data)
106        episode_lengths = []
107        state_list = []
108        value_list = []
109        action_list = []
110        action_probs_list = []
111        true_reward_list = []
112        reward_list = []
113        terminal_list = []
114        actor_hidden_state_list = []
115        critic_hidden_state_list = []
116        for r in tqdm.trange(number_round):
117            round_data = data_collect_helper.total_round_data
118            [r]
119            round_actions = data_collect_helper.
120            total_round_action_data[r]
121            round_action_dists = data_collect_helper.
122            total_round_action_dist_data[r]
123            # round_actor_hidden_state_list =
124            data_collect_helper.
125            total_round_actor_hidden_data[r]
126
127            round_state_list = []
128            round_value_list = []
129            round_action_list = []
130            round_action_probs_list = []
131            round_true_reward_list = []
132            round_reward_list = []
133            round_terminal_list = []
134            round_actor_hidden_state_list = []
```

```
128     round_critic_hidden_state_list = []
129
130     obsv = round_data[0][0]
131     if recurrent:
132         actor_.get_init_state(GATHER_DEVICE)
133         critic_.get_init_state(GATHER_DEVICE)
134     terminal = torch.tensor(0).float()
135     updaters = tqdm.tqdm(min(len(round_data) - 1, len(
136         round_actions)) - 1)
137     for i in range(1, min(len(round_data) - 1, len(
138         round_actions))):
139         if len(round_data[i]) == 1:
140             break
141         if recurrent:
142             round_actor_hidden_state_list.append(
143                 actor_.hidden_cell.squeeze(0).to(
144                     GATHER_DEVICE))
145             round_critic_hidden_state_list.append(
146                 critic_.hidden_cell.squeeze(0).to(
147                     GATHER_DEVICE))
148
149             state = torch.tensor(obsv, dtype=torch.
150                 float32)
151             round_state_list.append(state)
152             value = critic_(state.unsqueeze(0).to(
153                 GATHER_DEVICE), terminal.to(GATHER_DEVICE)
154                 )
155             round_value_list.append(value.squeeze())
156             action_dist = actor_(state.unsqueeze(0).to(
157                 GATHER_DEVICE), terminal.to(GATHER_DEVICE)
158                 )
159             action = round_actions[i - 1]
160             round_action_list.append(action.squeeze().to(
161                 GATHER_DEVICE))
162             round_action_probs_list.append(action_dist.
163                 log_prob(action).to(GATHER_DEVICE))
```

```
152         # get obsv, reward, done data
153         obsv, reward, done, _ = round_data[i]
154
155         terminal = torch.tensor(done, device=
156                         GATHER_DEVICE).float()
157         # transformed_reward = normalize_reward(
158         #     reward)
159         transformed_reward = reward
160         round_true_reward_list.append(torch.tensor(
161             reward, device=GATHER_DEVICE).float())
162         round_reward_list.append(torch.tensor(
163             transformed_reward, device=GATHER_DEVICE).
164             float())
165         round_terminal_list.append(terminal)
166         updater.update(1)
167
168         # compute for final state
169         state = torch.tensor(obsv, dtype=torch.float32)
170         value = critic(state.unsqueeze(0).to(
171             GATHER_DEVICE))
172         round_value_list.append(value.squeeze())
173
174         episode_lengths.append(len(round_state_list))
175         state_list.append(round_state_list)
176         value_list.append(round_value_list)
177         action_list.append(round_action_list)
178         action_probs_list.append(round_action_probs_list)
179         true_reward_list.append(round_true_reward_list)
180         reward_list.append(round_reward_list)
181         terminal_list.append(round_terminal_list)
182         actor_hidden_state_list.append(
183             round_actor_hidden_state_list)
184         critic_hidden_state_list.append(
185             round_critic_hidden_state_list)
186
187         if recurrent:
188             trajectories_data = {
```

```
181         "actions": action_list,
182         "action_probabilities": action_probs_list,
183         "states": state_list,
184         "rewards": reward_list,
185         "values": value_list,
186         "true_rewards": true_reward_list,
187         "terminals": terminal_list,
188         # add hidden state list
189         "actor_hidden_states":
190             actor_hidden_state_list,
191         "critic_hidden_states":
192             critic_hidden_state_list,
193     }
194 else:
195     trajectories_data = {
196         "actions": action_list,
197         "action_probabilities": action_probs_list,
198         "states": state_list,
199         "rewards": reward_list,
200         "values": value_list,
201         "true_rewards": true_reward_list,
202         "terminals": terminal_list,
203         # add hidden state list
204     }
205 # print({key: {type(v[0]) for v in value} for key,
206 #         value in trajectories_data.items()})
207 # return tensor
208 for (key, value) in trajectories_data.items():
209     for i in range(len(value)):
210         if len(value[i]) == 0:
211             logger.info(f'empty tensorlist: {key} {i}')
212
213     return {key: [torch.stack(v) for v in value] for key,
214             value in trajectories_data.items()},
215     episode_lengths
216
217 ## maximin normalization
```

