深層強化学習を使用した
エッジコンピューティングのための
動的オフロードメカニズムの開発

井手　慎太郎

九州産業大学大学院
情報科学研究科情報科学専攻

令和４年１月

目次

図目次

表目次

個人文献

Proceedings

1. Shintaro Ide, Bernady. O. Apduhan, "Development of an RL-based Mechanism to Augment Computation Offloading in Edge Computing", 21st International Conference on Computational Science and its Applications (ICCSA 2021) July 5-8, 2021, Cagliari, Italy(Online)

2. 井手　慎太郎, アプドゥハン・ベーナディ, “エッジコンピューティングにおける RL ベースの効率的なタスクオフロードと割り当てに向けたメカニズムの開発,” IEICE Technical Committee on Computer Systems (CPSY), Hot Spring Annual meeting 2021 (HotSPA2021), 電子情報通信学会, 福岡県 (Online)

第1章　序論

1.1　研究背景

過去 10 年間で，移動通信システムは第 3・4 世代（3G・4G）から今日の第 5 世代（5G）へと進化を遂げている．現在家電やウェアラブルデバイス，工場のライン整備に幅広く IoT（Internet of Things）が活用されており、今後 5G ネットワークが社会に普及していくことで，自動車の自動運転やリアルタイム監視システム，スマートホームなど，よりデータ量が大きく，リアルタイム性が要求されるような利用シナリオが増加することが予測されている [1].

現在，IoT デバイス・アプリケーションから生成される大量のデータを処理するための技術として，一般的にクラウドコンピューティングが利用されている．クラウドコンピューティングは，ソフトウェアやアプリケーションをユーザが必要な時に必要な分だけネットワーク経由で利用できるサービスである．従来のオンプレミス型の利用形態と異なり，独自にサーバをデプロイする必要がないため，運用コストなどが大幅に削減されるといった利点がある．しかし，大規模なクラウドサーバ（クラウドデータセンタ）で集中的にエンドユーザから受信したデータを処理するクラウドコンピューティングの処理形態は，近年の IoT デバイス・アプリケーションから生成される大容量データに対して高速に処理し，送受信を行うことは難しい．

そこで，通信の遅延（レイテンシ）を最小化する技術として注目を浴びている技術としてエッジコンピューティングがある．ユーザデバイスから生成されたデータの保存や処理を，クラウドサーバではなく，各地に分散配置された物理サーバ（エッジサーバ）もしくは計算能力を保持したユーザデバイス自身が行うことで，ユーザデバイスとクラウドサーバ間で生じたレイテンシを最小限に抑えることができる．他にも，データをエッジで処理するため，クラウドサーバのストレージ容量の節約やセキュリティ性の向上といった利点が存在する．

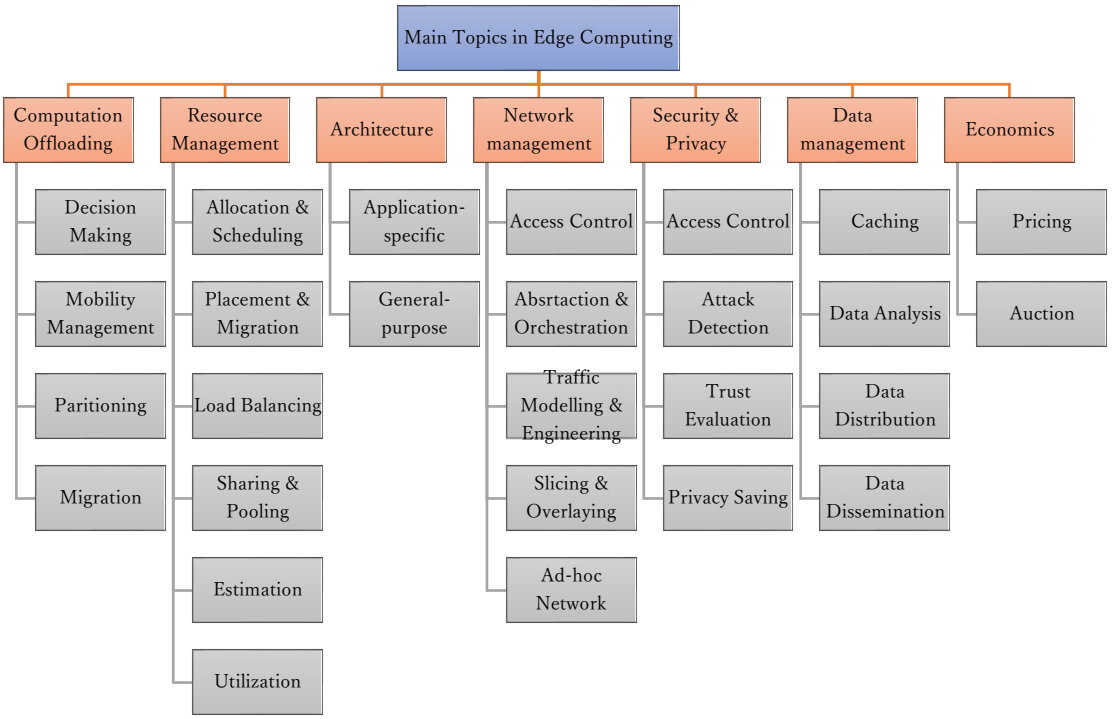


図 1. Edge Computing Research Tree [2]

その一方で，エッジコンピューティングにも現状解決すべき課題点が数多く存在する．エッジコンピューティングに関する研究は 20 年ほど前から取り組まれていたが，先述した IoT 文化の広がりに比例して，近年この研究分野の人気は高まり，図 1 に示すように数多くの研究分野で構成され，より活発に研究が行われるようになった [2]．図 2 はエッジコンピューティングにおける各分野の論文件数の経年変化を示している．全ての分野において論文や研究発表の数は増加傾向に見られるが，中でも Resource Management と Offloading の分野は，他と比較して注目度が高いことが分かる．これは，エッジコンピューティングにおける有限のリソースをどのように管理すべきか検討することが非常に重要であることを示唆している．また，計算オフロードの分野においても，モビリティ管理やパーティショニング，意思決定問題，マイグレーションなど複雑で多様な課題を抱えていることから，研究の活発化に繋がっていると考えられる．

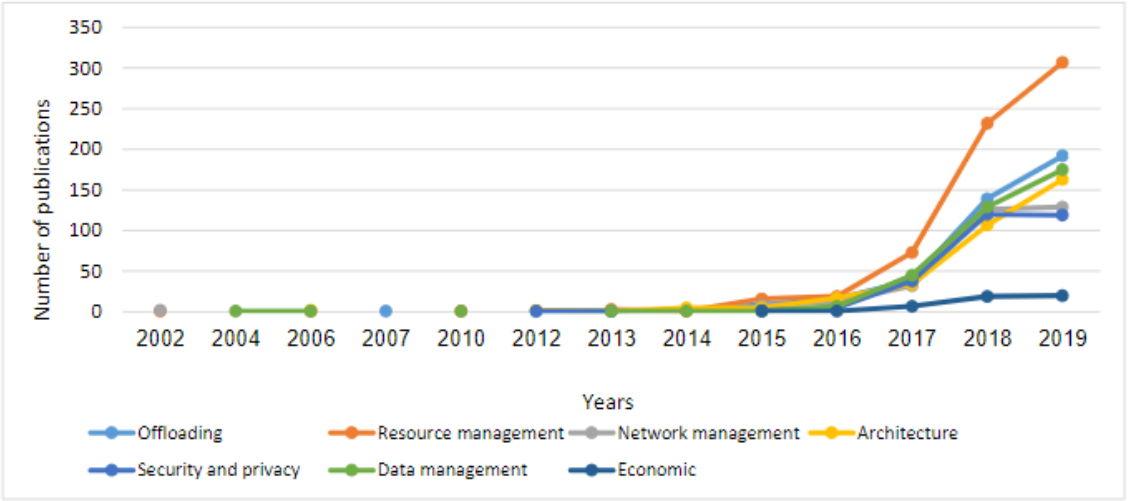


図 2. 各分野における論文出版件数の推移 [2]

しかし，エッジコンピューティングでの計算オフロードに関する研究において，その多くは最適なオフロード決定手法として多次元ナップサック問題などの数理モデルに置き換え，静的なシステムとして提案されることが多い．他にも教師あり学習などの機械学習手法を取り入れた研究も存在するが，学習データ不足などの影響でその活動はあまり活発でない．

1.2 研究の目的

本研究の目的は，以下のように大別できる．

- 現在のエッジコンピューティングが抱える問題点について調査する．特に，計算オフロードに関する技術的課題点を明らかにする．
- 従来手法である静的な計算オフロードシステムや，関連手法である DQN，Q 学習オフロードシステムについて調査し，その課題点を克服できる手法について検討する．
- 提案手法が環境内で正常に動作するか，ランダムに生成される疑似データを使用した予備実験を行うことで評価する．
- 実際のモビリティデータセットを使用したメカニズムの動作検証を行う実験を行うことで，提案手法の有効性について評価する．

これにより，エッジコンピューティングでの計算オフロードシステムに対して，より複雑な意思決定問題を導入することが可能となり，計算資源や帯域幅の利用効率が向上することで，さらに高品質なサービス品質の提供を期待できる．

1.3　研究方法

本研究の基本的な要件を理解するために，初めに理論研究を行った．これは，様々な専門書，国内外の研究論文，インターネット上の記事などの調査が含まれる．また，強化学習アルゴリズムや深層強化学習アルゴリズムに関する関連事例や文献について調査し，各アルゴリズムのパフォーマンス分析や動作原理についてサンプルシミュレータを使用して検証した．

その後，理論研究から得られた結果から本研究に有効なアルゴリズムについて検討し，第 3 章で述べるエッジコンピューティングシステムモデルに対して提案手法が有効であるか評価する実験を行った．

1.4　本論文の構成

本論文では，以下の構成をとる．まず，第 2 章で本研究に関連している強化学習や深層強化学習について述べ，第 3 章でシステムモデルと提案手法について述べ，第 4 章および第 5 章で評価実験・考察を行い，第 6 章で本研究のまとめと今後の課題について述べる．

次の図 3 は，本論文の構成と各章の概要を示したものである．

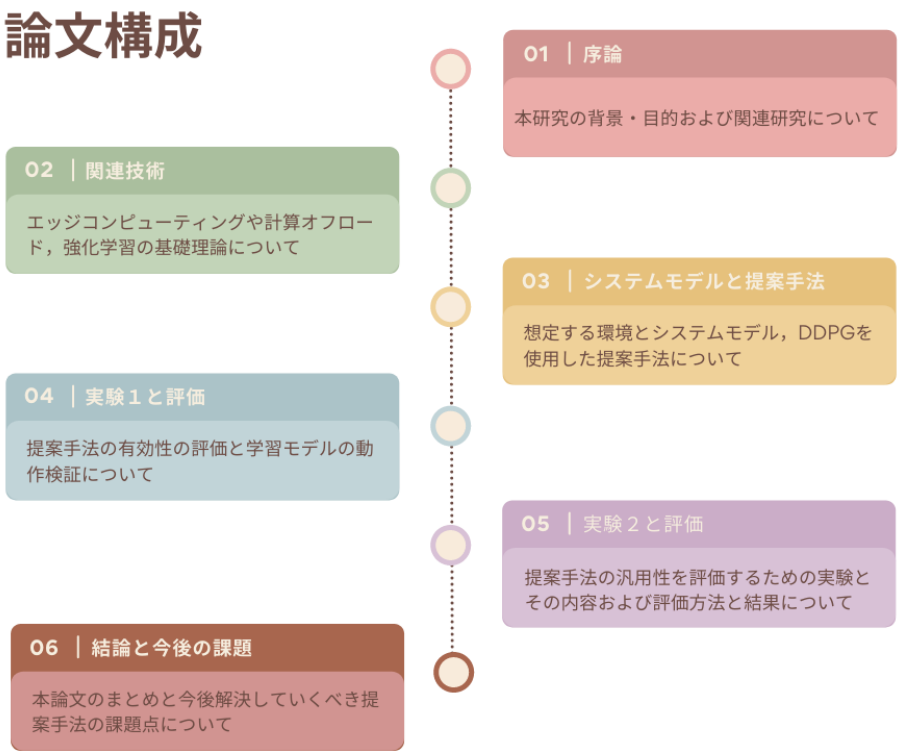


図 3. 本論文の構成

1.5 関連研究

効率的な計算オフロードを行うためには，タスクを「いつ」「どこで」「どの程度」オフロードすべきかを考慮する必要がある．このような異なるシナリオに対する最適な計算オフロード戦略については，既にいくつかの研究が行われている．

Gao ら [3]は，モバイルデバイス，エッジクラウド，クラウドデータセンタからなる協調コンピューティングシステムを開発し，それをベースにした最適なタスク割り当てとオフロードのための Q-Learning ベースの計算オフロードポリシーを設計した．具体的には，まず，オフロード決定プロセスをマルコフ決定過程としてモデル化し，経験の質(QoE)を測定するための状態損失関数(STLF)を設計する．その後，STLF の累積をシステム損失関数(SYLF)と定義し，SYLF の最小化問題を定式化した．定式化した問題は直接解くことが困難であったため，複数のサブ問題として分解し，エッジサーバと送信電力の計算周波数をそれぞれ最適化する．予め計算されたオフライン伝送電力とエッジサーバの計算周波数に基づいて，オフロード決定を最適化することで SYLF を最小化する Q-learning based offloading(QLOF)スキームを開発する．評価実験の結果，彼らの提案手法は，フルオフロードポリシーやフルローカルコンピューティングポリシーと比較して，システム全体の処理効率が増加し，SYLF を最小化するという結果を得ることができた．このことから，強化学習ベースの計算オフロードポリシーがエッジコンピューティング環境において非常に有効な手法であるという結論に達した．

Xu ら [4]は，地理的に険しい地域や未開発の地域におけるモバイルエッジコンピューティングのグリッド電源供給が実現困難であるという背景から，再生可能エネルギーを主要電源として利用する方法に着目した．しかし，再生可能エネルギーの高い間欠性と予測不可能性は，モバイルエッジコンピューティングシステムにおいて，ユーザに高いサービス品質を提供することを困難にしている．この問題を解決するために，Xu らは，効率的な強化学習ベースのリソース管理アルゴリズムについて提案した．具体的には，システムモデルを「Workload model」，「Delay cost model」，「Power model」，「Battery model」の 4 つのサブモデルに分類し，これらの長期的なシステムコストを最小化するために，クラウドデータセンタとエッジサーバ間の計算オフロードおよび資源割り当ての最適な方策を学習する．実験の結果，Q-Learning などの標準的な強化学習アルゴリズムと比較して，学習時間と実行時間が短縮し，エッジコンピューティングの性能が大幅に改善されることが示された．

