2016 年度 修士論文

自己適応システムのための
実行時環境モデル学習に関する研究

2017 年 1 月 30 日 (月) 提出

指導：深澤 良彰 教授

早稲田大学大学院 基幹理工学研究科
情報理工・情報通信専攻 深澤研究室

学籍番号：**5115F043-1**
田邉　萌香

目 次

第1章　はじめに

1.1　概要

環境の変化に対して，要求を満たしつづけるよう振る舞いを実行時に変更する，自己適応システム [23, 6] の必要性が高まってきている．近年の自己適応システムに関する研究 [8, 9] では，実行環境を離散的にモデル化し，安全性や活性といった要求充足を実行時に検査し，必要に応じて要求充足が保証された振る舞いに切替えることで自己適応を実現している．

しかしながら，この技術において，実行環境に沿わない誤った環境モデルを基に振る舞いを決定した場合，要求充足は保証されない．実行環境は不確実性を持つため，開発時に実行環境に沿った環境モデルを構築することは困難である [9]．また，実行時に，開発時に構築された環境モデルが持つ仮定から逸脱してしまう可能性がある．そのようなリスクを軽減するため，あらかじめ弱い仮定を持つ環境モデルを構築することがある．弱い仮定のもと構築されたモデルでは，高度な要求は保証できないが，環境が変化した際にも弱い仮定のもと動作を続けることが可能である．一方，強い仮定のもと構築された環境モデルでは，高度な要求が保証可能となるが，仮定から逸脱が頻繁に発生し，対応できなくなってしまう．したがって，実行環境との一貫性を維持するため，実行時に環境モデルを更新する必要がある．

環境モデルの構築に関しては，開発時にシステムをテスト実行し，そこで得られた履歴を基に環境モデルを学習する手法 [25, 7] が存在する．これらの手法は非実行時に環境モデルを学習することを想定している．学習に多くのデータを要することから，学習に時間がかかる．実行時に起こる変化を環境モデルに反映するには，実行時に素早く学習を行う必要があるため，既存手法は実行時に用いるのには適していない．

そこで本研究では，実行時に得られるデータから効率よく環境モデルを学習するために，環境モデルを差分学習する手法を提案する．データごとの計算が可能な確率的勾配降下法を応用し，差分学習をすることで，一度の学習に要する時間を削減し，実行時の学習を可能にする．評価では，自動倉庫管理システムの事例をもとに，既存の学習手法と本研究の差分学習手法について，学習の正確度とその収束性，学習に要する時間の比較を行う．

1.2　本論文の構成

本論文の構成は以下の通りである．２章では本研究の説明に用いる例題と，扱う環境モデルについて説明する．３章では環境モデルの学習に関する従来手法と，その課題について説明する．４章では実行時に適用可能な環境モデルの差分学習手法を提案する．５章では４章で提案した手法と従来手法の比較，評価を行う．６章で結論を述べる．

第2章　背景

2.1　自己適応システム

システムの実行環境の多様化により，実行時に環境が変化することを想定したシステム開発の必要性が高まってきている．しかしながら，そのような環境の変化をシステムの開発時に予測することは困難である．そこで，実行環境の変化に対して，要求を満たしつづけるよう，システム自身で実行時に振る舞いを変更するシステムである，自己適応システム [23, 6] に関する研究が近年多くなされている．

Zave らの研究 [28] では，システムが充足すべき要求 R，システムの振る舞い仕様 S，システムの実行環境に関する知識 D の関係を，次の式 2.1 のように表している．

$$S, D \models R \tag{2.1}$$

上式 2.1 は，「S は D において R を充足する」，ということを表している．前述のような実行環境の変化が生じた場合（すなわち D が D' に変化した場合），開発時の S では R は充足されず，式 2.1 は成立不可能となる場合がある．その際に，D' のもとで R を充足する新しい仕様 S' を決定，実行する必要があり，それをシステム自身で行うのが自己適応システムである [2]．

自己適応システムにおける適応ロジックは，MAPE ループモデルとしてモデル化される．このモデルは，Monitor（監視），Analyze（分析），Plan（計画），Execute（実行）という 4 つの過程によって構成される．これらの過程をシステム自身で繰り返すことで，自己適応が実現可能である．MAPE ループモデルとアプリケーションロジックを分離してシステムを構成することで，高い保守性を得ることも可能となる [5]．

Monitor システムの実行環境を監視する．実行環境に変化があれば，システムが持つ実行環境に関する知識を更新する．

Analyze 更新した実行環境において，要求を充足しているか否かを分析する．

Plan 要求が充足できていなければ，要求を充足するような新たな仕様を計画する．

Execute 決定した仕様に基づいて，システムの振る舞いを修正，実行する．

近年の自己適応システムに関する研究 [8, 9] では，離散制御器合成技術 [1, 22] を用いて，実行環境を離散的にモデル化し，安全性や活性といった要求充足を実行

時に検査し，必要に応じて要求充足が保証された振る舞い仕様に切替えることで，自己適応を実現している．離散制御器合成技術により自動生成される仕様は，要求充足が保証されているため，この技術を用いることで，要求充足が保証された振る舞いを実行する自己適応システムの実現が可能となる．次節にて本研究の説明に用いる例題について述べた後，離散制御器合成技術について説明する．

2.2 例題：自動倉庫管理システム

本研究の説明をするにあたり，自動倉庫管理システムを例題として扱う．この自動倉庫管理システムは，図 2.1 に示すような倉庫内をロボットが移動し，商品の出荷準備を行うようなシステムを想定している．

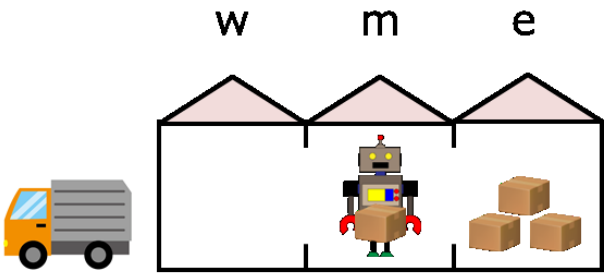

図 2.1: 自動倉庫管理システム

倉庫は次の 3 つのエリアで構成されている．

- エリア w：商品の出荷準備を行うエリア
- エリア m：２つのエリア w と e をつなぐ通路
- エリア e：商品が保管されているエリア

システムは，ロボットがエリア w にいる状態で実行を開始する．ロボットは，エリア e へ移動し，エリア e に保管されている商品を受け取り，エリア w まで運搬する，という一連の作業を繰り返し行う．ロボットはエリア間の移動と商品の持ち上げ下げを行うこと，また自身の現在地や商品の持ち上げ下げが成功したかどうかを認識することが可能である．

システムが実行中に充足しなければならない機能的な要求は次のとおりである．

要求１ ロボットはエリア e からエリア w へ商品を運ぶまで動作を続けなくてはならない．

要求２ ロボットは商品を持っている状態で商品を持ち上げる動作を行ってはならない．

要求３ ロボットは商品を持っていない状態で商品を下ろす動作を行ってはならない．

要求４ ロボットはエリア w を出たら，荷物の持ち上げ動作が成功するまでエリア w に戻らない．

要求５ ロボットはエリア e を出たら，荷物を下ろす動作が成功するまでエリア e に戻らない．

2.3 離散制御器合成

離散制御器合成 [1, 22] とは，形式化されたシステムが充足すべき要求と，システムの実行環境のモデルを入力とし，与えられた環境下での要求充足が保証されたシステムの振る舞い仕様である制御器を自動生成する技術である．この技術を自己適応システムに取り入れることで，要求充足が保証された振る舞いを実行する自己適応システムの実現が可能となる．本研究では特に，Modal Transition System Analyzer(MTSA)[11] というツールを用いて，離散制御器合成による仕様生成を行うことを想定している．入力となる形式的な要求と環境モデル，出力される制御器について，順に説明する．

2.3.1 形式的な要求

システムが充足すべき要求は，Fluent Linear Temporal Logic(FLTL)[15] によって形式的に記述し，MTSA の入力とする．これにより，安全性，活性に関する機能的な要求を扱うことができる．安全性とは，システムが常に満たすべき性質であり，活性とは，システムが常にいつかは満たすべき性質である．自動倉庫管理システムの例題において，2.2 節で挙げた 5 つの要求は，全て安全性に関する要求である．