```
212 def normalize_reward(reward):
213     return (reward + 400) / 800
214
215 # collect trajectories
216 def collect_trajectories(actor, critic, port, game_num,
217                           p2, rnn, n_frame):
217     logger.info(f 'start fight with {p2}')
218     logger.info(f 'game_num value {game_num}')
219     error = True
220     while error:
221         gateway = JavaGateway(gateway_parameters=
222                               GatewayParameters(port=port),
223                               callback_server_parameters=
224                               CallbackServerParameters()
225 )
226
227     try:
228         manager = gateway.entry_point
229         current_time = int(time.time() * 1000)
230         # register AIs
231         collect_data_helper = CollectDataHelper(
232             logger)
233         agent = SoundAgent(gateway, actor=actor,
234                             critic=critic, collect_data_helper=
235                             collect_data_helper, logger=logger,
236                             n_frame=n_frame, rnn=rnn, device=
237                             GATHER_DEVICE)
238         sandbox_agent = SandboxAgent(gateway)
239         manager.registerAI(f 'SoundAgent', agent)
240         manager.registerAI('Sandbox', sandbox_agent)
241         game = manager.createGame('ZEN', 'ZEN', '
242             SoundAgent', p2, game_num)
243
244         # start game
245         manager.runGame(game)
246
247         # finish game
248         logger.info('Finish game')
```

```
239         sys.stdout.flush()
240
241         # close gateway
242         gateway.close_callback_server()
243         gateway.close()
244         error = False
245     except Exception as ex:
246         print(ex)
247         logger.info('There is an error with the
248             gateway, restarting')
249         gateway.close_callback_server()
250         gateway.close()
251         error = True
252
253     # return agent.get_trajectories_data()
254     agent_data = process_game_agent_data(actor, critic,
255                                         agent.collect_data_helper, rnn)
256
257     # try:
258     #     kill_proc_tree(java_env.pid, False)
259     # except:
260     #     print('kill process')
261     # agent.reset()
262
263     return agent_data
264
265 def calc_discounted_return(rewards, discount, final_value):
266     """
267     Calculate discounted returns based on rewards and
268     discount factor.
269     """
270
271     seq_len = len(rewards)
272     discounted_returns = torch.zeros(seq_len).to(rewards.
273                                     device)
274     discounted_returns[-1] = rewards[-1] + discount *
275                             final_value
276
277     for i in range(seq_len - 2, -1, -1):
```

```
270         discounted_returns[i] = rewards[i] + discount *
271             discounted_returns[i + 1]
272     return discounted_returns
273
274 def compute_advantages(rewards, values, discount,
275                         gae_lambda):
276     """
277     Compute General Advantage.
278     """
279     deltas = rewards + discount * values[1:] - values
280     [:-1]
281     seq_len = len(rewards)
282     advs = torch.zeros(seq_len + 1).to(rewards.device)
283     multiplier = discount * gae_lambda
284     for i in range(seq_len - 1, -1, -1):
285         advs[i] = advs[i + 1] * multiplier + deltas[i]
286     return advs[:-1]
287
288
289
290 def pad_and_compute_returns(trajectory_episodes,
291                            len_episodes):
292     """
293     Pad the trajectories up to hp.rollout_steps so they
294     can be combined in a
295     single tensor.
296     Add advantages and discounted_returns to trajectories
297     .
298     """
299
300     episode_count = len(len_episodes)
301     advantages_episodes, discounted_returns_episodes =
302         [], []
303     padded_trajectories = {key: [] for key in
304                           trajectory_episodes.keys()}
305     padded_trajectories["advantages"] = []
306     padded_trajectories["discounted_returns"] = []
307     episode_max_length = ROLL_OUT
```

```
299
300     for i in range(episode_count):
301         single_padding = torch.zeros(episode_max_length -
302                                       len_episodes[i], device=GATHER_DEVICE)
303         for key, value in trajectory_episodes.items():
304             if value[i].ndim > 2:
305                 padding = torch.zeros(episode_max_length
306                                       - len_episodes[i], value[0].shape[1],
307                                       value[0].shape[2],
308                                       dtype=value[i].
309                                       dtype, device=
310                                       value[i].device)
311             elif value[i].ndim > 1:
312                 padding = torch.zeros(episode_max_length
313                                       - len_episodes[i], value[0].shape[1],
314                                       dtype=value[i].dtype, device=value[i].
315                                       device)
316             else:
317                 padding = torch.zeros(episode_max_length
318                                       - len_episodes[i], dtype=value[i].
319                                       dtype, device=value[i].device)
320
321         try:
322             padded_trajectories[key].append(torch.cat(
323               ((value[i], padding)))
324         except Exception as ex:
325             print(ex)
326         padded_trajectories["advantages"].append(
327             torch.cat((compute_advantages(rewards=
328                           trajectory_episodes["rewards"])[i],
329                           values=
330                               trajectory_episodes
331                               [ "values" ][i
332                               ],
333                               discount=GAMMA,
334                               gae_lambda=
335                               LAMBDA),
336                               single_padding
```

```
        )))

319     padded_trajectories["discounted_returns"].append(
320         torch.cat((calc_discounted_return(rewards=
321                     trajectory_episodes["rewards"])[i],
322                               discount=
323                               GAMMA,
324                               final_value
325                               =
326                               trajectory_episode
327                               ["values"
328                               ""] [i
329                               ] [-1]),
330                               single_padding
331                               )))
332
333     return_val = {k: torch.stack(v) for k, v in
334     padded_trajectories.items()}
335
336     return_val["seq_len"] = torch.tensor(len_episodes)
337
338     return return_val
339
340
341 @dataclass
342 class TrajectoryBatchRNN():
343     """
344     Dataclass for storing data batch.
345     """
346
347     states: torch.tensor
348     actions: torch.tensor
349     action_probabilities: torch.tensor
350     advantages: torch.tensor
351     discounted_returns: torch.tensor
352     batch_size: torch.tensor
353     actor_hidden_states: torch.tensor
354     # actor_cell_states: torch.tensor
355     critic_hidden_states: torch.tensor
356     # critic_cell_states: torch.tensor
357
358
359 @dataclass
```