これらの関連研究は，一般的なエッジコンピューティング環境に強化学習を使用した効率的な計算オフロード手法を導入することで，システムのレスポンスや電力消費などの面で大きな成果を上げていることが報告されている．しかしながら，これらの提案手法およびアルゴリズムの課題点として離散的な行動空間(電力消費量の割り当てや計算資源の割り当て)のみに対応している点や，エッジサーバとクラウドデータセンタとの距離によるレイテンシ問題を考慮していない点が挙げられる．エッジコンピューティング環境での限られた

計算資源や帯域幅を効率的に活用するためには，行動空間を連続値として定義し，タスクの種類やオフロード時の状況に応じて柔軟に値を決定できることが望ましい． また，リアルタイム性を要求するアプリケーションの高いサービス品質を維持するために，タスクの処理はクラウドデータセンタへアップロードせずに，エッジノードで処理を完結することが望ましいと考えられる．

第2章　関連技術

2.1　クラウドコンピューティング

インターネットをベースにしたクラウドコンピューティングは，最も強力な計算アーキテクチャを保有している．これは，統合されネットワーク化されたハードウェア，ソフトウェア，インターネットインフラの集合体を意味する．グリッドコンピューティングやその他のコンピューティングシステムに対しても様々な利点がある．

クラウドコンピューティングのクラウドとは，実際の雲が水分子の集合体であるように，ネットワークの集まりを意味している．ユーザは必要に応じて，クラウドコンピューティングのリソースを自由に利用することが可能である [5]．リソースはクラウドを形成するネットワーク上で処理されるため，アプリケーションの実行中にローカルコンピュータの負荷が重くなることがないという特徴を持つ．

2.1.1　クラウドコンピューティングの種類

クラウドコンピューティングは主に以下の 3 種類のシステムに大別できる．

(1) パブリッククラウド

プロバイダなどの第三者がインターネット上で提供しているコンピューティングサービスを指す．これらのサービスは誰でも利用が可能であり，ユーザは利用したサービスに対してのみ料金を支払う必要がある．

当初は，クラウドサービスといえばパブリッククラウドを指す場合が一般的であったが，プライベートクラウドの提供が一般化された影響で，両者を区別する意味でパブリッククラウドの表現が使用されるようになった．

(2) プライベートクラウド

プライベートネットワーク上で提供されるコンピューティングサービスを指す．これらのサービスは，許可されたユーザにのみ提供され，一般の人々には公開されない．プライベートクラウドでは，ファイアウォールと内部ホスティングを通じて，より高いセキュリティとプライバシーが委ねられる．

また，プライベートクラウドの能力が不足している場合などに，パブリッククラウドから一時的に処理能力を借りたり，両クラウドで動作するアプリケーションやサービス同士の連携などを行うことも可能である．

(3) ハイブリッドクラウド

パブリッククラウドとプライベートクラウドを組み合わせたものを指す．ハイブリッドクラウドでは，各クラウドは独立して管理ができるが，データやアプリケーションはハイブリッドクラウド全体のクラウド間で共有できるといった特徴を持つ．

2.1.2 クラウドコンピューティングのメリット

クラウドコンピューティングのメリットとして主に以下のようなものが挙げられる．

(1) 柔軟性

ビジネスの急速な拡大あるいは縮小に伴い，企業はハードウェアやリソースを迅速に調整する必要が多々ある．クラウドコンピューティングはこのような変化に迅速に対応できる柔軟性を備えている．

(2) コスト削減

クラウドコンピューティングを利用するユーザは，利用したサービスに対しての料金のみ支払えば良いため，自身でインフラを購入・整備する必要がない．そのため，メンテナンスコストを最小限に抑えることができる．

2.1.3 クラウドコンピューティングのデメリット

クラウドコンピューティングおよびクラウドサービスは現在の IT 業界のトレンドとなっており，クラウドを利用したシステムを構築するユーザが急速に増加している．一方で，クラウドを利用するにあたって，デメリットが気になるユーザも少なくない．

クラウドコンピューティングのデメリットとして，主に以下のようなものがある．

(1) カスタマイズに関する柔軟性が低い

クラウドサービスは，オンプレミス型に比べてカスタマイズに関する柔軟性が低く，サービスを提供するプロバイダの仕様を超えたカスタマイズなどは基本的に不可能である．

(2) サービスの安定性が低い

クラウドサービスは，ネットワーク障害などの影響でサービスが一時的に停止する可能性がある．サイバー攻撃などで構築したシステムにダメージを受け，クラウドサービスの提供が中止される可能性もあるため，クラウドサービスを導入する際はプロバイダやセキュリティを慎重に検討し契約する必要がある．

(3) インターネット環境の影響を受けやすい

クラウドは，インターネットを介したサービスのため，インターネット環境の影響を受けざるを得ない．日常的に利用しているシステムのトラフィックが増加すれば，ネットワーク回線への負荷が増大するため，通信速度が遅くなる可能性もある．

2.2 エッジコンピューティング

エッジコンピューティングは，IoT デバイスやローカルエッジサーバなどのデータソースにアプリケーションを物理的に近づける分散型フレームワークである．これにより，5 G 技術が目指すレイテンシや帯域幅のボトルネックを解消することが期待されている．

2.2.1 エッジコンピューティングの需要

近年では，IoT デバイスの爆発的な普及と必要な演算能力の向上により，世界中でこれまでにない大量のデータが発生している．また，5G ネットワークによって接続されるユーザデバイスの数はさらに増加し，それに伴って発生するデータ量も増大することが予想される．

かつて，クラウドや AI は，データから実用的な知見を導き出すことで，イノベーションを自動化することが期待されていたが，大量のユーザデバイスによって生み出される前例のない規模と複雑なデータは，ネットワークのインフラの能力を上回っている．

従来のクラウドコンピューティングは，ユーザデバイスで生成されたすべてのデータやタスクを中央のデータセンタやクラウドに送信していたが，帯域幅やレイテンシの問題が発生していた．IoT アプリケーションの中には，大量のデータをリアルタイムに処理する必要がある場合も多い．

エッジコンピューティングはそれらの需要に応えるより効率的で有用な代替手段として提供された新たなフレームワークである．

2.2.2 エッジコンピューティングのメリット

エッジコンピューティングにおいて，データあるいはタスクは，生成された場所の近くで処理と分析が行われる．処理・分析を行うためにネットワークを介してクラウドやデータセンタに移動する必要がないため，サーバまでの物理的な距離が短縮され，レイテンシが大幅に削減される．

2.2.3 既存のエッジコンピューティングの課題点

クラウドコンピューティングにおける課題点を克服するための新たな分散フレームワークとして注目されるエッジコンピューティングだが，効率的に運用し，高品質のサービスを継続的に提供するためには，各エッジサーバの計算能力について調査し，効率的な負荷分散手法について検討する必要がある．

一般的に，エッジコンピューティングに利用されるエッジサーバは，クラウドやデータセンタに設置されているサーバと比較して非常に計算能力が低い．そのため，ユーザデバイスからの処理要求が単一のエッジサーバへ送信されるなどの場合において，要求を受けたエッジサーバはその負荷に対応できず，大幅な処理遅延や処理の中止といった現象が起こる可能性が考えられる．

2.3 計算オフロード（Computation Offloading）

2.2.3 で述べたエッジコンピューティングにおける課題点を解決するための 1 手法として考えられるのが，計算オフロードである．

計算オフロードは，リソースを大量に消費する計算タスクやデータを，ハードウェアアクセラレータなどの別のプロセッサやクラスタ，グリッド，クラウドなどの外部プラットフォームに転送する技術をいう．計算オフロードは，例えばプロセッサへオフロードする場合，画像レンダリングや数学的計算などのアプリケーションを高速化するために使用できる．ネットワークを介して外部プラットフォームに計算資源をオフロードすることで，計算能力を提供し，デバイスのハードウェアの制限を克服するといった用途にも利用される [6]．

一般的に，計算オフロードはモバイルコンピューティングの分野で活発に研究が行われているが，ほとんどの研究がワークロードをクラウドへオフロードしている [7] [8] [9]．これは，Vertical Scaling またはスケールアップなどと呼ばれる一種のスケーラビリティであると言える．

スケールアップとは，図 4 に示すように，サーバの CPU やメモリなどの能力を増強し，処理性能を向上させる方法をいう．また，スケールアップと対照的な方法としてスケールアウトがある．

スケールアウトも同様，スケーラビリティの一種であり，Horizontal Scaling とも呼ばれる．サーバ自身の能力を増強するのではなく，サーバの台数を増やすことでシステム全体の性能を向上させる [10]．

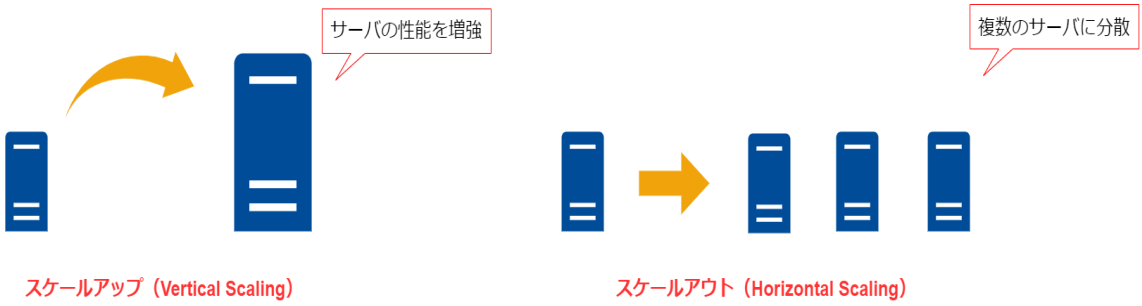


図 4. スケールアップ（Vertical Scaling）とスケールアウト（Horizontal Scaling）

エッジコンピューティングを含む一般的なモバイルコンピューティングでは，ユーザデバイスやエッジサーバに負荷が集中した場合，スケールアップの手法が採用され，より計算能力の高いクラウドやデータセンタへ処理がオフロードされる．しかしながら，IoT アプリケーションなどから生成されるデータやタスクは，リアルタイム性を要求するものが多い．地理的に離れたクラウドへ処理をオフロードする場合には大幅なレイテンシが発生し，サービス品質に影響を及ぼす可能性がある．

2.4 機械学習

機械学習は，人工知能（AI）やコンピュータサイエンスの分野で，データやアルゴリズムを用いてコンピュータが自律的に学習を行い，データの背景にあるルールやパターンを発見するデータ分析手法をいう．機械学習の類義語として，人工知能（AI）や深層学習（Deep Learning）があるが，人工知能を実現するためのデータ分析技術の一つとして機械学習が存在し，機械学習における代表的な分析手法が深層学習であると言われている．以下図 5 にこれらの相互関係について示す．

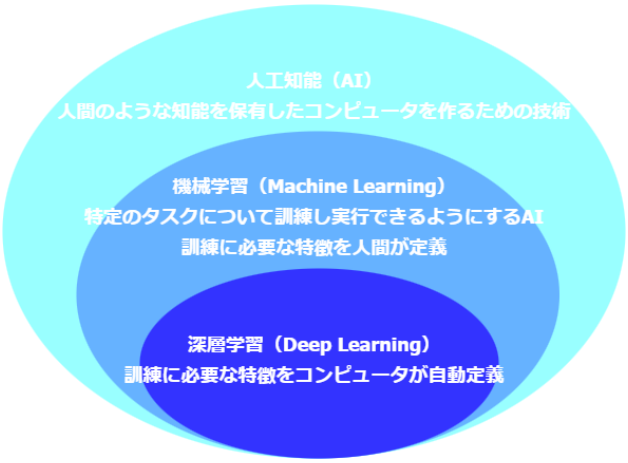


図 5. 人工知能・機械学習・深層学習の相互関係

また，機械学習は以下の 3 種類の学習手法に大別することができる．

(1) 教師あり学習（Supervised Learning）

教師あり学習は，ラベル付けされた訓練データからモデルを学習し，未知のデータや将来のデータを予測する学習手法である．「教師あり」とは，望ましい出力信号（ラベル）がすでに判明している訓練データの集合を意味する．

教師あり学習の利用用途は主に分類と回帰の 2 種類である．分類は，過去の観測に基づき，新たなインスタンスを対象として，クラスラベルを予測することが目標となる．クラスラベルとは，離散的で順序性のない値を意味する．

分類の利用例として，スパムメール分類や手書き文字認識などが挙げられる．スパムメール分類は，モデルが入力されたメールに対してスパムか否かを判断する 2 値分類問題である．一方手書き文字認識は，ユーザが入力デバイスなどを使用して入力した手書き文字に対して，数多く存在する文字の中から正しい文字を一定の正解率で予測し，入力された文字の結果を返す多値分類問題である．

回帰の利用例として，試験を受験した学生の点数予測などが挙げられる．これは，試験勉強に費やした時間と最終的な点数との間に関係があるとすれば，それらを訓練データとして利用することで，今後同様の試験を受験する学生に対して点数を予測するモデルが学習できる．このような問題は一般的に回帰分析問題と呼ばれ，複数の特徴量と連続値の目的変数が与えられ，結果を予測できるようにそれらの関係を探ることで，モデルが学習を行う．

(2) 教師なし学習（Unsupervised Learning）

教師なし学習では，ラベル付けされていないデータや構造が不明なデータを扱う．教師あり学習や後述する強化学習と異なり，目的変数や報酬関数がなくても，データの構造を調査して意味のある情報を取り出すことができる．

教師なし学習の利用用途は主にクラスタリングによるグループの発見とデータ圧縮のための次元削減の 2 種類である．クラスタリングは，大量のデータを意味のあるグループ（クラスタ）として構造化できる探索的データ解析手法である．教師なし学習を用いたクラスタリングの例として，マーケティングプログラム開発における顧客の関心に基づいた顧客集団の発見戦略などがある．

次元削減は，機械学習アルゴリズムを実行する際の記憶域や計算能力を節約するために，特徴量の前処理の段階で使用されるアプローチの一つである．データからノイズを取り除き，関連する大半の情報を維持したうえで，データをより低い次元の部分空間に圧縮する．

(3) 強化学習（Reinforcement Learning）

強化学習は，与えられた環境内で行動を起こす主体（エージェントと呼ばれる）が試行錯誤し，環境から与えられる報酬を基に行動を最適化していく理論的枠組みである．強化学習の利用例として有名なものが，Google 社の関連会社であるディープマインド社が開発した AlphaGo と呼ばれる囲碁プログラムである．一部の学習を人間の囲碁のプロによる教師あり学習が行っているものの，基本的には自己対戦による強化学習で最適化を行い，非常に高い学習成果を出すことに成功し，世界トップ棋士にも勝利するという結果を残した．