2.3.2 環境モデル

本研究で扱う環境モデルは，システムとその外部環境との相互作用を Labelled Transition System(LTS)[18] によって離散的にモデル化したものである．Finite State Process(FSP) によって記述し，MTSA の入力とする．

FSP 記述の例を図 2.2 に示す．`MAP[‘{w, m, e}]` は倉庫内のエリア w にロボットが存在する状態を表しており，その状態で実行可能である制御可能な動作が `move[‘e]`，`move[‘w]` であること，また各制御可能な動作の結果として観測可能な動作が `arrive[‘{w, m, e}]` であることを表している．各観測可能な動作は，それを受理した場合の FSP 記述中の状態 `MAP[‘{w, m, e}]` と結び付けられている．

```
MAP['w]=(
    move['e]->(arrive['m]->MAP['m]
        |arrive['w]->MAP['w]
        |arrive['e]->MAP['e]
    )
    |move['w]->(arrive['w]->MAP['w]
        |arrive['m]->MAP['m]
        |arrive['e]->MAP['e]
    )
).
```

図 2.2: FSP 記述の例

LTS は，$E = (S, A, \Delta, s_0)$ と定義される．S は状態の集合，A は動作の集合，$\Delta \subseteq (S \times A \times S)$ は状態の遷移関係である． また s_0 は E の初期状態である．各遷移には動作 $a \in A$ のラベルが付いており，制御可能または観測可能な動作を表している．システムは，制御可能な動作を実行することで，環境側に何かしらの影響を与え，その影響を観測可能な動作として受理する．環境モデルは，これらの２種類の動作が交互に行われるような状態遷移を持ち，それによりシステムとその外部環境との相互作用を表している．

自動倉庫管理システムの例題では，倉庫の状態，ロボットの状態，荷物の状態等が環境となる．この例題における環境モデルを，図 2.3 に示す．

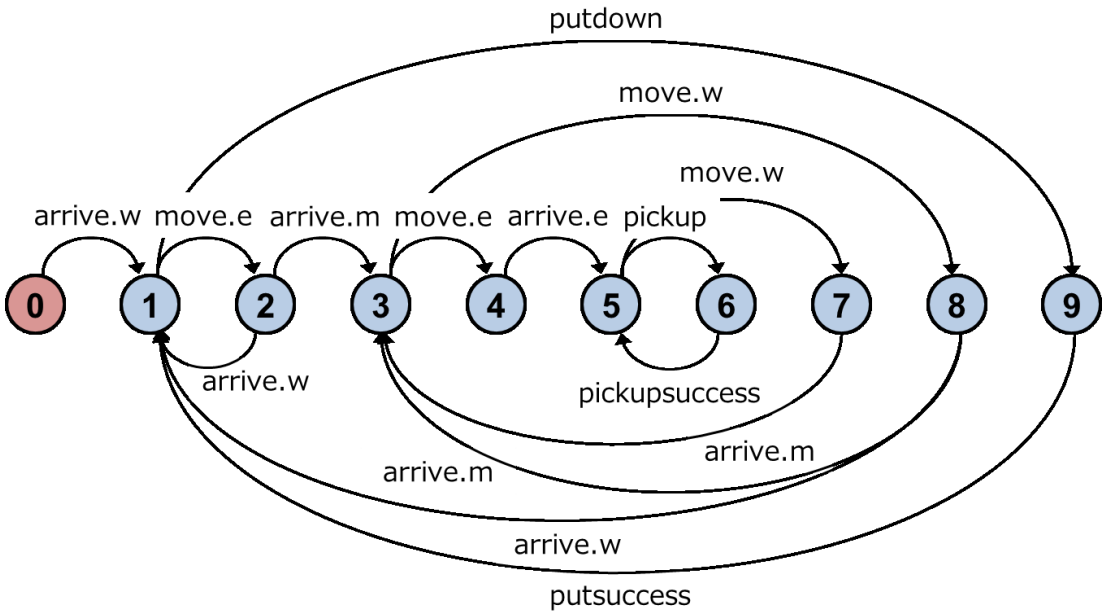

図 2.3: 自動倉庫管理システムの環境モデル

図 2.3 における制御可能な動作，観測可能な動作は次のとおりである．

- 制御可能な動作
  - move.{e，w}：ロボットをエリア {e, w} に向かって移動させる
  - pickup：ロボットに商品を持ち上げる動作をさせる
  - putdown：ロボットに商品を下ろす動作をさせる
- 観測可能な動作
  - arrive.{e，m，w}：ロボットがエリア {e, m, w} へ到達した
  - pickupsuccess：ロボットが pickup に成功した
  - putsuccess：ロボットが putdown に成功した

例えば，図 2.3 の環境モデルにおいて，初期状態（状態 0）からシステムが実行を開始し，ロボットがエリア w にいる場合（すなわち，arrive.w が受理された状態 1 において），システムは move.e または putdown という制御可能な動作が実行可能である．ここで move.e を実行して状態 2 に遷移した場合，エリア m に到達する（arrive.m が受理されて状態 3 に遷移する），またはエリア w に到達する（arrive.w が受理されて状態 1 に遷移する）ことが想定されている．

2.3.3 生成される制御器

MTSA を用いることで，前述の要求と環境モデルから，システムの振る舞い仕様である制御器が生成される．制御器は，与えられた環境モデル下で，システムの制御可能な動作により与えられた要求が充足可能かどうかをゲーム理論を用いて分析し，可能であれば生成される．ここで生成される制御器は，環境モデルと同様に LTS としてモデル化される．システムは，制御器に基づいて制御可能な動作を行うことで，与えられた要求を充足することが保証されている．

自動倉庫管理システムにおいて，MTSA によって自動生成される制御器を図 2.4 に示す．例えば，ロボットがエリア w にいる場合（すなわち，arrive.w が受理された状態 3 において），システムは move.e という制御可能な動作を実行する．その結果エリア m に到達した場合（すなわち，arrive.m が受理された状態 7 において）は，move.e という制御可能な動作を実行し，エリア w に到達した場合（すなわち，arrive.w が受理された状態 3 において）は，move.e という制御可能な動作を実行する．このように実行と観測を繰り返すことで，システムは与えられた要求の充足が可能である．

2.4 環境の変化

システムの振る舞いは，開発時に想定した環境モデルの下で動作した場合に，与えられた要求を充足するように決定される．しかしながら，実行環境が変化し，環

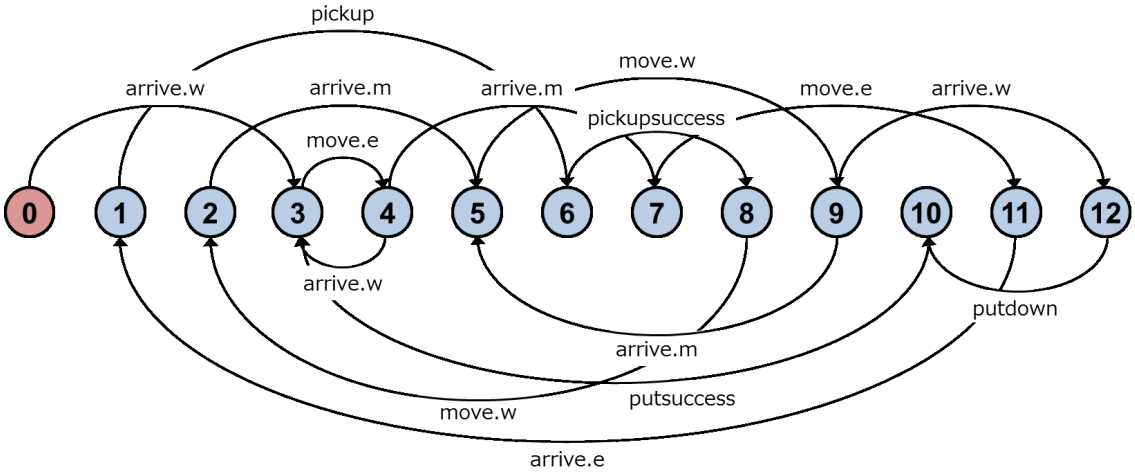

図 2.4: 自動倉庫管理システムの制御器

境モデルとの間に差異が生じてしまった場合，開発時に決定された振る舞いでは与えられた要求が充足不可能となる可能性がある．

実行環境は不確実性を持つため，開発時に実行環境を正確に表現する環境モデルを構築することは困難である [9]．また，実行時に，開発時に構築された環境モデルが持つ仮定から逸脱してしまう可能性がある．そのようなリスクを軽減するため，あらかじめ弱い仮定を持つ環境モデルを構築することがある．弱い仮定のもと構築されたモデルでは，高度な要求は保証できないが，環境が変化した際にも弱い仮定のもと動作を続けることが可能である．一方，強い仮定のもと構築された環境モデルでは，高度な要求が保証可能となるが，仮定からの逸脱が頻繁に発生し，対応できなくなってしまう．したがって，実行環境との一貫性を維持するため，実行時に環境モデルを更新する必要がある．