```
345 class TrajectoryBatch():
346     """
347     Dataclass for storing data batch.
348     """
349     states: torch.tensor
350     actions: torch.tensor
351     action_probabilities: torch.tensor
352     advantages: torch.tensor
353     discounted_returns: torch.tensor
354     batch_size: torch.tensor
355     # actor_hidden_states: torch.tensor
356     # actor_cell_states: torch.tensor
357     # critic_hidden_states: torch.tensor
358     # critic_cell_states: torch.tensor
359
360
361 class TrajectoryDataset():
362     """
363     Fast dataset for producing training batches from
364     trajectories.
365     """
366     def __init__(self, trajectories, batch_size, device,
367                  sequence_len, recurrent=True):
368
369         # Combine multiple trajectories into
370         self.trajectories = {key: value.to(device) for
371                             key, value in trajectories.items()}
372         self.sequence_len = sequence_len
373         truncated_seq_len = torch.clamp(trajectories[""
374                                         "seq_len"] - sequence_len + 1, 0, ROLL_OUT)
375         self.cumsum_seq_len = np.cumsum(np.concatenate((
376             np.array([0]), truncated_seq_len.numpy())))
377         self.batch_size = batch_size
378         self.recurrent = recurrent
379
380     def __iter__(self):
```

```
377         self.valid_idx = np.arange(self.cumsum_seq_len
378                                     [-1])
379         self.batch_count = 0
380         return self
381
382     def __next__(self):
383         # print(self.batch_count * self.batch_size, math.
384         #        ceil(self.cumsum_seq_len[-1] / self.batch_len)
385         #        )
386
387         if self.batch_count * self.batch_size >= math.
388             ceil(self.cumsum_seq_len[-1] / self.
389                  sequence_len):
390             raise StopIteration
391         else:
392             actual_batch_size = min(len(self.valid_idx),
393                                     self.batch_size)
394
395             start_idx = np.random.choice(self.valid_idx,
396                                         size=actual_batch_size, replace=False)
397
398             self.valid_idx = np.setdiff1d(self.valid_idx,
399                                         start_idx)
400
401             eps_idx = np.digitize(start_idx, bins=self.
402                                   cumsum_seq_len, right=False) - 1
403
404             seq_idx = start_idx - self.cumsum_seq_len[
405                         eps_idx]
406
407             series_idx = np.linspace(seq_idx, seq_idx +
408                                     self.sequence_len - 1, num=self.
409                                     sequence_len, dtype=np.int64)
410
411             self.batch_count += 1
412
413             if self.recurrent:
414                 return TrajectorBatchRNN(**{key:
415
416                     magic_combine(value[eps_idx],
417                                   series_idx], 0, 2) for key, value
418
419                     in self.trajectories
420                     .items() if key
421
422                     in
423                     TrajectorBatchRNN
```

```
396                                         __dataclass_fields__
397                                         .keys()}) ,
398                                         batch_size=
399                                         actual_batch_size)
400                                         return TrajectoryBatch(**{key: magic_combine(
401                                         value[eps_idx, series_idx], 0, 2) for key,
402                                         value
403                                         in self.trajectories
404                                         .items() if key
405                                         in TrajectoryBatch
406                                         .
407                                         __dataclass_fields__
408                                         .keys()}) ,
409                                         batch_size=
410                                         actual_batch_size)
411
412 def train_model(actor, critic, actor_optimizer,
413                 critic_optimizer, iteration, port, encoder_name,
414                 experiment_id, p2, recurrent, n_frame):
415     loop_count = 0
416     if not recurrent:
417         writer = SummaryWriter(log_dir=f'ppo-pytorch/logs
418                               /{encoder_name}/{experiment_id}')
419     else:
420         writer = SummaryWriter(log_dir=f'ppo-pytorch/logs
421                               /{encoder_name}/rnn/{experiment_id}')
422     iteration += 1
423     while iteration < TRAINING_ITERATION:
424         torch.cuda.empty_cache()
425         logger.info(f'clear cuda memory')
426         logger.info(f"Training iterator {iteration}
427                     ")
428         actor = actor.to(GATHER_DEVICE)
429         critic = critic.to(GATHER_DEVICE)
430         start_gather_time = time.time()
```

```
416     loop_count += 0
417     # gather trajectories
418     logger.info(f 'Gathering trajectories data for {
419         GAME_NUM} games')
420     total_trajectories_data = []
421     total_episode_lengths = []
422
423     trajectories_data = {}
424     # run 1 game in GAME_NUM times and concatenate
425     # game data together
426     for i in range(GAME_NUM):
427         logger.info('Start game {}'.format(i+1))
428         game_trajectories_data, game_episode_lengths
429             = collect_trajectories(actor, critic, port
430             , 1, p2, recurrent, n_frame)
431         total_trajectories_data.append(
432             game_trajectories_data)
433
434         for k, v in game_trajectories_data.items():
435             if trajectories_data.get(k, None) is None
436                 :
437                 trajectories_data[k] = v
438             else:
439                 trajectories_data[k] =
440                     trajectories_data[k] + v
441             total_episode_lengths += game_episode_lengths
442
443             # for game_trajectories_data in
444             episode_lengths = total_episode_lengths
445             # trajectories_data = process_trajectories(
446             #     trajectories_data)
447             # episode_lengths = trajectories_data['
448             #     episode_lengths']
449             logger.info('Calculate returns')
450             trajectories = pad_and_compute_returns(
451                 trajectories_data, episode_lengths)
452             logger.info('Calculate mean reward')
```