ゲーム内における最適行動関連問題以外にも，自動車の自動運転技術や通信ネットワーク経路の最適化など実社会における様々な分野に応用ができる手法として現在注目されている．本研究においても，提案手法を実現するための機械学習手法として，この強化学習を採用している．強化学習の詳細については後述の 2.5 で述べる．

2.5　強化学習

強化学習では，報酬関数を最大化するためにモデルが環境とのやり取りから学習を行う．報酬関数の最大化は，教師あり学習のコスト関数の最小化の概念に関連しているが，強化学習は，一連の行動を学習するための正解ラベルは事前に定義されていない．ユーザにとって望ましい結果を得るためには，環境とのやり取りによって一連の行動について学習していく必要がある．強化学習のモデルは一般的にエージェントと呼ばれる．エージェントはその環境とやり取りすることで，エピソードと呼ばれるシーケンスを生成し，エピソードを通じてその環境が決定する一連の報酬を収集する．図 6 は，強化学習の基本的な枠組みを示している．

強化学習では，エージェントに対して達成すべきゴールのみを指示として与え，その後のエージェントの行動に応じて報酬を決めるため，複雑な環境での意思決定問題に取り組む際に非常に有効な手法と言える．問題解決に一連の手順が要求され，それらの手順が不明あるいは定義することが難しい場合は特に効果が期待できる．

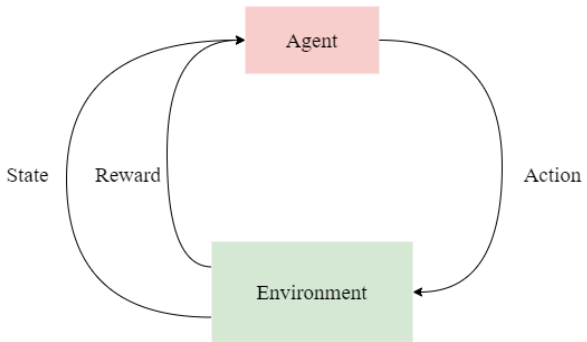


図 6．強化学習の基本的なフレームワーク

2.5.1　強化学習モデル

強化学習モデルの基本的なフレームワークは図 6 の通りだが，実際に学習を行う場合にはこれらの要素に加え，他にも様々な要素について定義する必要がある．強化学習モデルに必要な要素は表 1 に示す通りである．

表 1．強化学習モデルに必要な要素

	記号	備考
状態空間	$S = \{s_0, s_1, \dots, s_M\}$	環境の取りうる状態の集合
行動空間	$A = \{a_0, a_1, \dots, a_N\}$	エージェントが取りうる行動の集合
報酬関数	$R_{t+1} = r(s_t, a_t, s_{t+1})$	ある状態s_tから行動a_tを行い，次の状態s_{t+1}へ遷移したときに環境から得られる報酬
状態遷移確率	$P(s_{t+1}\|s_t, a_t)$	ある状態s_tにて行動a_tを実行した後に次の状態s_{t+1}へ遷移する確率
エピソード時刻	$0 \dots t \dots T$	

表 1 から，状態遷移確率$P(s_{t+1}|s_t, a_t)$は，次の状態に遷移する確率を返す関数である．モデルに関して，状態遷移確率$P(s_{t+1}|s_t, a_t)$と報酬R_{t+1}が分かっていることを前提にした学習モデルをモデルベース，分からないことを前提にした学習をモデルフリーという．

次に，ある時刻tから最終エピソードまでの報酬を式(1)のように定義する．

$$r_t + r_{t+1} + \cdots + r_T \tag{1}$$

tは時刻を表しているため，$t = 0$とすれば 1 エピソードで獲得した全報酬を示すことになる．しかし，ある時刻tから見た未来の報酬には不確定要素が入るため，実際の報酬は未来の報酬に割引率γを適応することになる．この割引率には一般的に 0.9 などの 1 に近い定数を与える．未来の報酬に割引率を適応したものを割引報酬G_tと呼び，式(2)のように定義する．

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T \tag{2}$$

この割引報酬G_tは価値とも言われ，強化学習の学習を行う上で重要な値となる．

2.5.2 価値

強化学習では，ある時刻tにおいて次にどのような行動を選択するべきかを考える．これは，選択可能な全ての行動で，行動後の遷移した状態の価値G_{t+1}が最大となるものを選択すれば良いと考えられる．しかし，G_{t+1}を求めるためには未来で獲得できる報酬について計算する必要がある．未来の報酬の計算には行動後の状態が確率的に遷移することから，実際に行動してみないと求めることが困難であることが分かる．そこで，G_{t+1}を求める代わりに将来獲得できる報酬の期待値を求めることを考える．期待値を求めることで，状態の良し悪しを比較することができ，より良い状態へ遷移することが可能となる．

期待値を用いて，将来の価値を予測するために，ここで方策πを定義する．方策πはある状態sにおいて行動aを選択する確率を意味し，$\pi(a|s)$のように定義する．また，方策πにおいて，ある状態sの価値$V_\pi(s)$は式(3)のように定義する．

$$V_\pi(s_t) = E_\pi[r_{t+1} + \gamma V_\pi(s_{t+1})] \tag{3}$$

ここで，Eを期待値，r_{t+1}を報酬，γを割引率とする．

式(4)は一般的にベルマン方程式と呼ばれ，強化学習アルゴリズムの価値推定手法のベースとなっている．強化学習では大きく分けて，各状態の価値を学習する方法と方策πを学習する方法がある．また，価値と方策の両方を学習する Actor-Critic という方法もあり，本研究ではこの Actor-Critic をベースとしたアルゴリズムを使用しているため，価値ベースと方策ベースの両方の学習方法について説明する．

2.5.3 価値関数と方策関数

価値関数とは，その名の通り状態あるいは行動の価値を推定する関数である．価値関数には，状態価値関数と行動価値関数の 2 種類が存在する．

状態価値関数は，状態sにいることの価値を意味し，2.5.2 の式(4)で定義したベルマン方程式で求められる$V_\pi(s)$となる．行動価値関数$Q_\pi(s,a)$は，状態sにいて行動aを取ることの価値を意味し，式(4)のように定義する．

$$Q_\pi(s_t, a_t) = \sum_{s'} T(s_{t+1}|s_t, a_t)\big(r_{t+1} + \gamma V_\pi(s_{t+1})\big) \tag{4}$$

価値関数Vが既知の場合，方策関数πを決めることで期待値を求めることができる．この期待値が最大となるような方策を学習する手法が方策ベースの学習であり，一般的にオンポリシーな学習と呼ばれる．反対に，価値関数を学習する手法は，方策がないという意味からオフポリシーな学習と呼ばれる．

2.5.4 TD 法とモンテカルロ法

強化学習で価値を計算する場合, 式(4)の$V_\pi(s_{t+1})$のような未来の価値について知る必要がある．これを計算する方法として，予測値で計算する方法と実測値で計算する方法がある．

予測値で計算する方法は TD 法と呼ばれ, 1 ステップ毎にアルゴリズム内の予測値を用いて価値関数の更新を行う．

ここで，実際に TD 法を用いた価値関数の更新を考えてみる．現在の価値を予測したときのある状態での価値を$V(s_t)$と定義する．次に，実際に 1 ステップ進めた時のある状態での価値を$V'(s_t)$と定義すると，式(5)のように定義できる．

$$V'(s_t) = r_{t+1} + \gamma V(s_{t+1}) \tag{5}$$

式(6)内の$V(s_{t+1})$が次の状態の価値を意味している．この値が推定値となり，現在の価値関数から予測した値となる．ここで，現在予測している価値と実際に得られた価値の差異である$V'(s_t) - V(s_t)$を TD 誤差と呼び，式(6)のように定義できる．価値関数はこの TD 誤差を基に式(7)のように価値を更新していく．

$$TD_{error} = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{6}$$

$$V(s_t) \leftarrow V(s_t) + \alpha(TD_{error}) \tag{7}$$

ここで，式(8)内のαは学習率を表している．学習率とは，機械学習の最適化においてどれほど値を動かすかを決定するハイパーパラメータである．学習率を大きくし過ぎると発散し，逆に小さくし過ぎると収束に時間がかかってしまう．

価値を計算するもう一つの方法として，実測値で計算するモンテカルロ法がある．これは，1 エピソード終了した後に 1 エピソードの情報を元に計算を行う．上記の式で，1 ステップ毎に$V(s_{t+1})$に対して TD 法では予測値を使用したが，モンテカルロ法ではエピソード最後まで展開する方法を取る. モンテカルロ法による TD 誤差の計算と更新式は式(8)(9)のように定義できる．

$$TD_{error} = (r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_{T-t}) - V(s_t) \tag{8}$$

$$V(s_t) \leftarrow V(s_t) + \alpha(TD_{error}) \tag{9}$$

2.5.5 Actor-Critic

Actor-Critic とは，TD 誤差を用いた初期の強化学習で使用されているアルゴリズムであり，2.5.3 で述べた価値関数と方策関数の両方を学習する手法である．状態価値関数の更新は式(10)のように，方策関数の更新は式(11)のように定義できる．

$$V(s_t) \leftarrow V(s_t) + \alpha\big(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big) \tag{10}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big) \tag{11}$$

式(11)(12)から，状態価値の TD 誤差を基に状態価値関数と方策関数を更新していることが分かる．

また，Actor-Critic を利用する主な利点として下記の 2 点がある．

(1) 行動選択に必要な計算量が最小限

連続値行動のような可能な行動選択が無限大であるときに，Q 学習などの場合は 1 つの行動を選択するために無限集合の中から探索を行うことになるが，Actor-Critic では，行動選択に最小限の計算量しか必要としない．

(2) 確率的な行動選択を学習可能

いろいろな行動に対して，その行動を選択するような最適確率を選択することができるため，確率的な行動選択を学習することができる．

2.5.6 Q 学習

Q 学習とは，モデルフリーの行動価値関数を求める強化学習アルゴリズムである．モデルフリーとは，環境が提供する状態遷移関数と報酬関数を使わずに学習を行うものをいう．モデルフリーのアルゴリズムでは，実際に行動する上で探索と活用のトレードオフを考える必要がある．Q 学習では$\varepsilon - greedy$法を使用して探索を行う．$\varepsilon - greedy$法とは，予め閾値として定義したハイパーパラメータεよりも小さい場合はランダムな行動を選択し，それ以外の場合は Q 値（行動価値関数）が最大の行動を選択する手法である．

Q 学習における Q 値の更新は式(12)のように定義できる．

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big(r_{t+1} + \gamma Q_{max}(s_{t+1}) - Q(s_t, a_t)\big) \tag{12}$$

式(12)から，$Q_{max}(s_{t+1})$が予測値となり，式(13)のようにs_{t+1}における各行動での最大値を返す．

$$Q_{max}(s_{t+1}) = \max_{a}\big(Q(s_{t+1}, a)\big) \tag{13}$$

2.5.7 決定論的方策勾配法（Deterministic Policy Gradient; DPG）

決定論的方策勾配法（以下 DPG）[13]は，連続行動空間を制御するために 2014 年に Silver らによって考案された強化学習アルゴリズムである．Actor-Critic なモデルを用いて行動価値関数と方策関数の両方を学習する．

従来の方策勾配法では，方策の出力に各行動の確率を出力していたのに対し，DPG では直接値（スカラー値）を出力するという特徴がある．このことから，DPG では方策に関して，従来の確率的な方策と比較して，決定論的方策と呼ばれている． DPG は方策を学習していくアルゴリズムのため, 方策勾配法と比較されることがあるが, 実際の学習手法は 2.5.6 で述べた Q 学習に似ている．そのため，DPG の学習方法について 2.5.6 の Q 学習の内容をベースにして以下に述べる．

Q 学習は式(14)に示すような行動価値関数を最大化するように学習を行う手法である．

$$Q(s_t, a_t) = E_{s_{t+1}}\left[r(s_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})\right] \tag{14}$$

ここで，$E_{s_{t+1}}$は次に取りうる全ての状態に対する期待値，$\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$は取りうる行動の中から最大の Q 値を返す関数を意味する．DPG では，行動の方策を$\mu(s)$と置いたときに式(15)のように近似することができる．

$$\operatorname*{Max}_{a_{t+1}} Q(s, a) \approx Q\big(s, \mu(s)\big) \tag{15}$$

これは，Q 値が最大となる行動を行動の方策$\mu(s)$が選択(学習)していると言える．

2.6　深層強化学習

深層強化学習は強化学習における価値関数や方策関数の関数近似にニューラルネットワークを使用したものをいう．従来の強化学習では価値関数や方策関数の関数近似に線形関数が使用されていた．また，特徴量の設計には人の手が必要であった．しかし，深層強化学習では，特徴量の設計をニューラルネットワークが行うため，人手が必要ない．さらに直接環境を入力として与えるだけで従来の強化学習よりも大幅に上回る性能が確認された［15］［16］.