自己適応システムは，新たな環境モデルの構築，その下で要求を充足するような新たな振る舞いの決定や変更を実行時にシステム自身で行うことで，環境の変化に対応することができる．その際に，環境の変化をシステム自身が正確に認識し，モデル化することが求められる．

環境の変化について，例題をもとに説明する．自動倉庫管理システムにおいて，「エリア w とエリア m の間が商品で塞がれて通行不可能となる」，という物理環境の変化が生じたとする．この時の環境モデルを図 2.5 に示す．点線の矢印は環境の変化によって削除された遷移を表している．この変化をシステム自身が認識し，環境モデルに反映することができなかった場合，エリア m で商品を持っている状態（図 2.4 における状態 5）のロボットは，与えられた振る舞い仕様に基づいて `move.w` という動作を実行する（状態 9 へ遷移する）．図 2.3 の環境モデル上では `arrive.w` が観測されることが想定されているが，実際の環境下では `move.w` によって `arrive.w` を観測することはなく，`arrive.m` が観測される．したがって，再度 `move.w` を実行し，`arrive.m` を観測する，という過程を繰り返し，「商品をエリ

アwに運ぶ」という要求は充足不可能となってしまう．ここで，環境の変化を正しく認識することができれば，代替動作に切り替える等の方法で「商品をエリアwに運ぶ」ことが可能となる．

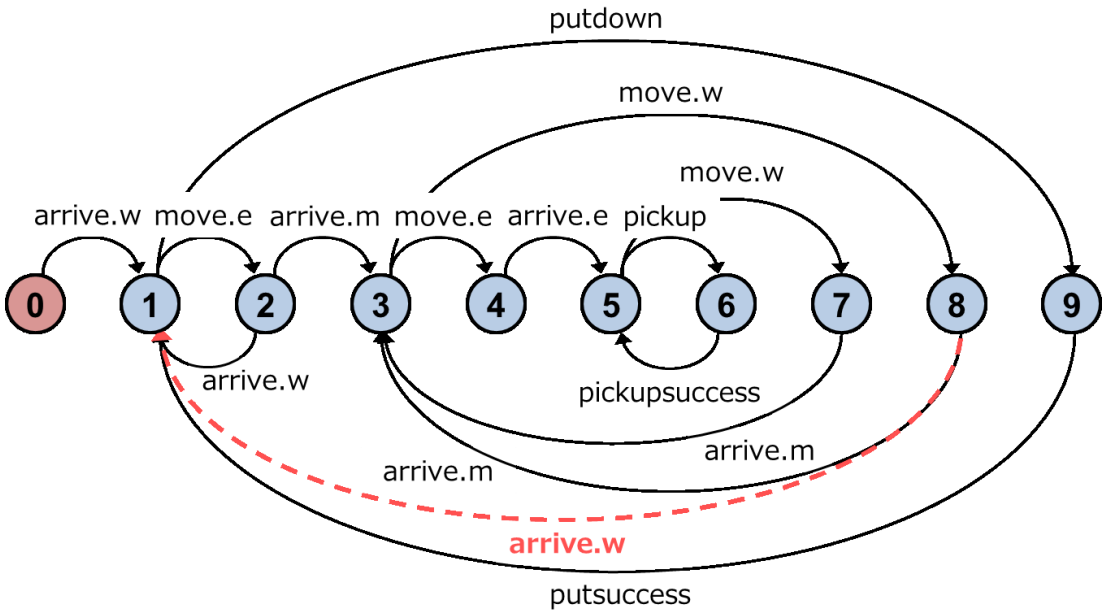

図 2.5: 自動倉庫管理システムの環境モデル（変化後）

自己適応システムは，新たな環境モデルの構築，その下で要求を充足するような新たな振る舞いの決定や変更を実行時にシステム自身で行うことで，環境の変化に対応することができる．その際に，環境の変化をシステム自身が正確に認識，モデル化すること，また，システムの実行に支障をきたさないよう，短い時間で環境の変化を学習することが求められる．D'Ippolito らの研究 [9] では，離散制御器合成技術を用いて，事前に想定した環境の変化にのみ対応可能な自己適応システムを実現している．これに対し，本研究では，事前に想定することなく環境の変化に対応可能な自己適応システムの実現のため，実行時に実行環境を正確に表現する環境モデルを素早く学習することを目的とする．

第3章　関連研究

3.1　自己適応システムに関する研究

2で述べた，離散制御器技術を用いた自己適応システムの実現に関する研究 [9, 10] について説明する．これらの研究では，異なる仮定を持つ複数の環境を予め想定している．段階的な仮定を持つ複数の環境モデル，その環境モデル下で充足可能な要求，制御器の組を用意し，動作の観測結果に応じて実行時に制御器を切り替えることで，環境の変化に対処している．強い仮定を持つ環境モデルの組から弱い仮定を持つ環境モデルの組へ切り替える際の要求緩和も実現している．しかしながら，予め想定した環境と変化後の環境が一致するとは限らず，その差によっては過剰な要求緩和が行われてしまう場合も存在する．したがって，どの程度の段階に分けて環境モデルを用意するかが課題となる．本研究では，予め段階的な仮定を持つ環境モデルを想定するのではなく，動作の観測結果から，実行時に実行環境を正確に表現する環境モデルを学習することを目的とする．

3.2　環境モデルの非実行時学習に関する研究

Fahland ら [13] や Ding ら [7] の研究では，プロセスマイニングにより環境を学習する手法が提案されている．環境はペトリネットとしてモデル化されている．ペトリネットとは，アクティビティ間の関連が記述されたプロセスモデルである．ペトリネットとシステムの実行ログを用意し，実行ログが再現可能かつできる限り直前のモデルに近いようなペトリネットを，プロセスマイニングにより学習している．マイニングの技術を用いた学習手法は，Yuan ら [27] の研究においても提案されている．この研究では，システムの実行中に発生したトランザクション間の関連をデータマイニングにより抽出している．

Nikravesh らの研究 [21] では，クラウドコンピューティングにおけるオートスケーリング技術に着目し，サーバにかかる負荷の予測をサポートベクトルマシン (SVM: Support Vector Machine)，ニューラルネットワーク (NN: Neural Network) を用いて行っている．負荷のかかり方と 2 つの予測アルゴリズムの予測精度の関係について検証し，観測した負荷のかかり方に応じて予測アルゴリズムを切り替えることによるオートスケーリングの精度向上を目指している．

Sykes ら [25] や Martínez ら [19] の研究では，環境は論理プログラムとしてモデル化されている．Sykes らの研究では，論理プログラムが持つ規則を学習する，NoMPRoL が提案されている．NoMPRoL では，テスト実行により得られた実行トレースから，解集合プログラミングを用いて仮説群を抽出している．各仮説は頭部（システムが取り得る動作）と本体（頭部の動作の成功条件群）で構成されている．本体が持つ各条件の尤もらしさを勾配降下法を用いて学習し，尤もらしい条件を持つ仮説を規則とし，論理プログラムを構築している．

以上の研究では，学習は非実行時に行われている．本研究では学習は実行時に行うことを想定しているため，次節では実行時の学習に関する研究について説明する．

3.3 環境モデルの実行時学習に関する研究

Ghezzi ら [14] の研究では，環境はマルコフ決定過程を用いてモデル化されている．マルコフ決定過程とは，確率的で非決定的な遷移を持つ有限状態機械である．システムの状態と起こり得る遷移は開発時に全て用意されており，各遷移によって得られる報酬を実行時に更新している．システムが持つ非機能的な要求の達成度を報酬としており，この研究では報酬の最大化を扱っている．これに対して，本研究では機能的な要求の充足を扱う．