```
443
444     # sum of rewards over all steps is actually
        HP_self - HP_opp at the end of round.
445     complete_episode_count = len(trajectories_data['
        states'])
446     terminal_episodes_rewards = trajectories["
        true_rewards"].sum(axis=1).sum()
447     mean_reward = terminal_episodes_rewards /
        complete_episode_count
448     stddev_reward = np.std(trajectories["
        true_rewards"
        ].sum(axis=1).cpu().numpy())
449
450     # normalized rewards
451     normalized_terminal_episodes_rewards =
        trajectories["
        rewards"].sum(axis=1).sum()
452     normalized_mean_reward =
        normalized_terminal_episodes_rewards /
        complete_episode_count
453     normalized_stddev_reward = np.std(trajectories["
        rewards"]
        .sum(axis=1).cpu().numpy())
454
455     # log mean_reward
456     trajectory_dataset = TrajectoryDataset(
        trajectories, batch_size=BATCH_SIZE,
457                                         device=
                                                TRAIN_DEVICE
                                                ,
                                                sequence_len
                                                =
                                                SEQUENCE_LEN
                                                ,
                                                recurrent
                                                =
                                                recurrent
                                                )
458     end_gather_time = time.time()
459     start_train_time = time.time()
```

```
460     actor = actor.to(TRAIN_DEVICE)
461     critic = critic.to(TRAIN_DEVICE)
462     train_count = 0
463     logger.info('Start policy gradient')
464     # Train actor and critic
465     for _ in tqdm.trange(TRAIN_EPOCH):
466         for batch in trajectory_dataset:
467             # Get batch
468             # actor.hidden_cell = (batch.
469             # actor_hidden_states[:1], batch.
470             # actor_cell_states[:1])
471             if recurrent:
472                 actor.hidden_cell = batch.
473                 actor_hidden_states[:1]
474
475             # Update actor
476             actor_optimizer.zero_grad()
477             action_dist = actor(batch.states)
478             # Action dist runs on cpu as a workaround
479             # to CUDA illegal memory access.
480             # action_probabilities = action_dist.
481             log_prob(batch.actions.to("cpu")).to(
482             # TRAIN_DEVICE) # batch.actions[-1,
483             # :]
484             action_probabilities = action_dist.
485             log_prob(batch.actions).to(
486             TRAIN_DEVICE)
487             # Compute probability ratio from
488             # probabilities in logspace.
489             probabilities_ratio = torch.exp(
490                 action_probabilities - batch.
491                 action_probabilities) # [-1, :])
492             surrogate_loss_0 = probabilities_ratio *
493                 batch.advantages # [-1, :]
494             surrogate_loss_1 = torch.clamp(
495                 probabilities_ratio, 1. - PPO_CLIP,
496                 1. +
```

```
PPO_CLIP
) *
batch.
advantages
#
[-1, :]

485    actor_loss = -torch.mean(torch.min(
        surrogate_loss_0, surrogate_loss_1))
486    surrogate_loss_2 = action_dist.entropy().
        to(TRAIN_DEVICE)
487    # actor_loss = -torch.mean(torch.min(
        surrogate_loss_0, surrogate_loss_1)) -
        torch.mean(
488    #     ENTROPY_FACTOR * surrogate_loss_2)
489    actor_loss.backward()
490    torch.nn.utils.clip_grad.clip_grad_norm_(
        actor.parameters(), MAX_GRAD_NORM)
491    actor_optimizer.step()

492
493    # Update critic
494    critic_optimizer.zero_grad()
495    if recurrent:
496        critic.hidden_cell = batch.
            critic_hidden_states[:1] # ,
            batch.critic_cell_states[:1])
497        values = critic(batch.states)
498        critic_loss = F.mse_loss(batch.
            discounted_returns, values.squeeze())
            # batch.discounted_returns[-1, :]
499        torch.nn.utils.clip_grad.clip_grad_norm_(
            critic.parameters(), MAX_GRAD_NORM)
500        critic_loss.backward()
501        critic_optimizer.step()
502    end_train_time = time.time()
503
504    # save mean reward to text file
505    logger.info("Save mean reward to file")
```

```
506     save_reward_file(encoder_name, experiment_id,
507                         mean_reward.float(), recurrent=recurrent,
508                         filename='result')
509
510     # logger.info("Save stddev reward to file")
511     # save_reward_file(encoder_name, experiment_id,
512                         stddev_reward, filename="std", recurrent=
513                         recurrent)
514
515     # # save normalized_rewards to text file
516     # logger.info('Save mean normalized reward to
517                   file')
518     # save_reward_file(encoder_name, experiment_id,
519                         normalized_mean_reward.float(), filename='
520                         reward_normalized', recurrent=recurrent)
521
522     # logger.info('Save std normalized reward to file
523                   ')
524
525     # save_reward_file(encoder_name, experiment_id,
526                         normalized_stddev_reward, filename="
527                         std_normalized", recurrent=recurrent)
528
529     logger.info(f"Iteration: {iteration}, Reward std:
530                 {stddev_reward}, Mean reward: {mean_reward},
531                 Mean Entropy: {torch.mean(surrogate_loss_2)},
532                 "
533                 + f"complete_episode_count: {"
534                     complete_episode_count}, Gather time: {
535                         end_gather_time - start_gather_time:.2f}
536                         s, " +
537                         f"Train time: {end_train_time -
538                             start_train_time:.2f}s")
539
540     save_checkpoint(actor, critic, actor_optimizer,
541                     critic_optimizer, iteration, encoder_name,
542                     experiment_id, recurrent)
543
544     logger.info(f'Saved checkpoint')
545
546     # write tensorboard log
547     writer.add_scalar("complete_episode_count",
```