2.6.1　Deep Deterministic Policy Gradient（DDPG）

Deep deterministic policy gradient(DDPG)は，Lillicrap ら［17］によって提案された，連続行動空間の扱いに対応した深層強化学習手法の一つである．その構造は，2.5.7 で述べた Deterministic policy gradient(DPG)がベースとなっており，図 7 に示すように，アクタの行動出力とクリティックの Q 値推定にニューラルネットワークを用いることで，膨大な状態数から考えられる全ての行動の Q 値に対して最大値を取るような方策を近似することを可能にしている．

方策は DPG と同様に行動を直接スカラー値で出力するため決定論的である．ある状態の Q 値を計算するために，アクタの出力を Q ネットワークへ送り，Q 値を計算する．これは 2.5.4 で述べた TD 誤差の計算時のみ行われる．

学習を安定化させるために，クリティックとアクタの両方にターゲットネットワークを作成する．これらのネットワークはメインネットワークに基づくソフトな更新(ソフトアップデート)を行う．具体的な更新方法については後述する．

次にアクタとクリティック両モデルで計算される損失関数について示す．クリティックとアクタの損失関数は式(16)(17)のように定義できる．

$$J_Q = \frac{1}{N}\sum_{t=1}^{N}\left(r_t + \gamma(1-d)Q_{target}\left(s'_t, \mu_{target}(s'_t)\right) - Q(s_t, \mu(s_t))\right)^2 \tag{16}$$

$$J_\mu = \frac{1}{N}\sum_{t=1}^{N} Q\left(s_t, \mu(s_t)\right) \tag{17}$$

まず，式(16)で示したアクターネットワークの損失について示す．この損失は，単純に状態の Q 値の和である．Q 値の計算にはクリティックネットワークを使い，アクターネットワークで計算された行動を渡す．ここで，我々は最大の Q 値を得たいため，この結果を最大化する必要がある．

式(17)で示したクリティックの損失は TD 誤差で表される．次の状態の Q 値を計算するために後述するターゲットネットワークを使用する．TD 誤差は 2.5.4 で述べたように，現在予測している価値と実際に得られた価値の差異であるため，ここで得られる損失は最小化する必要がある．

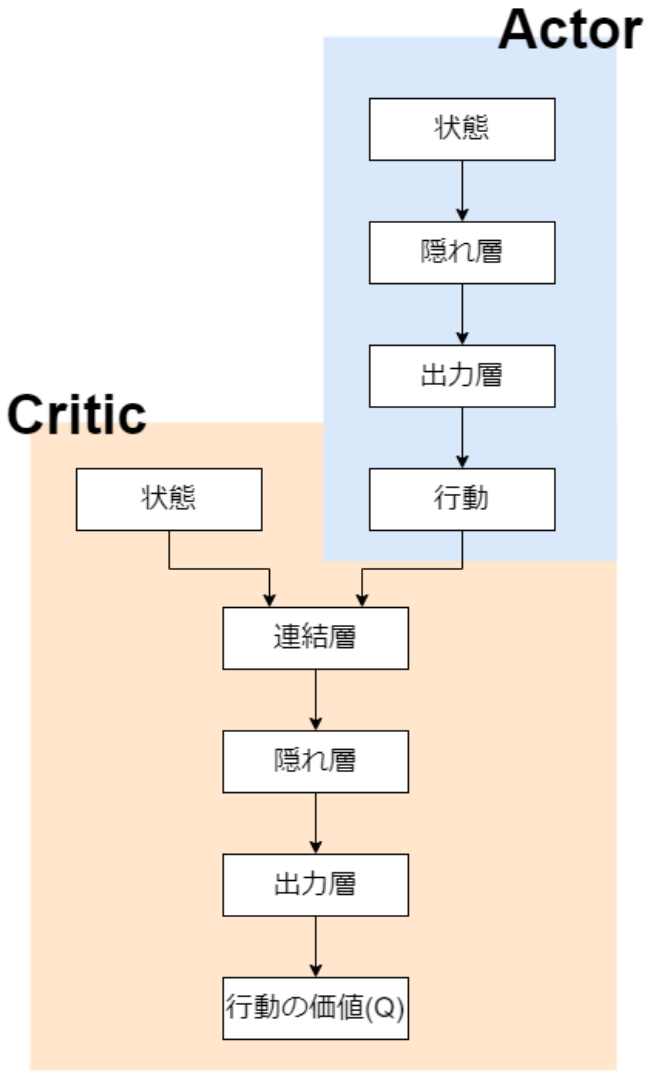


図 7．DDPG Model Architecture

また，DDPG アルゴリズムでは，学習の安定性を向上させるために以下に挙げるようないくつかの工夫が施されている．

（1）Experience Replay

強化学習に用いられるエピソードデータは，エージェントが蓄積した時系列データである．時系列データを数ステップごとに分割して Q 関数の更新を行う場合，デ

ータ間の相関が大きくなり，学習が進まない場合がある．より汎用的な Q 関数を学習するためには，様々なエピソードデータから学習する必要がある．

Experience Replay は，蓄積された過去のエピソードデータからランダムに選択されたミニバッチデータにより Q 関数の更新を行う．

(2) Target Network

Target Network は行動選択および Experience Replay の TD 誤差を計算する際に用いられるネットワークである．DQN や DDPG などの深層強化学習アルゴリズムでは，Q 値の更新に遷移先の状態の最大価値を用いる．強化学習では，行動価値が伝播していくため，他の状態の評価にも影響を与え，結果的に学習が不安的になる可能性がある．

そこで，図 8 に示すような 2 つのネットワークを使用することで，Q 学習における状態の過大評価問題を緩和することができる．

Experience Replay によって学習した Q Network のパラメータはその後 Target Network に反映されるが，この反映方法には以下の 2 種類の方法がある．

(1) Hard Update

Q Network のパラメータを定期的に Target Network にコピーする．

(2) Soft Update

式(18)のように，Q Network を更新するたびに，少しずつパラメータを反映させていく．

$$\theta \leftarrow (1-\alpha)\theta + a\theta \tag{18}$$

α はハイパーパラメータであり，一度にどれほどパラメータを Q Network に反映させるかを指定する．

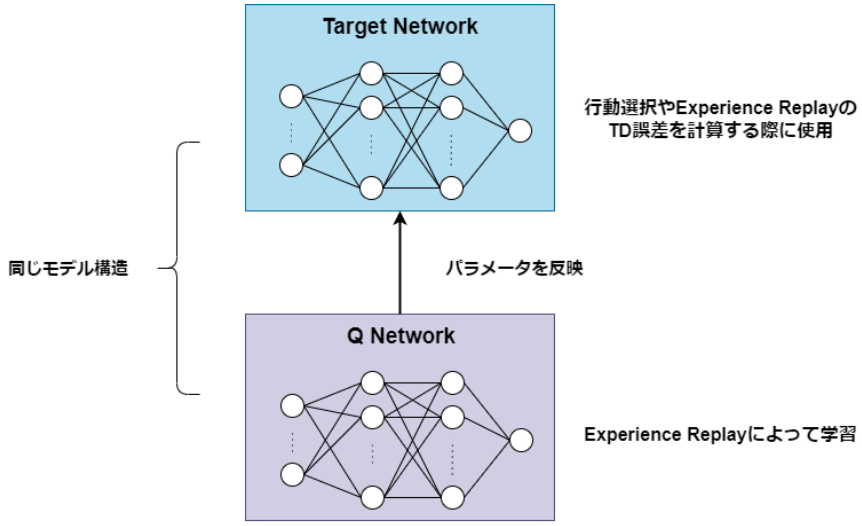


図 8. Target Network と Q Network の相互関係

(3) ε - greedy（貪欲法）

強化学習における行動選択の際にランダム性を導入した手法．ε は探索率と呼ばれるパラメータで，エージェントがランダムに行動選択する確率を指定できる．ε 値は 0~1 の範囲をとり，値が大きいほど行動のランダム性が大きくなる．一般的に，この ε 値は 0.1 などの非常に小さい値が設定されることが多いが，適応させる環境によっては最適な行動を見つけることができない場合もある．

第3章　システムモデルと提案手法

3.1　システムモデル

本稿では，図 9 に示すように，複数のユーザデバイス e={1,2,3,...,n}から生成されるタスクやデータを，複数のエッジサーバ E={1,2,3,...,N}で処理する一般的なエッジコンピューティングを想定している．各ユーザデバイスは，デスクトップパソコンやノート PC などの一般的なコンピューティングデバイスから，モバイルヘルスモニタリングなどに使用されるウェブカメラやウェアラブルデバイスなど多岐にわたる．これらはそれぞれ異なるサイズの要求タスクを発生させる．

また，ユーザデバイス自身が持つ計算能力も様々である．仮にユーザデバイスがタスクをエッジサーバへオフロードすることを選択した場合，各ユーザデバイスは自身の利益のみを考え，最も計算能力の高いエッジサーバへタスクをオフロードしようと考える．しかし，高性能なエッジサーバでも，限られた計算能力ではこれらのタスクすべてに対してリアルタイムに処理を行うことは難しい．　処理に必要なリソースの割り当てが間に合わず，大きな遅延を引き起こす可能性もある．

このことから，エッジコンピューティング環境における計算オフロードは，タスクの分配をバランスよく行い，各エッジサーバの稼働率をある程度の値まで向上させることが重要であると考えられる．

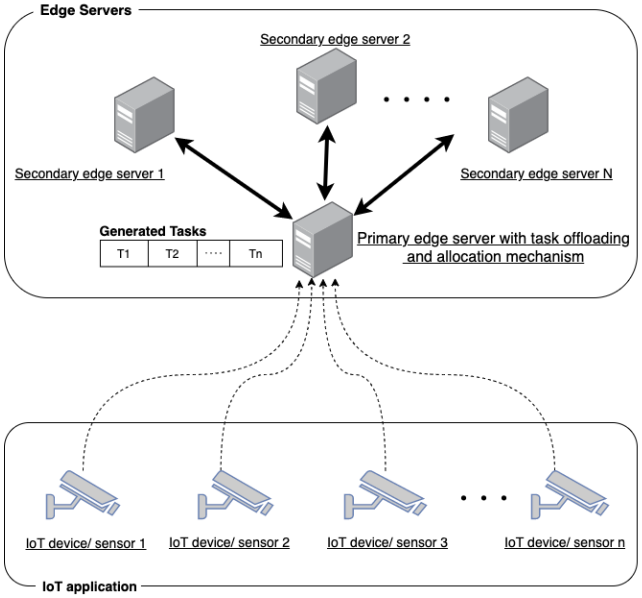


図 9. エッジコンピューティングと計算オフロードのシステムアーキテクチャ（提案）

3.2 提案手法

強化学習アルゴリズムは，エッジコンピューティング環境におけるネットワーク経路問題に対する有効な手法であることが既にいくつかの文献で提案されている．特に，深層強化学習アルゴリズムの一つである Deep Q-Network（以下 DQN）は，複雑なリソース割り当て問題と高次元な状態空間に対応できるため，非常に有用な手法と言える．しかし，DQN を含む一般的な強化学習アルゴリズムの短所として，常に離散的な行動決定しかできず，連続的な行動空間に対応できない点がある．

本稿では，Primary edge server をエージェントとしたシングルエージェントシナリオを想定した連続的な行動決定を行うタスクオフロードメカニズムについて提案する．エージェントの複雑な意思決定を実現させるために，Actor-Critic モデルに基づく深層強化学習アルゴリズムである Deep Deterministic Policy Gradient (DDPG)を採用する．

DDPG は 2.5.7 および 2.6.2 で述べたように，連続的な行動空間に対応できるという特徴を持っている．DDPG を使用することで，動的な負荷分散ポリシーを学習することが可能となる．ここでは，Primary Edge Server が各ユーザデバイスから収集したタスクが，環境の観測に基づいて各エッジサーバのスロットに格納される．

3.2.1 状態

本手法では行動を決定するために考慮する状態について以下の 4 つの要素を定義した．それぞれの状態について以下に示す．

- 各エッジサーバで利用可能な計算資源（連続値）

  各エッジサーバには最大 60GB の計算資源が備わっており，行動として決定された計算資源の割り当て率に従って，対象のタスクに資源を分配する．

- 各接続で利用可能な帯域幅（連続値）

  各エッジサーバ間には最大 1Gbps の帯域が備わっており，行動として決定されたマイグレーション帯域幅に従って，帯域を提供する．

- ユーザデバイスのオフロード状況（離散値）

  ユーザデバイスがタスクをエッジサーバへオフロードしているか否かを判断する．オフロードの状況は 0 もしくは 1 の 2 値で表され，0 の場合はユーザデバイス内にタスクが存在しており，1 の場合はエッジサーバへオフロードされていることを意味する．

- ユーザデバイスの位置情報（離散値）

  各ユーザデバイスの位置情報を x 座標，y 座標の 2 次元で表現し取得している．

3.2.2 行動

本手法ではエージェントが取る行動について以下の 3 つの要素を定義した．それぞれの状態を以下に示す．

- 計算資源の割り当て率（連続値）

  対象のエッジサーバへオフロードする際にエッジサーバから割り当てられる計算資源について決定する．これは現在の方策に基づいて連続値で決定される．

- マイグレーション帯域幅（連続値）

  エッジサーバへオフロードする際のノード間のマイグレーション帯域幅について決定する．これは現在の方策に基づいて連続値で決定される．

- オフロード先エッジサーバの決定（離散値）

  どのエッジサーバへタスクをオフロードするか決定する．各エッジサーバにはそれぞれ ID が定められており，現在の方策に基づいて ID（離散値）を出力し，オフロード先を決定する．

3.2.3 報酬

即時報酬rは，1 ステップで処理されたタスクの総数と定義する．ステップ毎に得られるrの値が高いほど処理されるタスク数が多いことを示しているが，本手法では即時報酬rの最大化が目標ではなく，長期的に獲得できる累積報酬Rの最大化を目的としているため，学習の際にモデルの方策に影響を与えているのは即時報酬rではなく，2.5.2 で述べた価値(将来得られる報酬の期待値)であることに注意したい．

3.2.4 学習モデル

本手法では，2.5.5 で述べた Actor-Critic と呼ばれるアルゴリズムを用いて，価値関数と方策関数の両方を学習している．

2.6.2 で述べたように，DDPG ではアクターネットワーク，クリティックネットワークの両方で損失を計算している．アクターネットワークの損失は Q 値を示しており，アクタではこの値を最大化することが目的である．また，クリティックネットワークの損失は，2.5.4 で述べた TD 誤差を示している．TD 誤差は現在予測している価値を実際の価値の差異であるため，クリティック内で計算して得られる損失については最小化することが目的となる．

3.2.5 学習

本手法による学習の流れを図 10 の擬似コードに示す．1 行目の while による繰り返し処理では，あらかじめ指定している最大学習エピソード回数まで試行を繰り返す処理を記述している．3 行目の for による繰り返し処理では，1 エピソード内で更新されるステップ数の最大値と各ステップにおける処理について記述している．4，5 行目では，現在の方策に基づく行動選択とそれによって更新された状態と報酬を獲得する処理を意味しており，6 行目以降の if 文による条件分岐処理は 2.6.2 で述べた Experiment Replay による経験再生学習を意味している．また，学習には 2.5.6 で述べた$\varepsilon - greedy$法を採用している．学習の継続判定を行う処理は 11 行目の if 文で表現しており，エピソード毎に獲得した平均報酬があらかじめ指定したエピソード数以上増加していれば学習を継続し，そうでなければカウントを加算し，ある一定までカウントがたまれば学習を終了する．

Algorithm 1 DDPG Offloading Mecahnism

Input: $State, Action$
Output: $Reward$
$Initialisation : Environment, LearningModel$
LOOP Process
while $counter < MAXEPISODE$ **do**
  initialize reward $r \leftarrow 0$
  for $i = 0$ to $MAXSTEP$ **do**
    choose action a
    get parameter s', r
    if $(memory > capacity)$ **then**
      Randomly extracted from memory and trained
      Update State $s \leftarrow s'$
      Collecting reward $R \leftarrow r$
    end if
    if $(i == LastEpisode)$ **then**
      Calculate the average reward
      if $(reward > maxreward)$ **then**
        Change variation
      else $\{reward < maxreward\}$
        $counter + 1$
      end if
    end if
  end for
  $episode + 1$
end while

図 10. 実行プログラムの擬似コード

第4章　実験 1 と評価

4.1　目的

実験 1 では，提案手法が分散配置された各エッジサーバに対して最適なオフロード決定を行えるかを検証する [18].