強化学習を用いた実行時学習に関する研究も多く存在する．強化学習では，エージェントは環境に関する知識を探索・活用する．Godoy ら [16] の研究では，マルチエージェントのナビゲーション問題を扱っており，各エージェントに目的地へ到達するまでの適した振る舞いを学習させている．Menashe ら [20] の研究では，階層的に表されるモデルの学習を強化学習により行っている．Sharifloo ら [24] の研究では，動的ソフトウェアプロダクトラインにおけるシステム構成を強化学習によって学習している．強化学習においてシステムが知識を探索する際に，試行によってシステムが持つ機能的要求が充足されなくなる可能性がある．したがって，要求充足の保証を扱う本研究では強化学習は用いない．

第4章　従来手法による実行時学習の実現

本研究では，システムと環境との相互作用を表す環境モデルを学習することを目的としている．そこで，3 章で説明した Sykes らの研究 [25] をもとに，システムと環境との相互作用を記録した実行トレースからその関係性を学習することを考える．この研究では学習は勾配降下法によって行われているため，本章では，勾配降下法による環境モデルの実行時学習の実現方法について，説明する．

4.1　離散制御器合成技術を用いた自己適応システムの構成

離散制御器合成技術と実行時学習を用いた自己適応システムの構成について説明する．システムと環境，MAPE ループの関係を図 4.1 に示す．MAPE ループの各過程では，次のような処理を行う．

Monitor 制御器に従って動作するシステムが実行した制御可能な動作，受理した観測可能な動作を監視する．監視した動作を実行トレースとして記録し，それをもとに環境モデルを学習，更新する．

Analyze 更新した環境モデルにおいて，要求を充足しているか否かを分析する．必要に応じて要求緩和を行う [31]．

Plan 要求が充足できていなければ，更新した環境モデルと要求をもとに，新たな制御器を離散制御器合成技術によって生成する．

Execute 生成した制御器をシステムに適用する．

本研究では特に，Monitor 部分の環境モデルの学習に着目している．実行した制御可能な動作，受理した観測可能な動作から，実行時に実行環境を正確に表現する環境モデルの学習をすることで，環境の変化に対処する．次節より，勾配降下法を用いた実行時の環境モデル学習について説明する．

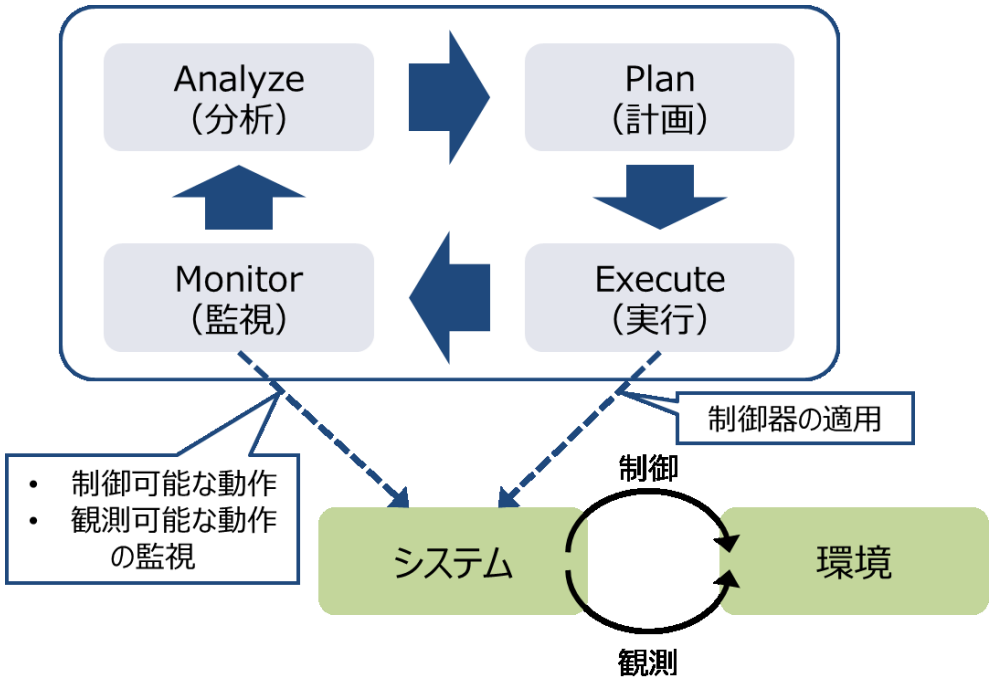

図 4.1: 実現したい自己適応システムの構成

4.2 勾配降下法

勾配降下法 (GD: Gradient Descent) は，パラメータ $\mathbf{p}$ を引数とする目的関数 $L(\mathbf{p})$ の値を最小化するための手法である．[30] 目的関数の勾配をもとにパラメータの調整をすることで，目的関数が凸であれば最小値を求めることができる．目的関数が凸でない場合は局所解が得られる可能性もある．解を得るまでの計算回数はパラメータの調整方法等により異なり，より速く解を得るための手法について様々な研究がなされている．

勾配降下法では，学習のために与えられた N 個のデータのうち i 番目のデータによって得られる値を $l_i(\mathbf{p})$ とすると，目的関数は次の式 4.1 ように表すことができる．

$$L(\mathbf{p}) = \sum_{i=1}^{N} l_i(\mathbf{p}) \tag{4.1}$$

パラメータの更新は，次の式 4.2 を用いて目的関数が収束するまで行われる．

$$\mathbf{p} = \mathbf{p} - \eta \nabla L(\mathbf{p}) \tag{4.2}$$

η は学習率である．学習率を変化させることにより，解を得るまでの計算回数も変化する．

4.3 勾配降下法による環境モデルの学習

勾配降下法を用いた環境モデルの学習手法について説明する．アルゴリズムを Algorithm1 に示す．

Algorithm 1 勾配降下法による学習のアルゴリズム

Input: R，ζ，$actionSets$
Output: updated R

for all $r \in R$ **do**
  for all $b \in r.B$ **do**
    $//r.B$ は r が持つ事後条件群
    $\theta_b = \theta_b - \eta \frac{\partial MSE_{gd}}{\partial \theta_b}$
  end for
  for all $b \in r.B$ **do**
    $\theta_b = \theta_b / sum(r.B)$
    $//sum(r.B)$ は $\theta_{b \in B}$ の合計値
    if $\theta_b \leq \zeta$ **then**
      $b.rule \leftarrow false$
    else
      $b.rule \leftarrow true$
    end if
  end for
end for
return R

学習の入力となるデータは，次の 3 つである．

1. アクションセット群 $actionSets$
2. 規則群 R
3. 閾値 ζ

システムは，実行した制御可能な動作と受理した観測可能な動作を，実行トレースとして記録する．自動倉庫管理システムの実行トレース例を図 4.2 に示す．アクションセットは，図 4.2 のように，実行トレースから，制御可能な動作とその前後の観測可能な動作を抽出したものとする．ここで，抽出した制御可能な動作は「動作」，その前後の観測可能な動作はそれぞれ「事前条件」,「事後条件」と呼ぶこととする．勾配降下法による学習では，図 4.3 のように，学習時点から過去のある一定期間に得られたアクションセット群 $actionSets$ を学習の入力とする．

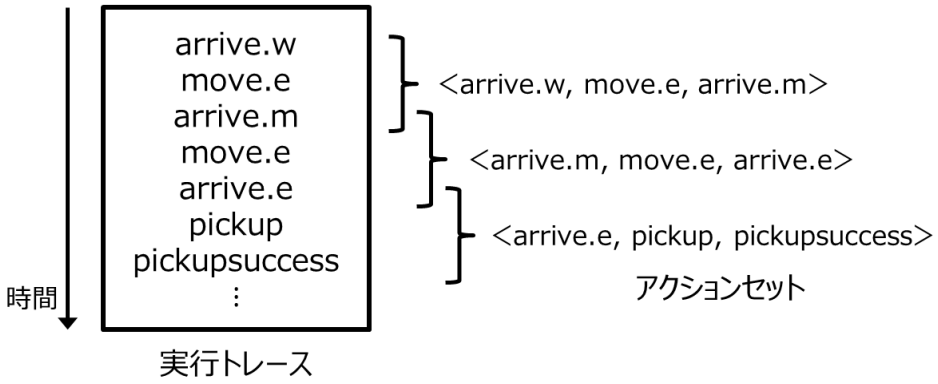

図 4.2: アクションセットの抽出

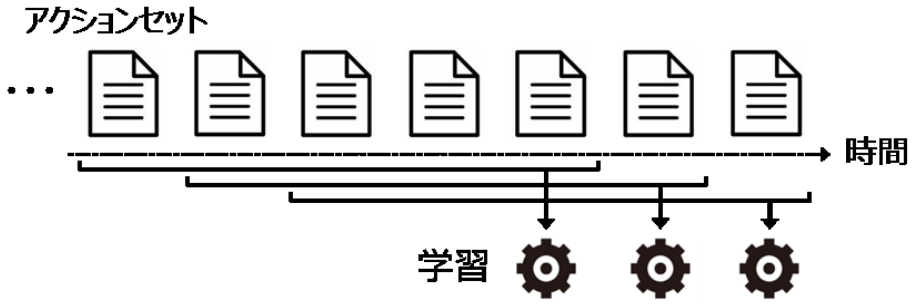

図 4.3: 勾配降下法による学習

R は，事前条件，動作，その結果観測される可能性のある事後条件群の組の集合であり，各規則は次のように表される．

$$< pre\text{-}condition, action, post\text{-}conditions\{\alpha,\beta,\gamma,...\} >$$