```
    complete_episode_count, iteration)
524 writer.add_scalar("total_reward", mean_reward,
                     iteration)
525 writer.add_scalar("actor_loss", actor_loss,
                     iteration)
526 writer.add_scalar("critic_loss", critic_loss,
                     iteration)
527 writer.add_scalar("policy_entropy", torch.mean(
                     surrogate_loss_2), iteration)
528 writer.add_scalar("total_normalized_reward",
                     normalized_mean_reward, iteration)
529 logger.info(f 'Saved report')
530
531 # write actor, critic parameters
532 save_parameters(writer, "actor", actor, iteration
                  , encoder_name, experiment_id)
533 save_parameters(writer, "value", critic,
                  iteration, encoder_name, experiment_id)
534 logger.info(f 'Saved parameters')
535 #logger.info(f'{iteration}')
536 iteration += 1
537 torch.cuda.empty_cache()
538 logger.info(f 'clear cuda memory')
539
540
541 # del loss
542 del surrogate_loss_0
543 del surrogate_loss_1
544 del probabilities_ratio
545 del critic_loss
546 del actor_loss
547 print(iteration)
548 if iteration>13:
549     sys.exit()
550
551
552
```

```
553 def save_parameters(writer, tag, model, batch_idx,
554     encoder, experiment_id):
555     """
556     Save model parameters for tensorboard.
557     """
558     _INVALID_TAG_CHARACTERS = re.compile(r"[^-/\w\.]")
559     for k, v in model.state_dict().items():
560         shape = v.shape
561         # Fix shape definition for tensorboard.
562         shape_formatted = _INVALID_TAG_CHARACTERS.sub("_",
563             str(shape))
564         # Don't do this for single weights or biases
565         if np.any(np.array(shape) > 1):
566             mean = torch.mean(v)
567             std_dev = torch.std(v)
568             maximum = torch.max(v)
569             minimum = torch.min(v)
570             writer.add_scalars(
571                 "{}_{}_{}-weights/{}{}".format(encoder,
572                     experiment_id, tag, k, shape_formatted),
573                 {"mean": mean, "std_dev": std_dev, "max": maximum,
574                  "min": minimum},
575                 batch_idx
576             )
577         else:
578             writer.add_scalar("{}_{}{}".format(tag, k,
579                 shape_formatted), v.data, batch_idx)
580
581     def save_reward_file(encoder, experiment_id, reward,
582         filename="reward", recurrent=True):
583         if not recurrent:
584             file_name = f'{filename}_{encoder}_{experiment_id}.txt'
585         else:
586             file_name = f'{filename}_{encoder}_{experiment_id}
```

```
        }_rnn.txt'
582     with open(file_name, 'a') as f:
583         f.write(str(reward))
584         f.write('\n')
585
586
587 def init(encoder_name, experiment_id, n_frame, rnn=True):
588     # TODO load data from checkpoint
589     if rnn:
590         print('init recurrent network')
591     else:
592         print('init feedforward network')
593     # initialize new data
594     if rnn:
595         actor_model = RecurrentActor(STATE_DIM[n_frame][
596             encoder_name], HIDDEN_SIZE, RECURRENT_LAYERS,
597             get_sound_encoder(encoder_name, n_frame),
598             action_num=ACTION_NUM)
599     else:
600         actor_model = FeedForwardActor(STATE_DIM[n_frame][
601             encoder_name], HIDDEN_SIZE, RECURRENT_LAYERS,
602             get_sound_encoder(encoder_name, n_frame),
603             action_num=ACTION_NUM)
604     actor_opt = optim.Adam(actor_model.parameters(), lr=
605         LEARNING_RATE)
606     if rnn:
607         critic_model = RecurrentCritic(STATE_DIM[n_frame][
608             encoder_name], HIDDEN_SIZE, RECURRENT_LAYERS,
609             get_sound_encoder(encoder_name, n_frame))
610     else:
611         critic_model = FeedForwardCritic(STATE_DIM[
612             n_frame][encoder_name], HIDDEN_SIZE,
613             RECURRENT_LAYERS, get_sound_encoder(
614                 encoder_name, n_frame))
615     critic_opt = optim.Adam(critic_model.parameters(), lr
616         =LEARNING_RATE)
617
618     return actor_model, critic_model, actor_opt, critic_opt, file_name
```

```
607     max_checkpoint_iteration =
608         get_last_checkpoint_iteration(encoder_name,
609             experiment_id, rnn)
610     if max_checkpoint_iteration > -1:
611         actor_state_dict, critic_state_dict,
612             actor_optimizer_state_dict,
613             critic_optimizer_state_dict = load_checkpoint(
614                 encoder_name, experiment_id,
615                     max_checkpoint_iteration, rnn)
616         actor_model.load_state_dict(actor_state_dict,
617             strict=True)
618         critic_model.load_state_dict(critic_state_dict,
619             strict=True)
620         actor_opt.load_state_dict(
621             actor_optimizer_state_dict)
622         critic_opt.load_state_dict(
623             critic_optimizer_state_dict)
624     # We have to move manually move optimizer states
625         to TRAIN_DEVICE manually since optimizer doesn
626         't yet have a "to" method.
627     for state in actor_opt.state.values():
628         for k, v in state.items():
629             if isinstance(v, torch.Tensor):
630                 state[k] = v.to(TRAIN_DEVICE)
631     for state in critic_opt.state.values():
632         for k, v in state.items():
633             if isinstance(v, torch.Tensor):
634                 state[k] = v.to(TRAIN_DEVICE)
635     return actor_model, critic_model, actor_opt,
636             critic_opt, max_checkpoint_iteration
637
638
639 def get_last_checkpoint_iteration(encoder_name,
640     experiment_id, rnn):
641     """
642     Determine latest checkpoint iteration.
643     """
644
```