4.2　実験方法

実験では，タスクサイズや位置情報がランダムに生成された数値データをエージェントに入力し，DDPG アルゴリズムの現在の方策に従って最適と思われるエッジサーバへオフロード決定を行う.

本実験は独自に構築した仮想的なローカルエッジコンピューティング環境でシミュレーション形式にて行うこととする.

システム管理者は，環境にデプロイするエッジサーバとエッジデバイスの数および生成されるタスクサイズやマイグレーションに必要なマイグレーション帯域幅などを事前にパラメータとして設定することができる.

4.3　実験環境

前述のとおり，本実験は仮想的なエッジコンピューティング環境を強化学習における環境と定義し，シミュレーションを行う．実験に使用したマシン性能を表 2 に示す.

表 2. 実験に使用したマシン性能

OS (Operating System)	Windows10 Home
CPU (Central Processing Unit)	Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz
GPU (Graphics Processing Unit)	NVIDIA GeForce RTX 2070 SUPER
RAM (Random Access Memory)	16 GB
ストレージ	500GB SSD

表 3. シミュレーションパラメータ

Parameter	Value
学習ステップ	3000 ステップ
データサイズ	50 MB
マイグレーション帯域幅	1 GB/s
ユーザデバイス数	30 台
エッジサーバ数	10 台
エッジサーバの最大同時処理数	5 台

表 4. モデルパラメータ

Parameter	Value
レイヤ数	5
活性化関数	ReLu 関数
Actor 学習率	1.0×10^{-4}
Critic 学習率	2.0×10^{-4}
割引率（γ）	0.9
Soft Update 更新率	1.0×10^{-2}
Experience Replay バッファサイズ	1.0×10^{4}
バッチサイズ	32

4.4 結果と考察

実験 1 では，提案手法に基づいて構築した強化学習モデルが仮想的なエッジコンピューティング内で正常に動作するか検証した．ここでは，街中の監視カメラや工場のライン管理センサなどの移動性を持たないデバイスから発生されるタスクを想定している．

図 11. はエピソード毎に環境から得られる報酬を示しているが，報酬の値はエピソードが進むにつれ増加していることが見て取れる．ランダムに行動決定を行う初期方策時のエピソードから最終エピソードまでの報酬範囲は 307 であった．これはエピソード内でのエッジサーバのタスク処理数の平均が 300 台以上増加していることを示している．

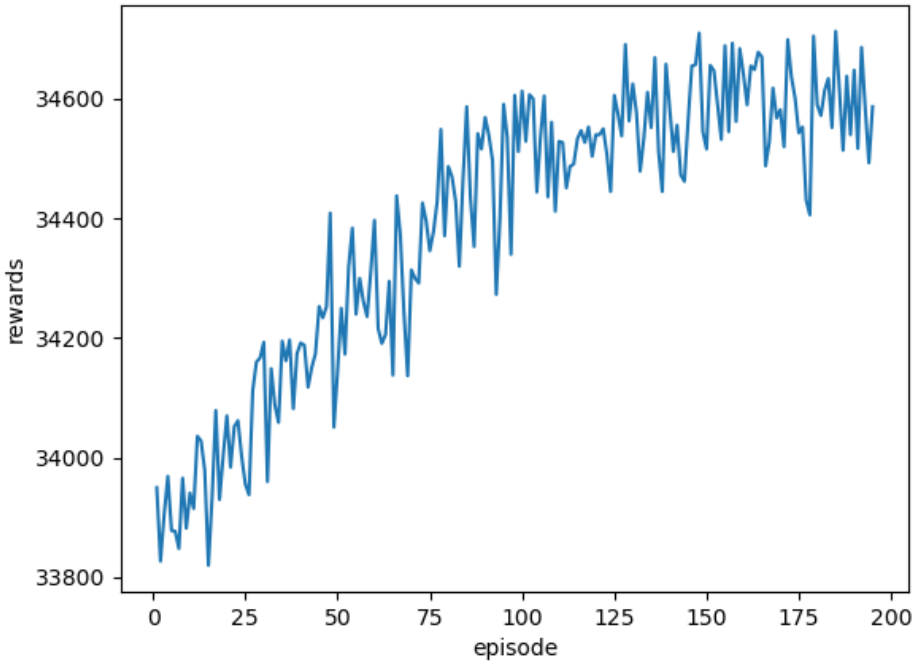


図 11. エピソード毎の報酬の変化

第5章　実験２と評価

5.1　目的

実験２では，実際のモビリティデータセットに対して提案手法を実装することで，提案手法の実社会における有用性を検証する［19].

5.2　実験方法

実験 1 と同様に，仮想環境でのシミュレーション形式で実験を行う．実験に使用するパラメータとその値は実験１と同じとする．

5.3　実験環境

実験に使用したマシンやシミュレーション環境は実験 1 と同様に，実験 2 では生成されるタスクとして，無線ネットワークデータのためのコミュニティリソース CRAWDAD(Community Resource for Archiving Wireless Data At Dartmouth)に公開されているモビリティデータセットを使用した(以下図 12.).

本データセットは，駅構内で使用されているモバイルデバイス全 90 台のモビリティデータを収集したものである．本データセットを使用することで，提案手法がモバイルデバイスの処理に対しても有効か評価することができる．

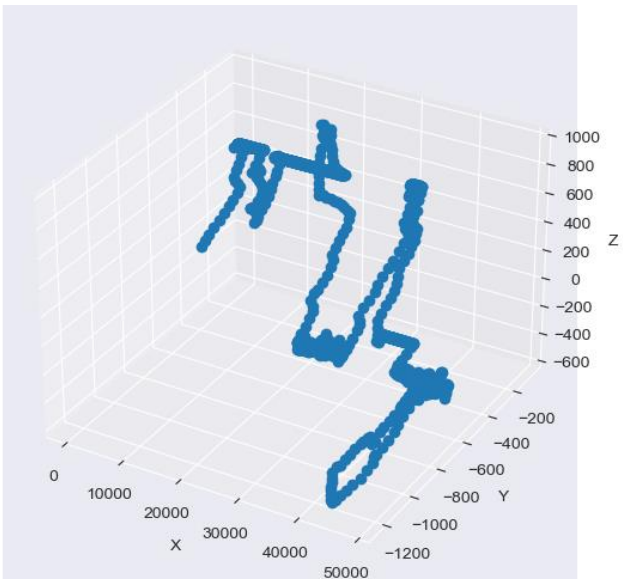


図 12. 使用するモビリティデータセット

5.4　結果と考察

実験 2 では，実際のモビリティデータセットを使用して実験を行うことで，オフロードメカニズムが移動性を持つデバイスに対しても有効であるか評価した．

実験結果として図 13. に示すように，学習開始時のランダムなオフロードポリシーに従っている場合と比較して，学習後半では，エピソードが進むごとに環境から得られる報酬を多く獲得していることが確認できる．これは，オフロードメカニズムがモビリティデータセットをタスクと想定した環境においても適切なオフロードポリシーを学習していることが考察できる．しかし，ユーザデバイス数が増加していくにつれエピソード毎に得られる報酬の範囲にブレが生じていることが分かる．これは，環境内に設置しているエッジサーバの数に対してユーザデバイス数が多く，メカニズムによるオフロード決定を行っても処理できないタスクがあり，結果として報酬が低下していると考えられる．

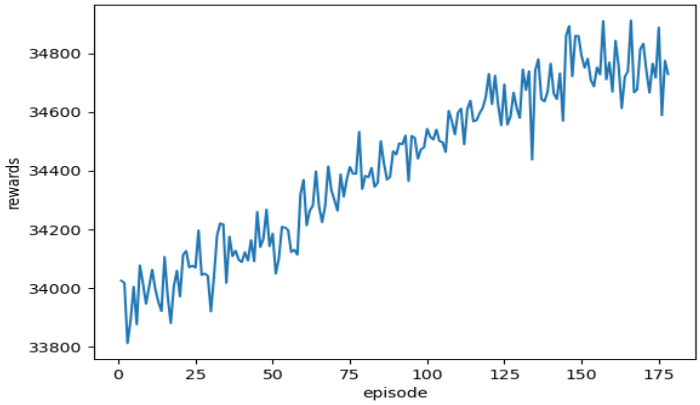
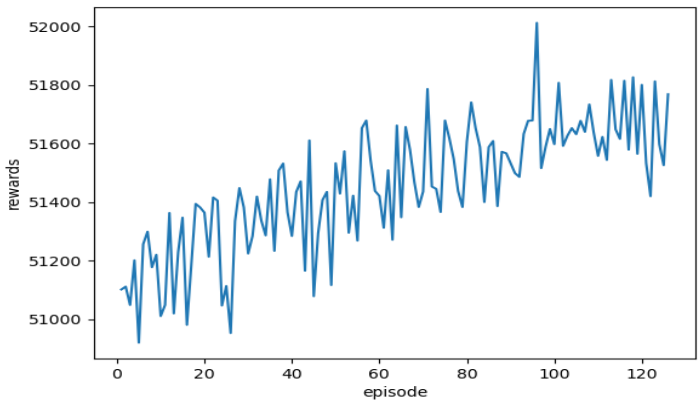
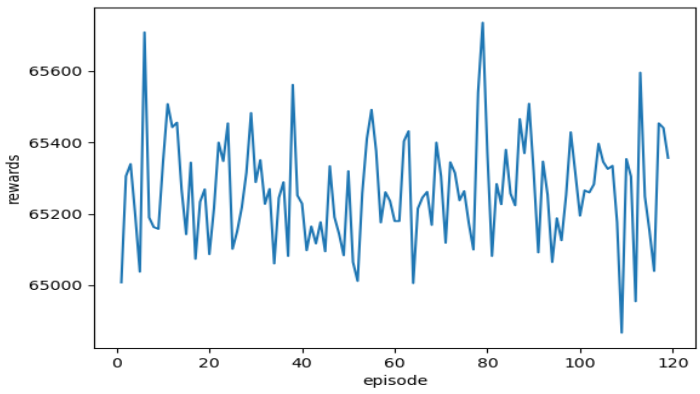
ユーザデバイス数	実行結果
30 台	
50 台	
80 台	

図 13. 実験に使用したユーザデバイス数と実行結果

第6章　結論と今後の課題

6.1　まとめ

現在, IoT アプリケーションやリアルタイムアプリケーションを高速に処理する方法としてエッジコンピューティングが注目されている．しかし，現状のエッジコンピューティングシステムの多くはエッジサーバへの負荷集中時の対策として地理的に距離の離れたクラウドデータセンタへ処理をオフロードする手法がとられている．

本研究では，エッジコンピューティング環境でのリアルタイムな処理を持続的に提供できるように分散配置されたエッジサーバに対して処理をオフロードするメカニズムを開発することを目的とした．

そこで，意思決定問題やネットワーク経路問題と相性の良い強化学習と高次元な状態空間に対応できる深層学習を組み合わせた深層強化学習を使用したオフロードメカニズムを提案した．

仮想的なエッジコンピューティング環境を独自に構築し，モビリティデータに対して提案手法がオフロードを行う様子を観測する実験を行った結果，提案手法のモデルの学習が進むにつれてシステム全体の稼働率が向上したことを確認し，提案手法の有効性について検証することができた．

6.2　今後の課題

今回実験に使用した仮想的なエッジコンピューティング環境は実験 1 では静止したユーザデバイスのみ，実験 2 ではモバイルユーザデバイスのみの環境となっている．実際の環境は静止したデバイスや常に移動しているデバイスが入り乱れた環境が大半であると予測されるため，提案手法がこれらの 2 種類のデバイスが同時に存在する環境でも有効か検証する必要がある．また，提案手法の評価方法が，現在は報酬として定義しているシステム全体の稼働率のみとなっている．より実用的なシステムにするために，今後は各エッジサーバの稼働率やタスク処理状況，処理時間などを可視化する必要がある．

参考文献

[1] K.Ashton, Internet of Things, 第 巻 Vol.6, Advances in Internet of Things, 2009, pp. 97-101.

[2] S. I. B. Z. S. H.-M.-Z. M. R.-Q. M. S. S. A. a. A. R. Jalal Sakhdari, “Edge Computing: A Systematic Mapping Study,” 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Iran, 2021.

[3] W. H. Z. H. S. Y. Z. Gao, “Q-Learning-Based Task Offloading and Resource Optimization for a Collaborative Computing System,” IEEE Access, August, 2020.

[4] L. C. S. R. J. Xu, “Online learning for offloading and autoscaling in energy harvesting mobile edge comuting,” IEEE Transaction on Cognitive Communications and Networking, Coral Gables, FL, USA, 2017.

[5] A. Tyagi, “A Review Paper on Cloud Computing,” International Journal of Engineering Research & Technology (IJERT), April, 2018.

[6] “Wikipedia, The Free Encyclopedia,” [オンライン]. Available: https://en.wikipedia.org/wiki/Computation_offloading. [アクセス日: 20 11 2021].

[7] B. B. G. A. F. P.-V. H. Madsen, “Reliability in the utility computing era: Towards reliable Fog computing,” 2013 20th International Conference on Systems, Signals and Image Processing (IWSSIP), Bucharest, Romania, 2013.

[8] S. I. P. M. M. N. Byung-Gon Chun, “CloneCloud: Elastic execution between mobile device and cloud,” Proceedings of the sixth European Conference on Computer Systems (EuroSys2011), alzburg, Austria, 2011.

[9] D. L. U. R. B. O. B. K. Kirak Hong, “Mobile fog: a programming model for large-scale applications on the internet of things,” Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing, 16 August, 2013.

[10] 藤. 貴行, “スケールアップとスケールアウトの違いとは？メリット・デメリットを解説,” KAGOYA JAPAN Inc., 20 10 2021. [オンライン]. Available: https://www.kagoya.jp/howto/engineer/itsystem/scale-up_scale-out/. [アクセス日: 25 11 2021].

[11] 木村　元, <第 1 回>強化学習の基礎, 第 巻 Vol.52, 公益社団法人 計測自動制御学会, 2021, pp. 72-77.

[12] 小. 前. C. Szepesvari(著), “Algorithms for Reinforcement Learning 速習　強化学習基礎理論とアルゴリズム,” 共立出版, 東京, 2017.