学習では各規則が持つ事後条件群の観測確率を推定する．環境モデルは，推定観測確率が高い事後条件を含む規則によって構成される．ζ は環境モデルへ採用する事後条件を決定する際に用いる値である．本研究で扱う環境モデルは LTS であり，確率的なモデルではない．そこで，推定した各事後条件の観測確率をもとに，ある値よりも高い推定観測確率を持つ事後条件のみを LTS としてモデル化する．その推定観測確率の閾値を ζ とする．R と ζ は実行時ではなく開発時に入力する．

勾配降下法によって最小化する誤差関数は式 4.3 のように定義する．これは，実行トレースとシステムが持つ規則の差を表す関数となっている．各規則 $r \in R$ が持つ各事後条件の観測確率を推定し， 推定された観測確率が一定値を超える事後条件を含む規則をもとに，環境モデルを構築する．学習は，推定観測確率に関する比の値をパラメータとし，式 4.3，式 4.4 に基づいて行う．

$$MSE_{gd}(\mathbf{p}) = \frac{1}{X_c} \sum_{j=1}^{X_c} (1 - P(x_j|B_c))^2 \tag{4.3}$$

$$P(x_j|B_c) = \frac{\sum_{\{b \in B_c, b| = x_j\}} \theta_b}{\sum_{\{b \in B_c\}} \theta_b} \tag{4.4}$$

c は事前条件と動作の組，B_c は c を持つ規則 r_c が持つ事後条件群，X_c は c の観測回数，x_j は X_c のうち j 番目に観測された事後条件，θ_b は事後条件 $b \in B_c$ の推定観測確率に関する比の値である．各パラメータは，式 4.3 をもとに，誤差関数が収束するまで更新される（Algorithm1，4 列目）．

$$\mathbf{p}_{t+1} = \mathbf{p}_t - \eta \nabla MSE_{gd}(\mathbf{p}_t) \tag{4.5}$$

$\mathbf{p}_t$ は t における $\theta_{\{b \in B_c\}}$ の値のベクトル，η は学習率である．ここで得られた $\mathbf{p}$ の値から，事後条件の推定観測確率を求め（7 列目），予め与えている閾値 ζ と比較し（9 列目），環境モデルへ採用する事後条件を決定する．$b.rule$ が true であれば b は環境モデルへ採用し，$b.rule$ が false であれば b は不採用とする．採用された事後条件を含む R を FSP 記述に変換し，LTS として表される環境モデルを構築する．R の FSP 記述への変換は，FSP 中の状態と制御可能な動作，観測可能な動作の関係を事前に定義し，それをもとに行う．

4.4 従来手法の課題

従来手法 [25] では，テスト実行段階で得られたデータをもとに，システムの実行前に学習を行っている．これに対し，本研究では，実行時に起こる変化を環境

モデルに反映するため，実行時に得られたデータをもとに実行時に学習を行うことを想定している．システムの実行に支障をきたさないよう，環境の変化は素早く環境モデルへ反映することが求められるが，前節の勾配降下法を用いた従来の学習手法では，学習に要する時間が課題となる．

従来手法では，学習の入力とするデータ量が増加するほど学習結果の正確度は高まるが，同時に計算時間も増加する．計算時間の削減のためにデータ量を削減することも可能ではあるが，正確度の点で限界がある．また式 4.4 の計算回数は，事後条件数，規則数，実行トレース長等，様々な要素に依存する．したがって，システムの規模の拡大により，計算回数，学習時間が大幅に増加してしまうことが予想される．自動倉庫管理システムの例題においても，エリア数の増加により規則数が増加した場合，計算回数も増加し，一度の学習に 10 秒以上の時間を要してしまうことがある．システムの実行時に新しいアクションセットが観測される度に学習をすることを考えると，この計算時間は現実的ではない．

また，環境の変化を学習するために，従来手法では新しい環境下で得られるデータを多く必要とする．しかしながら，実行時に得られるデータ量は限られており，十分なデータ量を得るまでにも多くの実行時間を要する．これはシステムの実行に支障をきたす大きな要因となってしまう．これらのことから，従来手法では正確かつ素早い実行時学習の実現が困難である．

第5章　環境モデルの実行時差分学習

本研究では，実行時に得られるアクションセットから効率良く環境モデルを学習するため，環境モデルを差分学習する手法を提案する．確率的勾配降下法 [3] を応用することで，学習に用いるデータ量と計算時間を削減し，実行時の学習を可能にする．

5.1　確率的勾配降下法

確率的勾配降下法 (SGD: Stochastic Gradient Descent)[3, 30] は，勾配法の一種であり，1 つのデータを読み込んだ際にそのデータのみを使って勾配を計算し，パラメータを更新する手法である．したがって，学習のために与えられたデータによって得られる値を $l(\mathbf{p})$ とすると，目的関数 $L(\mathbf{p})$ は次の式 5.1 のように表される．

$$L(\mathbf{p}) = l(\mathbf{p}) \tag{5.1}$$

上式 5.1 は勾配降下法における式 4.1 に相当するものである．パラメータの更新は，勾配降下法と同様の式 4.2 をもとに行われる．一度のパラメータ更新における計算量が勾配降下法よりも小さいため，大規模なデータに対して有効であるとされている [4].

確率的勾配降下法では，与えられた全てのデータをランダムに並べ替え，順番に 1 つずつ選択し，勾配の計算とパラメータの更新を行う．全てのデータをもとにパラメータを更新した後，再度全てのデータをランダムに並べ替え，パラメータの更新を行う．パラメータの更新は，指定回数，または目的関数の値が収束するまで行われる．

5.2　本手法の特徴

勾配降下法を用いる従来手法と，確率的勾配降下法を用いる手法，本差分学習手法の違いを図 5.1 に示す．

勾配降下法を使った従来手法では，4 章で述べたように，学習時点から過去のある一定期間に得られたアクションセットを入力として，入力された全てのアクションセットをもとに勾配の計算とパラメータの更新を行う．確率的勾配降下法を用いる場合も，従来手法と同様に過去のある一定期間に得られたアクションセット