```
631     if not rnn:
632         CHECKPOINT_PATH = f'{BASE_CHECKPOINT_PATH}/{encoder_name}/{experiment_id}/'
633     else:
634         CHECKPOINT_PATH = f'{BASE_CHECKPOINT_PATH}/{encoder_name}/rnn/{experiment_id}/'
635     if os.path.isdir(CHECKPOINT_PATH):
636         max_checkpoint_iteration = max([int(dirname) for
637                                         dirname in os.listdir(CHECKPOINT_PATH)])
638     else:
639         max_checkpoint_iteration = -1
640
641
642 def get_sound_encoder(encoder_name, n_frame):
643     encoder = None
644     if encoder_name == 'conv1d':
645         encoder = RawEncoder(frame_skip=n_frame)
646     elif encoder_name == 'fft':
647         encoder = FFTEncoder(frame_skip=n_frame)
648     elif encoder_name == 'mel':
649         encoder = MelSpecEncoder(frame_skip=n_frame)
650     elif encoder_name == 'trans':
651         encoder = TransformerEncoder(frame_skip=n_frame)
652     elif encoder_name == 'new':
653         encoder = NewEncoder(frame_skip=n_frame)
654     else:
655         encoder = SampleEncoder()
656
657
658
659 # combine dimensions of a tensor
660 def magic_combine(x, dim_begin, dim_end):
661     combined_shape = list(x.shape[:dim_begin]) + [-1] +
662                             list(x.shape[dim_end:])
663     return x.view(combined_shape)
```

```
664
665 def save_checkpoint(actor, critic, actor_optimizer,
666     critic_optimizer, iteration, encoder_name,
667     experiment_id, rnn):
668     """
669     Save training checkpoint.
670     """
671     checkpoint = DotMap()
672     # checkpoint.env = ENV
673     checkpoint.iteration = iteration
674     # checkpoint.stop_conditions = stop_conditions
675     # checkpoint.hp = hp
676     # CHECKPOINT_PATH = BASE_CHECKPOINT_PATH + f"{{"
677     #     iteration}}/"
678     if not rnn:
679         CHECKPOINT_PATH = f'{BASE_CHECKPOINT_PATH}/{{'
680         encoder_name}}/{{experiment_id}}/{{iteration}}/'
681     else:
682         CHECKPOINT_PATH = f'{BASE_CHECKPOINT_PATH}/{{'
683         encoder_name}}/{rnn}/{{experiment_id}}/{{iteration}}/'
684
685     pathlib.Path(CHECKPOINT_PATH).mkdir(parents=True,
686         exist_ok=True)
687     with open(CHECKPOINT_PATH + "parameters.pt", "wb") as
688         f:
689             pickle.dump(checkpoint, f)
690     with open(CHECKPOINT_PATH + "actor-class.pt", "wb")
691         as f:
692             pickle.dump(type(actor), f)
693     with open(CHECKPOINT_PATH + "critic-class.pt", "wb")
694         as f:
695             pickle.dump(type(critic), f)
696     torch.save(actor.state_dict(), CHECKPOINT_PATH + "
697         actor.pt")
698     torch.save(critic.state_dict(), CHECKPOINT_PATH + "
699         critic.pt")
```

```
689     torch.save(actor_optimizer.state_dict(),
690                CHECKPOINT_PATH + "actor_optimizer.pt")
691     torch.save(critic_optimizer.state_dict(),
692                CHECKPOINT_PATH + "critic_optimizer.pt")
693
694     def load_checkpoint(encoder_name, experiment_id,
695                          iteration, rnn):
696         if not rnn:
697             CHECKPOINT_PATH = f'{BASE_CHECKPOINT_PATH}/{encoder_name}/{experiment_id}/{iteration}/'
698         else:
699             CHECKPOINT_PATH = f'{BASE_CHECKPOINT_PATH}/{encoder_name}/rnn/{experiment_id}/{iteration}/'
700
701         with open(CHECKPOINT_PATH + 'parameters.pt', 'rb') as
702             f:
703                 checkpoint = pickle.load(f)
704
705         actor_state_dict = torch.load(CHECKPOINT_PATH + "
706                                         actor.pt", map_location=torch.device(TRAIN_DEVICE))
707
708         critic_state_dict = torch.load(CHECKPOINT_PATH + "
709                                         critic.pt", map_location=torch.device(TRAIN_DEVICE))
710
711         actor_optimizer_state_dict = torch.load(
712             CHECKPOINT_PATH + "actor_optimizer.pt",
713                                         map_location=
714                                         torch.
715                                         device(
716                                         TRAIN_DEVICE
717                                         ))
718
719         critic_optimizer_state_dict = torch.load(
720             CHECKPOINT_PATH + "critic_optimizer.pt",
721                                         map_location
722                                         =torch.
723                                         device(
```

```
TRAIN_DEVICE
))