[13] D. e. a. Silver, “Deterministic policy gradient algorithms.,” ICML, 2014.

[14] R. Frank, “The perceptron,” A perceiving and recognizing automaton Project Para, 1957.

[15] e. a. Mnih Volodvmvr, “Playing atari with deep reinforcement learning,” arXiv preprint, 2013.

[16] e. a. Mnih Volodvmvr, “Human-level control through deep reinforcement learning,” 2015.

[17] e. a. Lillicrap Timothy P, “Continuous control with deep reinforcement learning,” arXiv preprint, 2015.

[18] B. O. A. Shintaro Ide, “Development of an RL-based Mechanism to Augment Computation Offloading in Edge Computing,” International Conference on Computational Science and its Applications (ICCSA), Fukuoka, 2021.

[19] 井手　慎太郎, アプドゥハン・ベーナディ, “エッジコンピューティングにおける RL ベースの効率的なタスクオフロードと割り当てに向けたメカニズムの開発,” SPSY HotSPA2021, 電子情報通信学会, 福岡県, 2021.

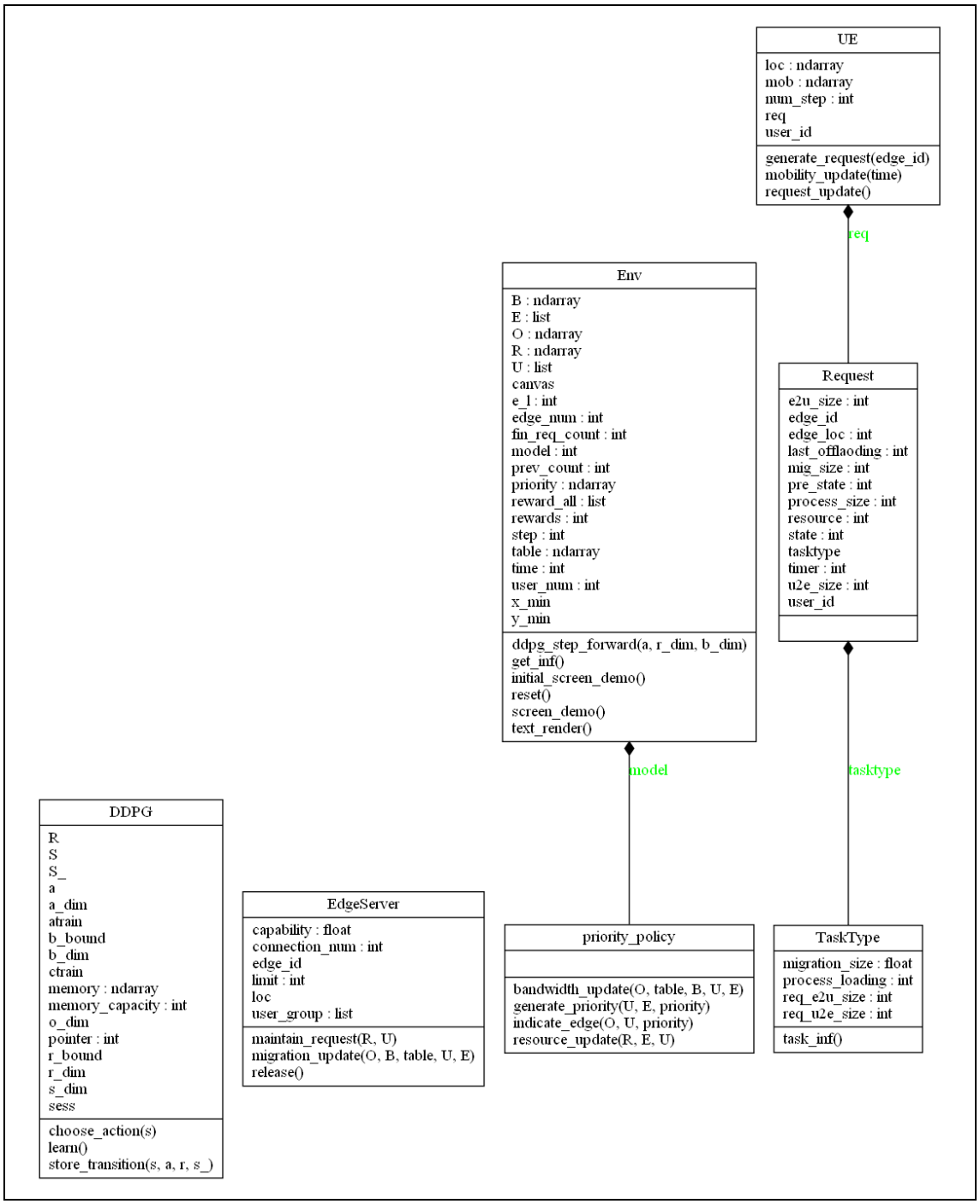
[20] A. Tyagi, “A Review Paper on Cloud Computing,” VIMPACT-2017 Conference Proceedings, jaipur, India, 2017.

[21] A. Geron, scikit-learn, Keras, Tensorflow による実践機械学習, 東京: 株式会社オライリー・ジャパン, 2020.

謝辞

本論文の執筆にあたり，多くの方々にご支援いただきました．
主指導教員であるアプドゥハン・ベーナディ教授には，研究の着想から，調査，論文執筆まで多くのご指導をいただきました．心から感謝申し上げます．
研究発表会では，アドバイザーである仲　隆教授，米元　聡教授をはじめ，多くの先生方から本研究および発表に関する貴重なご助言を賜りました．感謝申し上げます．
さらに，仲　隆教授，米元　聡教授には本論文の副査として，貴重なご助言を賜りました．重ねて感謝申し上げます．
また，所属するアプドゥハンゼミの 4 年生の皆さんには日頃から多くのご支援をいただきました．お礼申し上げます．
最後に，本研究ならびに学業全般にわたって経済的・心身的に支援して下さった家族に深く感謝し，お礼申し上げます．