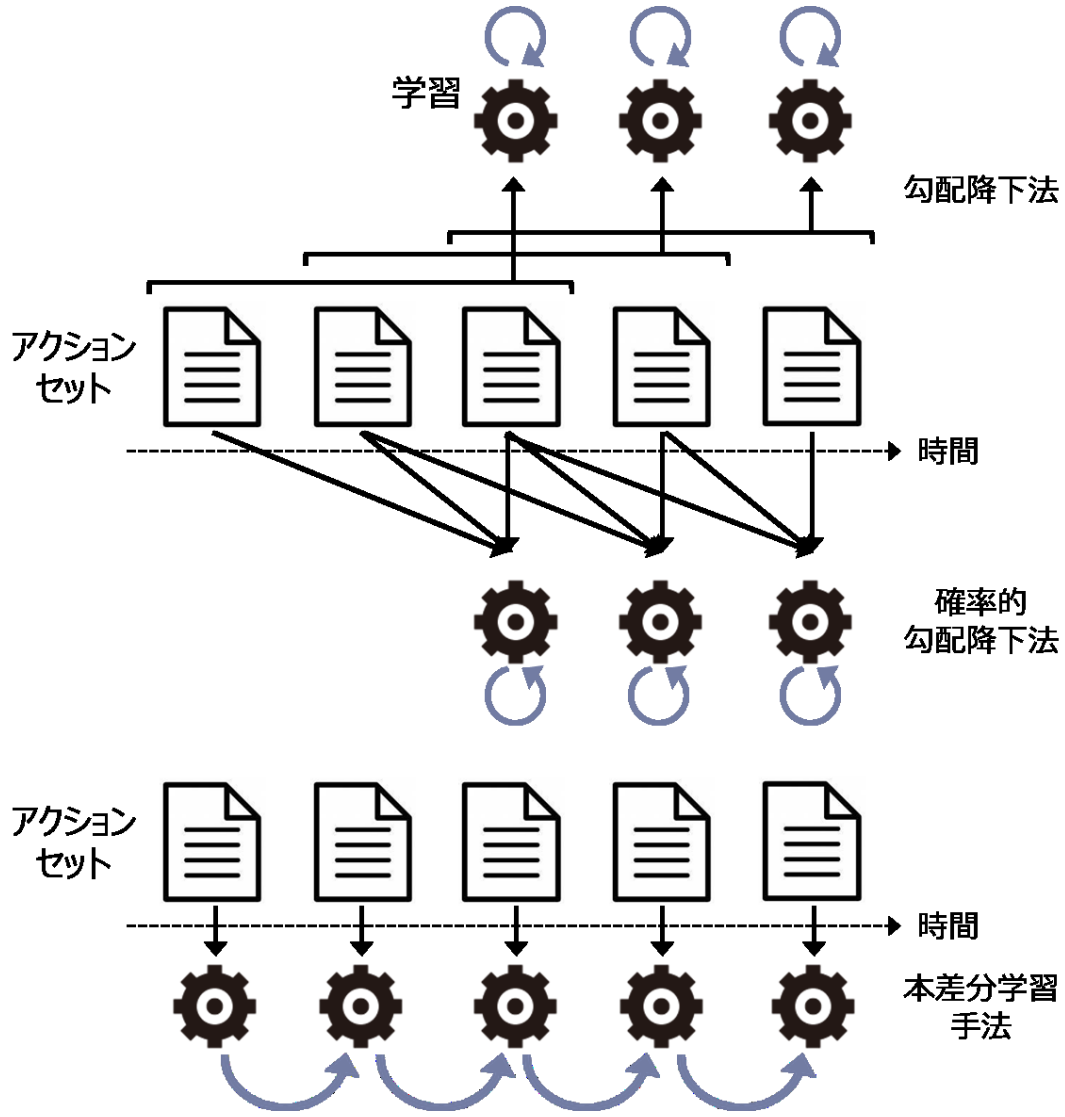

図 5.1: 従来手法と提案手法の違い

をもとに学習を行う．ただし，従来手法とは勾配の計算方法が異なり，ランダムに選択した 1 つのデータをもとに勾配の計算を行い，パラメータを更新する．これらの手法では，パラメータ更新は指定回数または誤差関数が収束するまで繰り返される．つまり，過去のある一定期間における各事後条件の観測確率を，学習時点の観測確率として推定している．

これに対して本手法では，前の学習結果を引き継ぎ，時系列に沿って得られる 1 つのデータをもとに，パラメータの更新を行う． 1 つのアクションセットのみをもとにパラメータを更新するという点は，確率的勾配降下法と同様である．また，計算時間削減のため，各アクションセットは一度だけ計算に用いることとする．過去に得られたアクションセットを考慮せず，学習時点で得られたアクションセットのみに着目した差分更新をすることで，学習時間の削減を実現する．

5.3 概要

環境モデルの差分学習手法の概要を図 5.2 に示す．

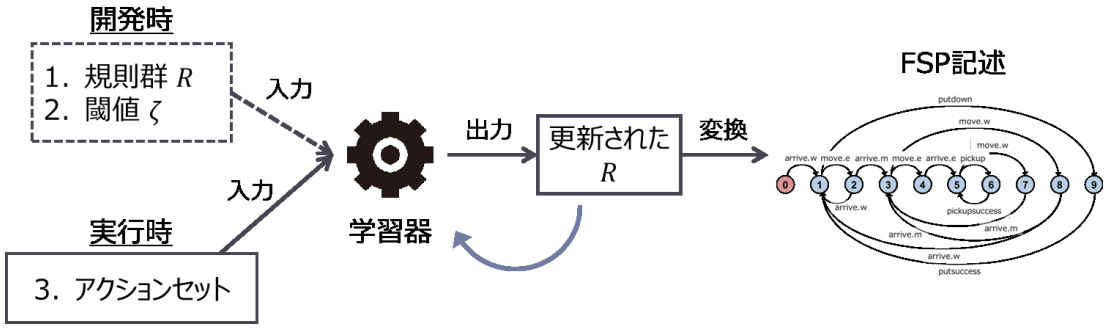

図 5.2: 実行時差分学習手法の概要

学習の入力は，次の 3 つである． 1 つ目のアクションセットのみ，従来手法と異なる．

1. アクションセット $< pre_o, a_o, b_o >$
2. 規則群 R
3. 閾値 ζ

規則群 R と閾値 ζ は予め与えておき，新しいアクションセットが得られる度に，差分学習を行う．pre_o，a_o，b_o は，それぞれ事前条件，動作，事後条件である．学習の出力は更新された規則である．これが環境モデルと異なる場合は，得られた規則を FSP 記述に変換し，環境モデルを更新する．

5.4 実行時差分学習手法

提案する実行時差分学習手法について，詳述する．図 5.2 における学習器である，差分学習のアルゴリズムを Algorithm2 に示す．

Algorithm 2 差分学習のアルゴリズム

Input: R，ζ，$< pre_o, a_o, b_o >$(得られたアクションセット)
Output: updated R

1: **for all** $r \in R$ **do**
2: **if** $r.pre == pre_o$ and $r.a == a_o$ **then**
3: **for all** $b \in r.B$ **do**
4: $\theta_b = \theta_b - \eta \frac{\partial MSE_{sgd}}{\partial \theta_b}$
5: **end for**
6: **for all** $b \in r.B$ **do**
7: $\theta_b = \theta_b / sum(r.B)$
8: **if** $\theta_b \leq \zeta$ **then**
9: $b.rule \leftarrow false$
10: **else**
11: $b.rule \leftarrow true$
12: **end if**
13: **end for**
14: **end if**
15: **end for**
16: **return** R

前述のとおり，新しいアクションセットが得られる度に，Algorithm2 によって差分学習を行う．まず，得られたアクションセットと同様の事前条件，動作を持つ規則を抽出する（2 列目）．抽出された規則について，その規則が持つ各事後条件 b の観測確率を，確率的勾配降下法のパラメータ更新手法をもとに推定する (4 列目)．

誤差関数 MSE_{sgd} は式 5.2 のように定義する．この誤差関数の勾配をもとにパラメータ $\mathbf{p}$ を更新する．この式は，勾配降下法における 4 章の式 4.3 に相当するものである．

$$MSE_{sgd}(\mathbf{p}) = (1 - P(x_j|B_c))^2 \tag{5.2}$$

上式において，$P(x_j|B_c)$ の計算には 4 章の式 4.4 を用いる．$\mathbf{p}$ の更新には，次の式 5.3 を用いる．

$$\mathbf{p}_{t+1} = \mathbf{p}_t - \eta \nabla MSE_{sgd}(\mathbf{p}_t) \tag{5.3}$$

$\mathbf{p}$ は，この式 5.3 を用いて一度だけ更新される．

以降は従来手法と同様の処理を行う．更新された $\mathbf{p}$ の値から，事後条件の推定観測確率を求め（7 列目），入力した ζ をもとに，環境モデルへ採用する事後条件を決定する (8 から 12 列目)．$sum(r.B)$ は $\theta_{b \in B}$ の合計値である．$b.rule$ が true であれば b は環境モデルに採用し，$b.rule$ が false であれば b は不採用とする．