707     return actor_state_dict, critic_state_dict, \
708         actor_optimizer_state_dict,
709         critic_optimizer_state_dict,
710
711 if __name__ == '__main__':
712     # EXPERIMENT_NAME
713     parser = argparse.ArgumentParser()
714     parser.add_argument('--encoder', type=str, choices=[ 'conv1d', 'fft', 'mel', 'trans', 'new'], default='conv1d', help='Choose an encoder for the Blind AI')
715     parser.add_argument('--port', type=int, default=JAVA_GATEWAY_PORT, help='Port used by DareFightingICE')
716     parser.add_argument('--id', type=str, required=True, help='Experiment id')
717     parser.add_argument('--p2', choices=[ 'Sandbox', 'MctsAi65', 'MctsAi'], type=str, required=True, help='The opponent AI')
718     parser.add_argument('--recurrent', action='store_true', help='Use GRU')
719     parser.add_argument('--n_frame', type=int, default=N_FRAME, help='Number of frame to sample data')
720     args = parser.parse_args()
721     logger.info('Input parameters: ')
722     logger.info(' '.join(f'{k}={v}' for k, v in vars(args).items()))
723     actor, critic, actor_optim, critic_optim, iteration =
724         init(args.encoder, args.id, args.n_frame, args.recurrent)
725     logger.info(f'iteration {iteration}')
726     while(iteration < TRAINING_ITERATION - 1):
727         logger.info(f'Start training at epoch: {iteration}
```

```
727     try:
728         actor, critic, actor_optim, critic_optim,
729             iteration = init(args.encoder, args.id,
730                 args.n_frame, args.recurrent)
731         train_model(actor, critic, actor_optim,
732             critic_optim, iteration, args.port, args.
733             encoder, args.id, args.p2, args.recurrent,
734             args.n_frame)
735     except Exception as ex:
736         print(ex)
737         logger.error("Error occurred while collecting
738             trajectories data, restarting")
```

付録.D テストのソースコード

付録.D-1 PvJ.Py

```
1 import sys
2 sys.path.append('..')
3 from time import sleep
4 from py4j.java_gateway import JavaGateway,
5     GatewayParameters, CallbackServerParameters, get_field
6 from fight_agent import SoundAgent
7 logger = logging.getLogger(__name__)
8 def check_args(args):
9     for i in range(argc):
10         if args[i] == "-n" or args[i] == "--n" or
11             args[i] == "--number":
12             global GAME_NUM
13             GAME_NUM = int(args[i+1])
14
15 def start_game(Character):
16     for Chara in Character:
17         # FFT GRU
18         for i in range(30):
19             gateway = JavaGateway(gateway_parameters=
20                 GatewayParameters(port=4242),
21                 callback_server_parameters=
22                 CallbackServerParameters());
23             manager = gateway.entry_point
24             ai_name = 'FFTGRU'
25             #manager.registerAI(ai_name, SoundAgent(
26                 gateway, logger=logger, encoder='trans',
27                 path='E:\\DareFightingICE\\FightingICE\\'
28                 'DareFightingICE\\SampleAI\\BlindAI\\'
29                 'ppo_pytorch\\checkpoints\\fft\\rnn\\'
30                 'rnn_1_frame_256_mctsai\\166', rnn=True,
```

```
    device="cuda:0"))

23 manager.registerAI(ai_name, SoundAgent(
    gateway, logger=logger, encoder='trans',
    path='E:\\DareFightingICE\\FightingICE\\
          DareFightingICE\\SampleAI\\BlindAI\\
          ppo-pytorch\\checkpoints\\trans\\rnn\\
          rnn_1-frame_256_mctsai\\2024', rnn=True,
    device="cuda:0"))

24 print("Start game")

25 game = manager.createGame(Chara, Chara,
    ai_name, "MctsAi65", GAME_NUM)

26 manager.runGame(game)
27 print("After game")
28 sys.stdout.flush()
29 close_gateway(gateway)

30

31 def close_gateway(g):
32     g.close_callback_server()
33     g.close()

34

35 def main_process(Chara):
36     check_args(args)
37     start_game(Chara)

38

39 args = sys.argv
40 argc = len(args)
41 GAME_NUM = 1
42 Character = ["ZEN"]

43

44 main_process(Character)
```

付録.D-2 fight_run.py

```
1 import logging
2 import time
3 import sys
4 from py4j.java_gateway import JavaGateway,
5     GatewayParameters, CallbackServerParameters, get_field
6 logger = logging.getLogger(__name__)
7 gateway = JavaGateway(gateway_parameters=
8     GatewayParameters(port=4242),
9     callback_server_parameters=
10    CallbackServerParameters()
11   )
12
13 manager = gateway.entry_point
14 current_time = int(time.time() * 1000)
15 encoder1 = 'conv1d'
16 encoder2 = 'mel'
17 encoder3 = 'fft'
18 encoders = ['conv1d']
19 # register AIs
20 # collect_data_helper = CollectDataHelper(logger)
21 agent1 = SoundAgent(gateway, encoder=encoder1, logger=
22 logger)
23 agent2 = SoundAgent(gateway, encoder=encoder2, logger=
24 logger)
25 agent3 = SoundAgent(gateway, encoder=encoder3, logger=
26 logger)
27
28 manager.registerAI(f'SoundAgent_{encoder1}', agent1)
29 manager.registerAI(f'SoundAgent_{encoder2}', agent2)
30 manager.registerAI(f'SoundAgent_{encoder3}', agent3)
31 # game = manager.createGame('ZEN', 'ZEN', f'SoundAgent_{
32     encoder1}', 'MctsAi65', 1)
33 # game = manager.createGame('ZEN', 'ZEN', f'SoundAgent_{
34     encoder2}', f'SoundAgent_{encoder3}', 3)
35 game = manager.createGame('ZEN', 'ZEN', f'SoundAgent_{'
```