付録



オフロードメカニズムのクラス図

ソースコード 1. 強化学習における「環境」を定義する env.py

```
1
2   import random
3   import numpy as np
4   import math
5   import matplotlib.pyplot as plt
6   import os
7   # from render import Demo
8
9   ####################  ハイパーパラメータ  ####################
10  LOCATION = "KSU"
11  USER_NUM = 50
12  EDGE_NUM = 10
13  LIMIT = 5
14  MAX_EP_STEPS = 3000
15  TXT_NUM = 92
16  r_bound = 1e9 * 0.063
17  b_bound = 1e9
18
19  ####################  function  ####################
20  def trans_rate(user_loc, edge_loc):
21      B = 2e6
22      P = 0.25
23      d = np.sqrt(np.sum(np.square(user_loc[0] - edge_loc))) + 0.01
24      h = 4.11 * math.pow(3e8 / (4 * math.pi * 915e6 * d), 2)
25      N = 1e-10
26      return B * math.log2(1 + P * h / N)
27
28  def BandwidthTable(edge_num):
29      BandwidthTable = np.zeros((edge_num, edge_num))
30      for i in range(0, edge_num):
31          for j in range(i+1, edge_num):
32              BandwidthTable[i][j] = 1e9
33      return BandwidthTable
```

```
34
35  def two_to_one(two_table):
36      one_table = two_table.flatten()
37      return one_table
38
39  def generate_state(two_table, U, E, x_min, y_min):
40      # 初期化
41      one_table = two_to_one(two_table)
42      S = np.zeros((len(E) + one_table.size + len(U) + len(U)*2))
43      # transform
44      count = 0
45      # 各エッジサーバの利用可能な計算資源を取得
46      for edge in E:
47          S[count] = edge.capability/(r_bound*10)
48          count += 1
49      # 各接続の利用可能な帯域幅を取得
50      for i in range(len(one_table)):
51          S[count] = one_table[i]/(b_bound*10)
52          count += 1
53      # 各ユーザデバイスのオフロード状況を取得（0 or 1）
54      for user in U:
55          S[count] = user.req.edge_id/100
56          count += 1
57      # ユーザデバイスの位置情報を取得
58      for user in U:
59          S[count] = (user.loc[0][0] + abs(x_min))/1e5
60          S[count+1] = (user.loc[0][1] + abs(y_min))/1e5
61          count += 2
62      return S
63
64  def generate_action(R, B, O):
65      # 計算資源割り当て（連続値）
66      a = np.zeros(USER_NUM + USER_NUM + EDGE_NUM * USER_NUM)
67      a[:USER_NUM] = R / r_bound
68      # 帯域幅割り当て（連続値）
69      a[USER_NUM:USER_NUM + USER_NUM] = B / b_bound
```

```
70          # オフロード決定（離散値）
71          base = USER_NUM + USER_NUM
72          for user_id in range(USER_NUM):
73              a[base + int(O[user_id])] = 1
74              base += EDGE_NUM
75          return a
76
77      # モビリティデータ取得
78      def get_minimum():
79          cal = np.zeros((1, 2))
80          for data_num in range(TXT_NUM):
81              data_name = str("%03d" % (data_num + 1))    # plus zero
82              file_name = LOCATION + "_30sec_" + data_name + ".txt"
83              file_path = "data/" + LOCATION + "/" + file_name
84              f = open(file_path, "r")
85              f1 = f.readlines()
86              # get line_num
87              line_num = 0
88              for line in f1:
89                  line_num += 1
90              # .txt からデータを収集
91              data = np.zeros((line_num, 2))
92              index = 0
93              for line in f1:
94                  data[index][0] = line.split()[1]    # x
95                  data[index][1] = line.split()[2]    # y
96                  index += 1
97              # データを cal に格納
98              cal = np.vstack((cal, data))
99          return min(cal[:, 0]), min(cal[:, 1])
100
101     # エッジサーバの位置情報取得
102     def proper_edge_loc(edge_num):
103
104         # e_l の初期化
105         e_l = np.zeros((edge_num, 2))
```

```
106         # データの平均を計算
107         group_num = math.floor(TXT_NUM / edge_num)
108         edge_id = 0
109         for base in range(0, group_num*edge_num, group_num):
110             for data_num in range(base, base + group_num):
111                 data_name = str("%03d" % (data_num + 1))    # plus zero
112                 file_name = LOCATION + "_30sec_" + data_name + ".txt"
113                 file_path = "data/" + LOCATION + "/" + file_name
114                 f = open(file_path, "r")
115                 f1 = f.readlines()
116                 # line_num を取得しデータを初期化
117                 line_num = 0
118                 for line in f1:
119                     line_num += 1
120                 data = np.zeros((line_num, 2))
121                 # .txt からデータを収集
122                 index = 0
123                 for line in f1:
124                     data[index][0] = line.split()[1]    # x
125                     data[index][1] = line.split()[2]    # y
126                     index += 1
127                 # 収集したデータをスタックとして積み重ねて保存
128                 if data_num % group_num == 0:
129                     cal = data
130                 else:
131                     cal = np.vstack((cal, data))
132             e_l[edge_id] = np.mean(cal, axis=0)
133             edge_id += 1
134         return e_l
135
136 ##########################    UE    ##########################
137 class UE():
138         def __init__(self, user_id, data_num):
139             self.user_id = user_id    # number of the user
140             self.loc = np.zeros((1, 2))
141             self.num_step = 0    # the number of step
```

```
142
143             # num_step を計算し self.mob を定義
144             data_num = str("%03d" % (data_num + 1))  # plus zero
145             file_name = LOCATION + "_30sec_" + data_num + ".txt"
146             file_path = "data/" + LOCATION + "/" + file_name
147             f = open(file_path, "r")
148             f1 = f.readlines()
149             data = 0
150             for line in f1:
151                 data += 1
152             self.num_step = data * 30
153             self.mob = np.zeros((self.num_step, 2))
154
155             # self.mob にデータを書き込む
156             now_sec = 0
157             for line in f1:
158                 for sec in range(30):
159                     self.mob[now_sec + sec][0] = line.split()[1]  # x
160                     self.mob[now_sec + sec][1] = line.split()[2]  # y
161                 now_sec += 30
162             self.loc[0] = self.mob[0]
163
164         def generate_request(self, edge_id):
165             self.req = Request(self.user_id, edge_id)
166
167         def request_update(self):
168             # default request.state == [5：disconnection（切断状態）,6：migration（移動状態）]
169             if self.req.state == 5:
170                 self.req.timer += 1
171             else:
172                 self.req.timer = 0
173                 if self.req.state == 0:
174                     self.req.state = 1
175                     self.req.u2e_size = self.req.tasktype.req_u2e_size
176                     self.req.u2e_size -= trans_rate(self.loc, self.req.edge_loc)
177                 elif self.req.state == 1:
```

```
178                         if self.req.u2e_size > 0:
179                             self.req.u2e_size -= trans_rate(self.loc, self.req.edge_loc)
180                         else:
181                             self.req.state = 2
182                             self.req.process_size = self.req.tasktype.process_loading
183                             self.req.process_size -= self.req.resource
184                     elif self.req.state == 2:
185                         if self.req.process_size > 0:
186                             self.req.process_size -= self.req.resource
187                         else:
188                             self.req.state = 3
189                             self.req.e2u_size = self.req.tasktype.req_e2u_size
190                             self.req.e2u_size -= 10000    # value is small,so simplify
191                     else:
192                         if self.req.e2u_size > 0:
193                             self.req.e2u_size  -=  10000     #  B*math.log(1+SINR(self.user.loc,
194 self.offloading_serv.loc), 2)/(8*time_scale)
195                         else:
196                             self.req.state = 4
197 
198         def mobility_update(self, time):    # t: second
199             if time < len(self.mob[:, 0]):
200                 self.loc[0] = self.mob[time]     # x
201 
202             else:
203                 self.loc[0][0] = np.inf
204                 self.loc[0][1] = np.inf
205 
206 class Request():
207         def __init__(self, user_id, edge_id):
208             # id
209             self.user_id = user_id
210             self.edge_id = edge_id
211             self.edge_loc = 0
212             # state（状態）
213             self.state = 5         # 5: not connect
```

```
214             self.pre_state=5
215             # transmission size（送信サイズ）
216             self.u2e_size = 0
217             self.process_size = 0
218             self.e2u_size = 0
219             # edge state（エッジサーバの状態）
220             self.resource = 0
221             self.mig_size = 0
222             # タスクタイプ
223             self.tasktype = TaskType()
224             self.last_offlaoding = 0
225             # timer
226             self.timer = 0
227
228 class TaskType():
229     def __init__(self):
230         ##Objection detection: VOC SSD300
231         # transmission
232         self.req_u2e_size = 300 * 300 * 3 * 1
233         self.process_loading = 300 * 300 * 3 * 4
234         self.req_e2u_size = 4 * 4 + 20 * 4
235         # migration
236         self.migration_size = 2e9
237     def task_inf(self):
238         return "req_u2e_size:" + str(self.req_u2e_size) + "¥nprocess_loading:" +
239 str(self.process_loading) + "¥nreq_e2u_size:" + str(self.req_e2u_size)
240
241 ##################  EdgeServer  ##################
242
243 class EdgeServer():
244     def __init__(self, edge_id, loc):
245         self.edge_id = edge_id  # edge server number
246         self.loc = loc
247         self.capability = 1e9 * 0.063
248         self.user_group = []
249         self.limit = LIMIT
```

```
250             self.connection_num = 0
251         def maintain_request(self, R, U):
252             for user in U:
253                 # 接続ユーザ数
254                 self.connection_num = 0
255                 for user_id in self.user_group:
256                     if U[user_id].req.state != 6:
257                         self.connection_num += 1
258                 # リクエストの維持
259                 if user.req.edge_id == self.edge_id and self.capability - R[user.user_id] > 0:
260                     # 予備接続の維持
261                     if user.req.user_id not in self.user_group and self.connection_num+1 <=
262 self.limit:
263                         # 初回：どのエッジサーバにも属さない（user_group）
264                         self.user_group.append(user.user_id)  # user_group に追加
265                         user.req.state = 0  # 接続準備
266                         # リクエストの通知
267                         user.req.edge_id = self.edge_id
268                         user.req.edge_loc = self.loc
269 
270                     # 計算資源の派遣（リソース提供）
271                     user.req.resource = R[user.user_id]
272                     self.capability -= R[user.user_id]
273 
274         def migration_update(self, O, B, table, U, E):
275 
276             # マイグレーションを維持する
277             for user_id in self.user_group:
278                 # マイグレーションを準備する
279                 if U[user_id].req.edge_id != O[user_id]:
280                     # 初期化
281                     ini_edge = int(U[user_id].req.edge_id)
282                     target_edge = int(O[user_id])
283                     if table[ini_edge][target_edge] - B[user_id] >= 0:
284                         # マイグレーション中だが，別のエッジサーバにオフロード(step 1)
285                         if U[user_id].req.state == 6 and target_edge !=
```

```
U[user_id].req.last_offlaoding:
                    # 帯域幅の削減
                    table[ini_edge][target_edge] -= B[user_id]
                    # マイグレーション開始
                    U[user_id].req.mig_size =
U[user_id].req.tasktype.migration_size
                    U[user_id].req.mig_size -= B[user_id]
                    #print("user", U[user_id].req.user_id, ":migration step 1")
                # first try to migration(step 1)
                elif U[user_id].req.state != 6:
                    table[ini_edge][target_edge] -= B[user_id]
                    # マイグレーション開始
                    U[user_id].req.mig_size =
U[user_id].req.tasktype.migration_size
                    U[user_id].req.mig_size -= B[user_id]
                    # 直前の状態を保存
                    U[user_id].req.pre_state = U[user_id].req.state
                    # マイグレーション中に旧エッジサーバとの接続を解除する
                    U[user_id].req.state = 6
                    #print("user", U[user_id].req.user_id, ":migration step 1")
                elif U[user_id].req.state == 6 and target_edge ==
U[user_id].req.last_offlaoding:
                    # マイグレーションの継続(step 2)
                    if U[user_id].req.mig_size > 0:
                        # reduce the bandwidth
                        table[ini_edge][target_edge] -= B[user_id]
                        U[user_id].req.mig_size -= B[user_id]
                        #print("user", U[user_id].req.user_id, ":migration step 2")
                    # マイグレーションの終了(step 3)
                    else:
                        # 接続ユーザ数
                        target_connection_num = 0
                        for target_user_id in E[target_edge].user_group:
                            if U[target_user_id].req.state != 6:
                                target_connection_num += 1
                        #print("user", U[user_id].req.user_id, ":migration step 3")
```

```
                                    # 別のエッジサーバに変更
                                    if E[target_edge].capability - U[user_id].req.resource >= 0
and target_connection_num + 1 <= E[target_edge].limit:
                                        # 新しいエッジサーバに登録
                                        E[target_edge].capability -= U[user_id].req.resource
                                        E[target_edge].user_group.append(user_id)
                                        self.user_group.remove(user_id)
                                        # リクエストをアップデート
                                        # id
                                        U[user_id].req.edge_id = E[target_edge].edge_id
                                        U[user_id].req.edge_loc = E[target_edge].loc
                                        # 直前の状態をリリースし送信処理を継続する
                                        U[user_id].req.state = U[user_id].req.pre_state
                                        #print("user",    U[user_id].req.user_id,    ":migration
finish")
                # 直前のオフロード決定を保存
                U[user_id].req.last_offlaoding = int(O[user_id])

            return table

        # 全ての計算資源をリリース
        def release(self):
            self.capability = 1e9 * 0.063

####################   Policy   ######################

class priority_policy():
        def generate_priority(self, U, E, priority):
            for user in U:
                # オフロード優先順位リストを取得
                dist = np.zeros(EDGE_NUM)
                for edge in E:
                    dist[edge.edge_id] = np.sqrt(np.sum(np.square(user.loc[0] - edge.loc)))
                dist_sort = np.sort(dist)
                for index in range(EDGE_NUM):
                    priority[user.user_id][index] = np.argwhere(dist == dist_sort[index])[0]
```

```
358             return priority
359
360         def indicate_edge(self, O, U, priority):
361             edge_limit = np.ones((EDGE_NUM)) * LIMIT
362             for user in U:
363                 for index in range(EDGE_NUM):
364                     if edge_limit[int(priority[user.user_id][index])] - 1 >= 0:
365                         edge_limit[int(priority[user.user_id][index])] -= 1
366                         O[user.user_id] = priority[user.user_id][index]
367                         break
368             return O
369
370         def resource_update(self, R, E ,U):
371             for edge in E:
372                 # 接続ユーザ数のカウント
373                 connect_num = 0
374                 for user_id in edge.user_group:
375                     if U[user_id].req.state != 5 and U[user_id].req.state != 6:
376                         connect_num += 1
377                 # 接続ユーザに計算資源を派遣
378                 for user_id in edge.user_group:
379                     # リクエスト状態が 5or6（切断状態）で，計算資源を提供する必要がな
380 い場合
381                     if U[user_id].req.state == 5 or U[user_id].req.state == 6:
382                         R[user_id] = 0
383                     # 接続したユーザに計算資源を提供
384                     else:
385                         R[user_id] = edge.capability/(connect_num+2)    # reserve the
386 resource to those want to migration
387             return R
388
389         def bandwidth_update(self, O, table, B, U, E):
390             for user in U:
391                 share_number = 1
392                 ini_edge = int(user.req.edge_id)
393                 target_edge = int(O[user.req.user_id])
```

```
                # マイグレーション不要
                if ini_edge == target_edge:
                    B[user.req.user_id] = 0
                # マイグレーションのための帯域を提供
                else:
                    # マイグレーション先のエッジサーバのユーザデバイスで帯域を共有
                    for user_id in E[target_edge].user_group:
                        if O[user_id] == ini_edge:
                            share_number += 1
                    # 旧エッジサーバから移行先エッジサーバまでのユーザと帯域を共有
                    for ini_user_id in E[ini_edge].user_group:
                        if ini_user_id != user.req.user_id and O[ini_user_id] == target_edge:
                            share_number += 1
                    # 帯域の割り当て
                    B[user.req.user_id] = table[min(ini_edge, target_edge)][max(ini_edge,
target_edge)] / (share_number+2)

        return B

#####################  Env  ######################

class Env():
    def __init__(self):
        self.step = 30
        self.time = 0
        self.edge_num = EDGE_NUM  # エッジサーバ数
        self.user_num = USER_NUM  # ユーザデバイス数
        # 環境オブジェクトの定義
        self.reward_all = []
        self.U = []
        self.fin_req_count = 0
        self.prev_count = 0
        self.rewards = 0
        self.R = np.zeros((self.user_num))
        self.O = np.zeros((self.user_num))
        self.B = np.zeros((self.user_num))
```

```
430             self.table = BandwidthTable(self.edge_num)
431             self.priority = np.zeros((self.user_num, self.edge_num))
432             self. E = []
433             self.x_min, self.y_min = get_minimum()
434
435             self.e_l = 0
436             self.model = 0
437
438         def get_inf(self):
439             # s_dim
440             self.reset()
441             s = generate_state(self.table, self.U, self.E, self.x_min, self.y_min)
442             s_dim = s.size
443
444             # a_dim
445             r_dim = len(self.U)
446             b_dim = len(self.U)
447             o_dim = self.edge_num * len(self.U)
448
449             # 計算資源の最大値
450             r_bound = self.E[0].capability
451
452             # 帯域幅の最大値
453             b_bound = self.table[0][1]
454             b_bound = b_bound.astype(np.float32)
455
456             # タスクサイズ
457             task = TaskType()
458             task_inf = task.task_inf()
459
460             return s_dim, r_dim, b_dim, o_dim, r_bound, b_bound, task_inf, LIMIT,
461     LOCATION
462
463         def reset(self):
464             # reset time
465             self.time = 0
```

```
466             # reward（報酬）
467             self.reward_all = []
468             # user
469             self.U = []
470             self.fin_req_count = 0
471             self.prev_count = 0
472             data_num = random.sample(list(range(TXT_NUM)), self.user_num)
473             for i in range(self.user_num):
474                 new_user = UE(i, data_num[i])
475                 self.U.append(new_user)
476             # Resource
477             self.R = np.zeros((self.user_num))
478             # Offlaoding
479             self.O = np.zeros((self.user_num))
480             # bandwidth
481             self.B = np.zeros((self.user_num))
482             # bandwidth table
483             self.table = BandwidthTable(self.edge_num)
484             # server
485             self.E = []
486             e_l = proper_edge_loc(self.edge_num)
487             for i in range(self.edge_num):
488                 new_e = EdgeServer(i, e_l[i, :])
489                 self.E.append(new_e)
490                 """
491                 print("edge", new_e.edge_id, "'s loc:¥n", new_e.loc)
492
493 print("==================================================
494 ====")
495             """
496             # model
497             self.model = priority_policy()
498
499             # リクエストの初期化
500             self.priority = self.model.generate_priority(self.U, self.E, self.priority)
501             self.O = self.model.indicate_edge(self.O, self.U, self.priority)
```

```
502             for user in self.U:
503                 user.generate_request(self.O[user.user_id])
504             return generate_state(self.table, self.U, self.E, self.x_min, self.y_min)
505
506         def ddpg_step_forward(self, a, r_dim, b_dim):
507             # release the bandwidth
508             self.table = BandwidthTable(self.edge_num)
509             # release the resource
510             for edge in self.E:
511                 edge.release()
512
513             # 1秒ごとにポリシーを更新
514             # resource update
515             self.R = a[:r_dim]
516             # bandwidth update
517             self.B = a[r_dim:r_dim + b_dim]
518             # offloading update
519             base = r_dim + b_dim
520             for user_id in range(self.user_num):
521                 prob_weights = a[base:base + self.edge_num]
522                 #print("user", user_id, ":", prob_weights)
523                 action = np.random.choice(range(len(prob_weights)),
524 p=prob_weights.ravel())  # select action w.r.t the actions prob
525                 base += self.edge_num
526                 self.O[user_id] = action
527
528
529             # request update
530             for user in self.U:
531                 # リクエストの状態を更新
532                 user.request_update()
533                 if user.req.timer >= 5:
534                     user.generate_request(self.O[user.user_id])  # offload according to the
535 priority
536                 # 既にリクエストが終了している場合
537                 if user.req.state == 4:
```

```
538                         # rewards（報酬）
539                         self.fin_req_count += 1
540                         user.req.state = 5   # request turn to "disconnect"
541                         self.E[int(user.req.edge_id)].user_group.remove(user.req.user_id)
542                         user.generate_request(self.O[user.user_id])   # offload according to the
543 priority
544
545             # edge update
546             for edge in self.E:
547                 edge.maintain_request(self.R, self.U)
548                 self.table = edge.migration_update(self.O, self.B, self.table, self.U, self.E)
549
550             # rewards
551             self.rewards = self.fin_req_count - self.prev_count
552             self.prev_count = self.fin_req_count
553
554             # 各ユーザデバイスが移動開始
555             if self.time % self.step == 0:
556                 for user in self.U:
557                     user.mobility_update(self.time)
558
559             # update time
560             self.time += 1
561
562             # return s_, r
563             return generate_state(self.table, self.U, self.E, self.x_min, self.y_min), self.rewards
564
565         def text_render(self):
566             print("R:", self.R)
567             print("B:", self.B)
568             """
569             base = USER_NUM +USER_NUM
570             for user in range(len(self.U)):
571                 print("user", user, " offload probabilty:", a[base:base + self.edge_num])
572                 base += self.edge_num
573             """
```

```
574             print("O:", self.O)
575             for user in self.U:
576                 print("user", user.user_id, "'s loc:¥n", user.loc)
577                 print("request state:", user.req.state)
578                 print("edge serve:", user.req.edge_id)
579             for edge in self.E:
580                 print("edge", edge.edge_id, "user_group:", edge.user_group)
581             print("reward:", self.rewards)
582             print("====================                                    update
583 ============================")
584
585     def initial_screen_demo(self):
586         self.canvas = Demo(self.E, self.U, self.O, MAX_EP_STEPS)
587
588     def screen_demo(self):
589         self.canvas.draw(self.E, self.U, self.O)
```