ここで，自動倉庫管理システムの例題を用いて，具体的な計算例を示す．システムの実行中に，次のアクションセットが得られたとする．

$$< arrive.m, move.w, arrive.m >$$

この場合，学習のために抽出される規則は次のような規則である．

$$< arrive.m, move.w, \{arrive.m, arrive.e, arrive.w\} >$$

この規則は，得られたアクションセットと同じ事前条件と動作，また 3 つの事後条件 $b1$（$arrive.m$），$b2(arrive.e)$, $b3(arrive.w)$ を持つ．各事後条件は，それぞれパラメータ θ_{b1}，θ_{b2}，θ_{b3} を持つ．これらのパラメータは，式 5.3 をもとに更新される．更新されたパラメータの値が $\theta_{b1} = 1.4$，$\theta_{b2} = 0.2$，$\theta_{b3} = 0.4$ であった場合，各事後条件の推定観測確率は次のようにして求められる．

$$b1 : \frac{\theta_{b1}}{\theta_{b1} + \theta_{b2} + \theta_{b3}} = 0.7$$

$$b2 : \frac{\theta_{b2}}{\theta_{b1} + \theta_{b2} + \theta_{b3}} = 0.1$$

$$b3 : \frac{\theta_{b3}}{\theta_{b1} + \theta_{b2} + \theta_{b3}} = 0.2$$

ここで得られた値は事前に入力された ζ と比較される．ζ が 0.15 の場合，$\theta_{b1}(0.7)$ と $\theta_{b3}(0.2)$ は ζ よりも大きいため，$b1$ と $b3$ は環境モデルへ採用される．一方，$\theta_{b2}(0.1)$ は ζ よりも小さいため，$b2$ は不採用となる．したがって，環境モデルの構築に用いられる規則は次のようになる．

$$< arrive.m, move.w, \{arrive.m, arrive.w\} >$$

この規則がその時点での環境モデルが持つ規則と異なる場合は，得られた規則をもとに環境モデルを更新する．

更新された環境モデルは，決定的である動作（制御可能な動作）と，その結果観測される非決定的な 1 つ以上の事後条件（観測可能な動作）によって構成される．本手法を用いることで，ある閾値以上の推定観測確率を持つ複数の選択不可能な事後条件を考慮したシステムの振る舞い仕様の生成が可能となる．

以上のように，確率的勾配降下法を応用することで，少ないデータ量での学習，また計算回数の増加の抑制が可能となる．時系列に沿って得られた 1 つのデータのみを用いた差分学習は，学習時間の削減につながる．また，計算回数の増加につながる要素も事後条件数のみであり，システムの規模の拡大による計算回数，学習時間の増加も抑制可能であることが推測できる．

第6章　評価

環境モデルの学習の正確度とその収束性，学習に要する計算時間について，従来手法との比較により評価する．２つの異なる規模の自動倉庫管理システムを例題とし，以下の３つの研究課題について，ケーススタディを行う．

研究課題１　どの程度の正確度で学習ができるのか．

研究課題２　正確度の収束性に違いはあるか．

研究課題３　一度の学習に要する計算時間はどの程度削減できるのか．

6.1　２つの例題

規模の異なる２つの自動倉庫管理システムの例題について説明する．各例題の設定を表 6.1 に示す．

表 6.1: 各例題の設定

設定	小規模	大規模
エリア数	3	153
規則数	10	1,171
事後条件数	26	5,259

小規模な例題は，2.2 節で説明した図 2.1 に示されるようなシステムである．この例題は 10 個の規則を持ち，各規則が持つ事後条件数の和は 26 である．

大規模な例題は，前述の例題よりも広い倉庫内でロボットが商品の出荷準備を行うようなシステムを想定している．倉庫は図 6.1 のように，153 のエリア（縦 15 ×横 10 ＋ 3 エリア）で構成されている．ロボットは，初期位置から移動を始め，（１）箱受け取りエリア（空箱を受け取るエリア），（２）商品箱詰めエリア（商品を箱に梱包するエリア），（３）出荷エリア（商品の出荷準備を行うエリア）を順に訪れる．システムが制御可能な動作は，ロボットへの四方向への移動と商品の持ち上げ下げの指示である．また観測可能な動作は，ロボットがどのエリアへ到達したか，ロボットの商品の持ち上げ下げが成功したか否か，である．この例題は，小規模な例題の約 12 倍である 1,171 個の規則を持ち，各規則が持つ事後条件数の和は 5,259 である．

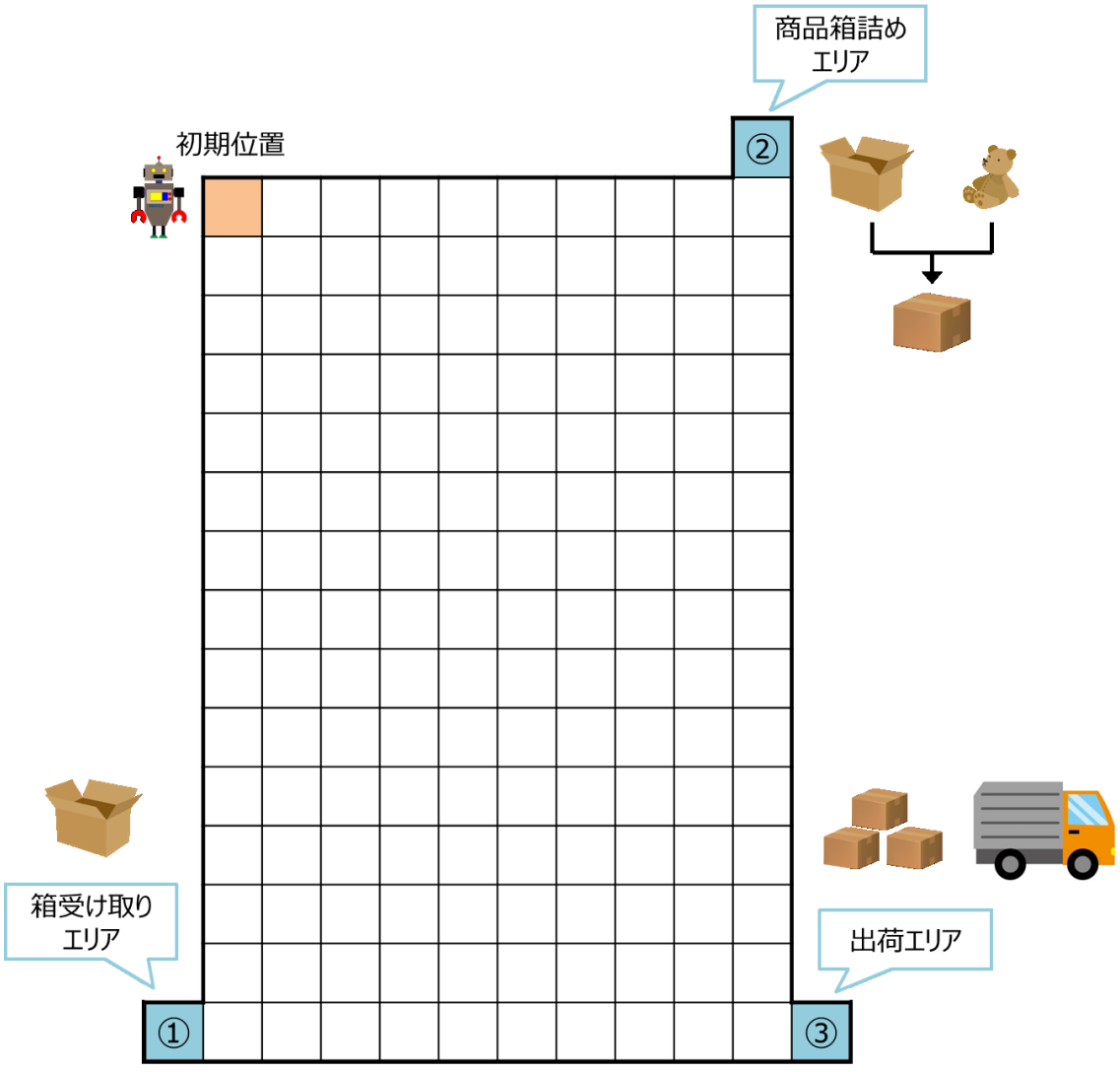

図 6.1: 自動倉庫管理システム（大規模）

6.2 評価方法

6.2.1 評価指標

研究課題 1，2 については，誤差の大きさをもとに環境モデルの学習の正確度を評価する．誤差は,「実行トレース生成時に設定した事後条件の真の観測確率と，従来手法・提案手法を用いた計算によって得られた推定観測確率の値の差」とし，ある事後条件 b に関する誤差を $error_b$，真の観測確率を p_{true_b}，推定観測確率を p_b とすると，次の式で表される．この誤差が小さいほど正確度が高いとする．

$$error_b = |p_{true\_b} - p_b|$$

研究課題 3 については，一度の学習において実行トレースの読み込みから最後のパラメータ更新までに要した時間を計算時間とし，評価する．

6.2.2 評価設定

従来手法の入力とする実行トレースは，小規模な例題では計算時点から過去 3,001 動作分（1,500 アクションセット），大規模な例題では 300,001 動作分（150,000 アクションセット）とする．自動倉庫管理システムの例題において，実行トレース長を変化させて従来手法の正確度に関する予備実験を行い，これらの実行トレース長を用いることで学習結果が収束するという結果が得られたためである．