```
encoder1} , 'MctsAi65' , 3)  
26  
27 # start game  
28 manager.runGame(game)  
29  
30 # finish game  
31 logger.info('Finish game')  
32 sys.stdout.flush()  
33  
34 # close gateway  
35 gateway.close_callback_server()  
36 gateway.close()  
37 error = False
```

付録.D-3 fight_agent.py

付録.E 結果分析のソースコード

付録.E-1 visualize.py

```
1 import argparse
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import seaborn as sns
5 from sklearn.linear_model import LinearRegression
6 def draw_curve(file, title):
7     with open(file, 'r') as f:
8         data_raw_mean = f.readlines()
9     # with open('std_fft_0.txt', 'r') as f:
10    #     data_raw_std = f.readlines()
11    data_raw_mean = np.array([float(u.replace('tensor( ','',
12        ')').replace(')', '').strip()) for u in
13        data_raw_mean])
14    x = [i for i in range(len(data_raw_mean))]
15    model = LinearRegression()
16    model.fit(np.expand_dims(x, 1), data_raw_mean)
17    plt.plot(data_raw_mean)
18    plt.plot(x, model.predict(np.expand_dims(x, 1)), color
19              ='k')
20    plt.title(title + ' ({})'.format(int(data_raw_mean
21        [-1] - data_raw_mean[0])))
22    plt.xlabel('Epoch')
23    plt.ylabel('HP Difference')
24    plt.show()
25    print()
26 def draw_smooth(file, title):
27     with open(file, 'r') as f:
28         data_raw_mean = f.readlines()
29     data_raw_mean = np.array([float(u.replace('tensor( ','',
30        ')').replace(')', '').strip()) for u in
31        data_raw_mean])
32     data = []
```

```
27     for i in range(len(data_raw_mean) - 5):
28         data.append(sum(data_raw_mean[i: i+5]))
29     x = [i for i in range(len(data))]
30     model = LinearRegression()
31     model.fit(np.expand_dims(x, 1), data)
32     plt.plot(data)
33     plt.plot(x, model.predict(np.expand_dims(x, 1)), color
34             = 'k')
35     y1 = model.predict(np.expand_dims(x[-1], 1))
36     y2 = model.predict(np.expand_dims(x[0], 1))
37     plt.title(title + ' ({})'.format((y1 - y2)[0]))
38     plt.xlabel('Epoch')
39     plt.ylabel('HP Difference')
40     plt.show()
41     print()
42 def draw_polynomial(file, title, degree=4):
43     from sklearn.preprocessing import PolynomialFeatures
44     with open(file, 'r') as f:
45         data_raw_mean = f.readlines()
46     data_raw_mean = np.array([float(u.replace('tensor(',
47         '')).replace(')', '').strip()) for u in
48         data_raw_mean])
49     data = data_raw_mean
50     x = [i for i in range(len(data))]
51     # poly
52     poly_features = PolynomialFeatures(degree=degree)
53     X_train_poly = poly_features.fit_transform(np.
54         expand_dims(x, 1))
55     poly_model = LinearRegression()
56     # print(np.expand_dims(X_train_poly[0, :, 1]))
57     poly_model.fit(X_train_poly, data)
58     plt.plot(x, data, label='Learning curve')
59     plt.plot(x, poly_model.predict(X_train_poly), label='
60             Polynomial Regression')
61     y1 = poly_model.predict(np.expand_dims(X_train_poly
62         [-1, :, 1], 0))
63     y2 = poly_model.predict(np.expand_dims(X_train_poly
```

```
[0,:],0))
58     i = np.poly1d(poly_model.predict(X_train_poly)).integ
      ()
59     print('Area under the learning curve is:', i(1) - i
      (0))
60     # plt.title(title + ' ({})'.format((y1 - y2)[0]))
61     plt.legend()
62     plt.title(title)
63     plt.xlabel('Epoch')
64     plt.ylabel('HP Difference')
65     plt.show()
66
67 # draw_polynomial(
       reward_fft_rnn_1_frame_256_mctsai65_rnn.txt", 'FFT',
       4)
68
69 if __name__ == '__main__':
70     parser = argparse.ArgumentParser()
71     parser.add_argument('--file', type=str, required=True,
       , help='The result file')
72     parser.add_argument('--title', type=str, required=
       True, help='Title of the plot')
73     parser.add_argument('--degree', type=int, default=4,
       help='Polynomial degree')
74     args = parser.parse_args()
75     print('Input parameters: ')
76     print(' '.join(f'{k}={v}' for k, v in vars(args).
       items()))
77     draw_polynomial(args.file, args.title, args.degree)
```

付録.E-2 analyze_fight_result.py

```
1 import argparse
2 import pandas as pd
3 import os
4
5 def calculate(path):
6     files = [path + '\\\\' + f for f in os.listdir(path)]
7     dfs = []
8     for f in files:
9         df = pd.read_csv(f, header=None)
10        dfs.append(df)
11    data_all = pd.concat(dfs, axis=0, ignore_index=True)
12    data_all.columns = ['Round', 'P1', 'P2', 'Time']
13    win_ratio = sum(data_all.P1 > data_all.P2) / data_all
14        .shape[0]
15    hp_diff = sum(data_all.P1 - data_all.P2) / data_all.
16        shape[0]
17    return win_ratio, hp_diff
18
19 if __name__ == '__main__':
20     parser = argparse.ArgumentParser()
21     parser.add_argument('--path', type=str, required=True,
22         help='The directory containing result log')
23     args = parser.parse_args()
24     win_ratio, hp_diff = calculate(args.path)
25     print('The winning ratio is:', win_ratio)
26     print('The average HP difference is:', hp_diff)
```