ソースコード 2. 最適なオフロード決定と資源管理を学習する DDPG.py

```
1
2   import tensorflow.compat.v1 as tf
3   tf.disable_v2_behavior()
4   import numpy as np
5   import time
6
7   ####################  ハイパーパラメータ  ####################
8   LR_A = 0.0001          # Actor の学習率
9   LR_C = 0.0002          # Critic の学習率
10  GAMMA = 0.9              # 報酬の割引率
11  TAU = 0.01             # ソフトアップデート更新率
12  BATCH_SIZE = 32        # Experiment Replay Buffer バッチサイズ
13  OUTPUT_GRAPH = True # Tensorflow 計算グラフ
14
15  ####################  DDPG  ####################
16
17  class DDPG(object):
18      def __init__(self, s_dim, r_dim, b_dim, o_dim, r_bound, b_bound):
19          self.memory_capacity = 10000
20          # dimension
21          self.s_dim = s_dim
22          self.a_dim = r_dim + b_dim + o_dim
23          self.r_dim = r_dim
24          self.b_dim = b_dim
25          self.o_dim = o_dim
26          # self.a_bound
27          self.r_bound = r_bound
28          self.b_bound = b_bound
29          # S, S_, R
30          self.S = tf.placeholder(tf.float32, [None, s_dim], 's')
31          self.S_ = tf.placeholder(tf.float32, [None, s_dim], 's_')
32          self.R = tf.placeholder(tf.float32, [None, 1], 'r')
33          # memory
34          self.memory = np.zeros((self.memory_capacity, s_dim * 2 + self.a_dim + 1),
```

```
35  dtype=np.float32)  # s_dim + a_dim + r + s_dim
36              self.pointer = 0
37              # session
38              self.sess = tf.Session()
39
40              # 入出力の定義
41              self.a = self._build_a(self.S,)
42              q = self._build_c(self.S, self.a, )
43
44              # ステップ毎に学習パラメータで目標パラメータを置き換え
45              a_params        =        tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
46  scope='Actor')
47              c_params        =        tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
48  scope='Critic')
49              ema = tf.train.ExponentialMovingAverage(decay=1 - TAU)                  # soft
50  replacement
51              def ema_getter(getter, name, *args, **kwargs):
52                  return ema.average(getter(name, *args, **kwargs))
53              # ステップ毎に重みを更新
54              target_update = [ema.apply(a_params), ema.apply(c_params)]          # soft update
55  operation
56              a_ = self._build_a(self.S_, reuse=True, custom_getter=ema_getter)    # replaced
57  target parameters
58              q_ = self._build_c(self.S_, a_, reuse=True, custom_getter=ema_getter)
59
60              # Actor learn()
61              a_loss = - tf.reduce_mean(q)  # maximize the q
62              self.atrain = tf.train.AdamOptimizer(LR_A).minimize(a_loss, var_list=a_params)
63
64              # Critic learn()
65              with tf.control_dependencies(target_update):      # soft replacement happened at
66  here
67                  q_target = self.R + GAMMA * q_
68                  td_error = tf.losses.mean_squared_error(labels=q_target, predictions=q)
69                  self.ctrain          =          tf.train.AdamOptimizer(LR_C).minimize(td_error,
70  var_list=c_params)
```

```
71
72              self.sess.run(tf.global_variables_initializer())
73
74              if OUTPUT_GRAPH:
75                  tf.summary.FileWriter("logs/", self.sess.graph)
76
77          def choose_action(self, s):
78              return self.sess.run(self.a, {self.S: s[np.newaxis, :]})[0]
79
80          def learn(self):
81              indices = np.random.choice(self.memory_capacity, size=BATCH_SIZE)
82              bt = self.memory[indices, :]
83              bs = bt[:, :self.s_dim]
84              ba = bt[:, self.s_dim: self.s_dim + self.a_dim]
85              br = bt[:, -self.s_dim - 1: -self.s_dim]
86              bs_ = bt[:, -self.s_dim:]
87
88              self.sess.run(self.atrain, {self.S: bs})
89              self.sess.run(self.ctrain, {self.S: bs, self.a: ba, self.R: br, self.S_: bs_})
90
91          def store_transition(self, s, a, r, s_):
92              transition = np.hstack((s, a, [r], s_))
93              index = self.pointer % self.memory_capacity  # replace the old memory with new
94  memory
95              self.memory[index, :] = transition
96              self.pointer += 1
97
98          # Actor Network
99          def _build_a(self, s, reuse=None, custom_getter=None):
100             trainable = True if reuse is None else False
101             with tf.variable_scope('Actor', reuse=reuse, custom_getter=custom_getter):
102                 n_l = 50
103                 net = tf.layers.dense(s, n_l, activation=tf.nn.relu, name='l1',
104 trainable=trainable)
105                 # resource (resource 範囲：0 - r_bound)
106                 layer_r0 = tf.layers.dense(net, n_l, activation=tf.nn.relu, name='r_0',
```

```
107 trainable=trainable)
108                 layer_r1 = tf.layers.dense(layer_r0, n_l, activation=tf.nn.relu, name='r_1',
109 trainable=trainable)
110                 layer_r2 = tf.layers.dense(layer_r1, n_l, activation=tf.nn.relu, name='r_2',
111 trainable=trainable)
112                 layer_r3 = tf.layers.dense(layer_r2, n_l, activation=tf.nn.relu, name='r_3',
113 trainable=trainable)
114                 layer_r4 = tf.layers.dense(layer_r3, self.r_dim, activation=tf.nn.relu,
115 name='r_4', trainable=trainable)
116
117                 # bandwidth (bandwidth：0 - b_bound)
118                 layer_b0 = tf.layers.dense(net, n_l, activation=tf.nn.relu, name='b_0',
119 trainable=trainable)
120                 layer_b1 = tf.layers.dense(layer_b0, n_l, activation=tf.nn.relu, name='b_1',
121 trainable=trainable)
122                 layer_b2 = tf.layers.dense(layer_b1, n_l, activation=tf.nn.relu, name='b_2',
123 trainable=trainable)
124                 layer_b3 = tf.layers.dense(layer_b2, n_l, activation=tf.nn.relu, name='b_3',
125 trainable=trainable)
126                 layer_b4 = tf.layers.dense(layer_b3, self.b_dim, activation=tf.nn.relu,
127 name='b_4', trainable=trainable)
128
129                 # offloading (確率： 0 - 1)
130                 # layer
131                 layer = [["layer"+str(user_id)+str(layer) for layer in range(4)] for user_id in
132 range(self.r_dim)]
133                 # name
134                 name = [["layer"+str(user_id) + str(layer) for layer in range(4)] for user_id in
135 range(self.r_dim)]
136                 # user
137                 user = ["user"+str(user_id) for user_id in range(self.r_dim)]
138                 # softmax
139                 softmax = ["softmax"+str(user_id) for user_id in range(self.r_dim)]
140                 for user_id in range(self.r_dim):
141                     layer[user_id][0] = tf.layers.dense(net, n_l, activation=tf.nn.relu,
142 name=name[user_id][0], trainable=trainable)
```

```
                    layer[user_id][1] = tf.layers.dense(layer[user_id][0], n_l,
activation=tf.nn.relu, name=name[user_id][1], trainable=trainable)
                    layer[user_id][2] = tf.layers.dense(layer[user_id][1], n_l,
activation=tf.nn.relu, name=name[user_id][2], trainable=trainable)
                    layer[user_id][3] = tf.layers.dense(layer[user_id][2],
(self.o_dim/self.r_dim), activation=tf.nn.relu, name=name[user_id][3],
trainable=trainable)
                    user[user_id] = tf.nn.softmax(layer[user_id][3], name=softmax[user_id])

                # concate
                a = tf.concat([layer_r4, layer_b4], 1)
                for user_id in range(self.r_dim):
                    a = tf.concat([a, user[user_id]], 1)
                return a

    # Critic Network
    def _build_c(self, s, a, reuse=None, custom_getter=None):
        trainable = True if reuse is None else False
        # Q value (Q value 範囲：0 - inf)
        with tf.variable_scope('Critic', reuse=reuse, custom_getter=custom_getter):
            n_l = 50
            w1_s = tf.get_variable('w1_s', [self.s_dim, n_l], trainable=trainable)
            w1_a = tf.get_variable('w1_a', [self.a_dim, n_l], trainable=trainable)
            b1 = tf.get_variable('b1', [1, n_l], trainable=trainable)
            net_1 = tf.nn.relu(tf.matmul(s, w1_s) + tf.matmul(a, w1_a) + b1)
            net_2 = tf.layers.dense(net_1, n_l, activation=tf.nn.relu, trainable=trainable)
            net_3 = tf.layers.dense(net_2, n_l, activation=tf.nn.relu, trainable=trainable)
            net_4 = tf.layers.dense(net_3, n_l, activation=tf.nn.relu, trainable=trainable)
            return tf.layers.dense(net_4, 1, activation=tf.nn.relu, trainable=trainable)   #
Q(s,a)
```

ソースコード 3. モデルの学習とログ記録を行う run.py

```
1
2   from env import Env
3   from DDPG import DDPG
4   import numpy as np
5   import matplotlib.pyplot as plt
6   import os
7   import time
8
9   ####################  ハイパーパラメータ  ####################
10  CHECK_EPISODE = 4
11  LEARNING_MAX_EPISODE = 30
12  MAX_EP_STEPS = 3000
13  TEXT_RENDER = True
14  SCREEN_RENDER = False
15  CHANGE = True
16  SLEEP_TIME = 0.01
17
18  ####################  function  ####################
19  def exploration (a, r_dim, b_dim, r_var, b_var):
20      for i in range(r_dim + b_dim):
21          # resource
22          if i < r_dim:
23              a[i] = np.clip(np.random.normal(a[i], r_var), 0, 1) * r_bound
24          # bandwidth
25          elif i < r_dim + b_dim:
26              a[i] = np.clip(np.random.normal(a[i], b_var), 0, 1) * b_bound
27      return a
28
29  ####################  training  ####################
30
31  if __name__ == "__main__":
32      env = Env()
33      s_dim, r_dim, b_dim, o_dim, r_bound, b_bound, task_inf, limit, location = env.get_inf()
34      ddpg = DDPG(s_dim, r_dim, b_dim, o_dim, r_bound, b_bound)
```

```
35
36          r_var = 1    # control exploration
37          b_var = 1
38          ep_reward = []
39          r_v, b_v = [], []
40          var_reward = []
41          max_rewards = 0
42          episode = 0
43          var_counter = 0
44          epoch_inf = []
45          epoch = 0
46          while var_counter < LEARNING_MAX_EPISODE:
47              # 初期化
48              s = env.reset()
49              ep_reward.append(0)
50              if SCREEN_RENDER:
51                  env.initial_screen_demo()
52
53              for j in range(MAX_EP_STEPS):
54                  time.sleep(SLEEP_TIME)
55                  # render
56                  if SCREEN_RENDER:
57                      env.screen_demo()
58                  if TEXT_RENDER and j % 30 == 0:
59                      env.text_render()
60
61                  # DDPG
62                  # 状態に応じて行動を選択
63                  a = ddpg.choose_action(s)    # a = [R B O]
64                  # 探索のための行動選択にランダム性を持たせる
65                  a = exploration(a, r_dim, b_dim, r_var, b_var)
66                  # 遷移パラメータを格納
67                  s_, r = env.ddpg_step_forward(a, r_dim, b_dim)
68                  ddpg.store_transition(s, a, r / 10, s_)
69                  # 学習
70                  if ddpg.pointer == ddpg.memory_capacity:
```

```
                    print("start learning")
                if ddpg.pointer > ddpg.memory_capacity:
                    ddpg.learn()
                    if CHANGE:
                        r_var *= .99999
                        b_var *= .99999
                # 状態を置き換える（更新）
                s = s_
                # 報酬を集計
                ep_reward[episode] += r
                # 最後のエピソード
                if j == MAX_EP_STEPS - 1:
                    var_reward.append(ep_reward[episode])
                    r_v.append(r_var)
                    b_v.append(b_var)
                    print('Episode:%3d' % episode, ' Reward: %5d' % ep_reward[episode],
'###    r_var: %.2f ' % r_var,'b_var: %.2f ' % b_var, )
                    string = 'Episode:%3d' % episode + ' Reward: %5d' % ep_reward[episode]
+ '###    r_var: %.2f ' % r_var + 'b_var: %.2f ' % b_var
                    epoch_inf.append(string)
                    # variation change
                    if var_counter >= CHECK_EPISODE and np.mean(var_reward[-
CHECK_EPISODE:]) >= max_rewards:
                        CHANGE = True
                        var_counter = 0
                        max_rewards = np.mean(var_reward[-CHECK_EPISODE:])
                        var_reward = []
                        '''epoch += 1
                        print('学習エピソード数：', epoch)'''
                    else:
                        CHANGE = False
                        var_counter += 1
                        '''
                        epoch += 1
                        print('学習エピソード数：', epoch)

```

```
107                             '''
108
109             # エピソードの終了
110             if SCREEN_RENDER:
111                 env.canvas.tk.destroy()
112             episode += 1
113
114         # ディレクトリ作成
115         dir_name = 'output/' + 'ddpg_'+str(r_dim) + 'u' + str(int(o_dim / r_dim)) + 'e' +
116     str(limit) + 'l' + location
117         if (os.path.isdir(dir_name)):
118             os.rmdir(dir_name)
119         os.makedirs(dir_name)
120         # plot the reward
121         fig_reward = plt.figure()
122         plt.plot([i+1 for i in range(episode)], ep_reward)
123         plt.xlabel("episode")
124         plt.ylabel("rewards")
125         fig_reward.savefig(dir_name + '/rewards.png')
126         # plot the variance
127         fig_variance = plt.figure()
128         plt.plot([i + 1 for i in range(episode)], r_v, b_v)
129         plt.xlabel("episode")
130         plt.ylabel("variance")
131         fig_variance.savefig(dir_name + '/variance.png')
132
133         # write the record
134         f = open(dir_name + '/record.txt', 'a')
135         f.write('time(s):' + str(MAX_EP_STEPS) + '¥n¥n')
136         f.write('user_number:' + str(r_dim) + '¥n¥n')
137         f.write('edge_number:' + str(int(o_dim / r_dim)) + '¥n¥n')
138         f.write('limit:' + str(limit) + '¥n¥n')
139         f.write('task information:' + '¥n')
140         f.write(task_inf + '¥n¥n')
141         for i in range(episode):
142             f.write(epoch_inf[i] + '¥n')
```

```
143         # 平均値
144         print("the mean of the rewards in the last", LEARNING_MAX_EPISODE, " epochs:",
145     str(np.mean(ep_reward[-LEARNING_MAX_EPISODE:])))
146         f.write("the mean of the rewards:" + str(np.mean(ep_reward[-
147     LEARNING_MAX_EPISODE:])) + '¥n¥n')
148         # 標準偏差
149         print("the standard deviation of the rewards:", str(np.std(ep_reward[-
150     LEARNING_MAX_EPISODE:])))
151         f.write("the standard deviation of the rewards:" + str(np.std(ep_reward[-
152     LEARNING_MAX_EPISODE:])) + '¥n¥n')
153         # 報酬範囲（分散）
154         print("the range of the rewards:", str(max(ep_reward[-LEARNING_MAX_EPISODE:])
155     - min(ep_reward[-LEARNING_MAX_EPISODE:])))
156         f.write("the range of the rewards:" + str(max(ep_reward[-
157     LEARNING_MAX_EPISODE:]) - min(ep_reward[-LEARNING_MAX_EPISODE:])) +
158     '¥n¥n')
159         f.close
```