実験を行うにあたり，大小 2 つの例題において次の 2 つの環境を用意した．

- 環境 1 ：観測可能な動作が決定的である環境
- 環境 2 ：観測可能な動作が非決定的である環境

環境 1 から環境 2 へ変化した場合，環境 2 から環境 1 へ変化した場合について，それぞれ実験を行った．小規模な例題では行った動作数が 5001 となった時点で，大規模な例題では行った動作数が 500,001 となった時点で環境を変化させた実行トレースを用意した．これらの実行トレースは，学習結果に応じたシステムの仕様変更を行わない場合に実行可能なものとなっている．

また，従来手法，提案手法共に，パラメータ $\mathbf{p}$ の初期値は 0.5，環境モデルに追加・削除する基準となる閾値 ζ は 0.1 とする．

6.2.3 パラメータの更新手法

勾配法におけるパラメータの更新については様々な手法が存在する．今回の実験では，単純に学習率 η を 0.5，0.1，0.05，0.01，0.005，0.001 と変化させることに加え，既存のパラメータ更新手法を用いた実験も行った．適用した既存のパラ

メータ更新手法は，Adam[17]，AdaDelta[29]，RMSProp[26]，AdaGrad[12] の 4 つである．これらの手法は，学習率を計算時に調整することで，誤差関数の素早い収束と振動の抑制を目指す手法である．AdaGrad では，各パラメータはそれぞれ異なる学習率を持ち，その学習率は計算をする度に更新される．ここで，急速な学習率の低下を防ぐために AdaGrad を改良したものが，RMSProp，AdaDelta，Adam である．各手法についての詳細は付録に記載する．

6.3 研究課題 1 ：学習の正確度

本節では，各手法を用いた学習の正確度について，比較評価する．今回の実験では，変化前後の環境では観測される規則が一部異なっており，観測されなくなった規則の不十分な学習の結果が従来手法と提案手法で大きく異なることがある．その際に得られる正確度は偶発的なものであり，比較が困難であるため，ここでは変化前の環境における学習結果に着目する．

6.3.1 環境 1 から環境 2 へ変化した場合

まず，環境 1 から環境 2 へ変化した場合の結果を示す．従来手法について，小規模な例題の学習結果を図 6.2 に，大規模な例題の学習結果を図 6.3 に示す．提案手法について，小規模な例題の学習結果を図 6.4，図 6.5 に，大規模な例題の学習結果を図 6.6，図 6.7 に示す．比較のため，図 6.4 から図 6.7 には従来手法の結果（学習率 0.5 の場合の結果）も載せている．縦軸は全事後条件における誤差の平均値であり，横軸は行った動作数である．また，GD は勾配降下法を用いた従来手法を表している．

図 6.2，図 6.3 より，従来手法ではパラメータの更新手法によって学習の正確度にはほとんど差がないことがわかる．従来手法では，誤差関数が収束するまで繰り返し計算を行うためである．したがって，以降は学習率 0.5 の場合の結果を従来手法の結果とし，提案手法との比較を行う．

図 6.4 から図 6.7 より，変化前の環境 1 における学習の正確度は，従来手法の方が優れている．提案手法では，学習率が 0.5 の場合に従来手法に近い正確度での学習が実現できていることがわかる．

しかし，この差は従来手法によって得られた十分に学習されていない結果が偶発的に正解に近くなったために得られたものである．小規模な例題の結果では，1 度しか観測されていないある 1 つの規則の学習結果により，図 6.4 に示されるような差が生じている． 1 度しか観測されていない規則が存在する場合，観測された事後条件の推定観測確率は，従来手法では 1.0，提案手法では初期値に近い値となり，学習結果に大きく差が生じてしまう． 1 度しか観測されていない規則を除いて誤差の平均値を求めると，その差は約 0.00005 であることから，全ての規則が十

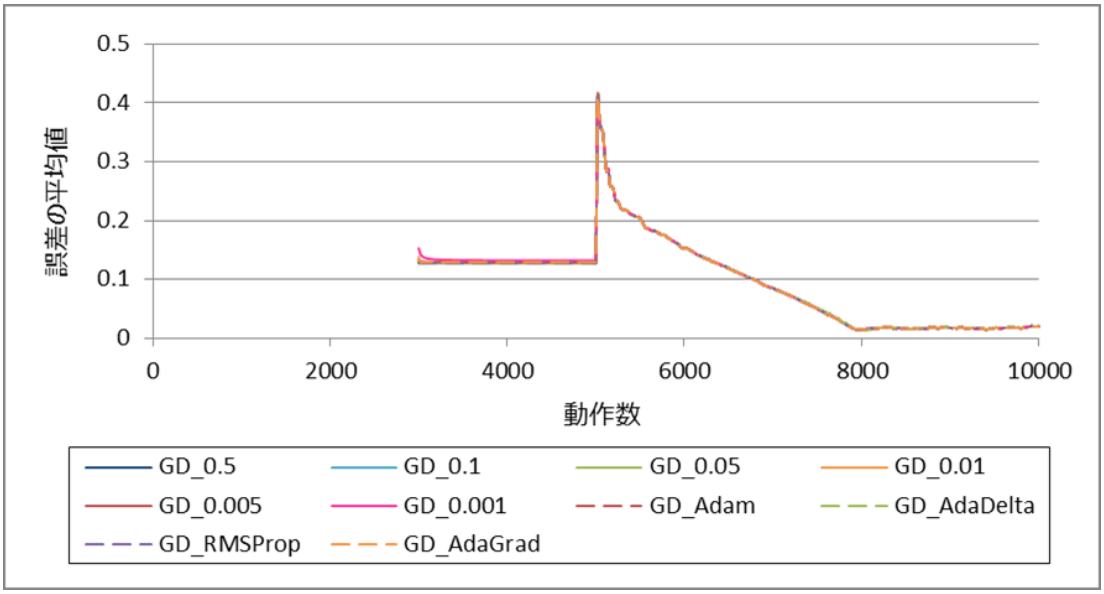

図 6.2: 従来手法の学習の正確度（小規模な例題，環境 1 →環境 2 ）

分に観測された場合は正確度の差は微小となり，どちらの手法を用いても同程度の正確度での学習が可能であることが推測できる．

6.3.2 環境 2 から環境 1 へ変化した場合

次に，環境 2 から環境 1 へ変化した場合の結果を示す．従来手法について，小規模な例題の学習結果を図 6.8 に，大規模な例題の学習結果を図 6.9 に示す．提案手法について，小規模な例題の学習結果を図 6.10，図 6.11 に，大規模な例題の学習結果を図 6.12，図 6.13 に示す．比較のため，図 6.10 から図 6.13 には，従来手法の結果（学習率 0.5 の場合の結果）も載せている．

図 6.8，図 6.9 より，環境 1 から環境 2 へ変化した場合の結果と同様に，従来手法ではパラメータの更新手法によって差はほとんど生じていないため，学習率 0.5 の場合の結果を従来手法の結果とし，提案手法との比較を行う．

図 6.10 から図 6.13 より，変化前の環境 2 における学習の正確度は，学習率が 0.001，0.005，0.01 の場合，既存の 4 つのパラメータ更新手法を用いた場合，従来手法で良いことがわかる．

提案手法では，学習率が大きくなるほど新しく得られた観測結果を学習結果に大きく反映するようになる．そのため，学習率を大きくした場合に，複数の事後条件が観測されるような環境下では，学習結果が不安定となり正確度が落ちてしまう場合がある．一方，学習率が小さい場合，新しく得られた観測結果は学習結果に小さく反映される．そのため，安定した学習が可能となり，このような結果が得られたと考えることができる．

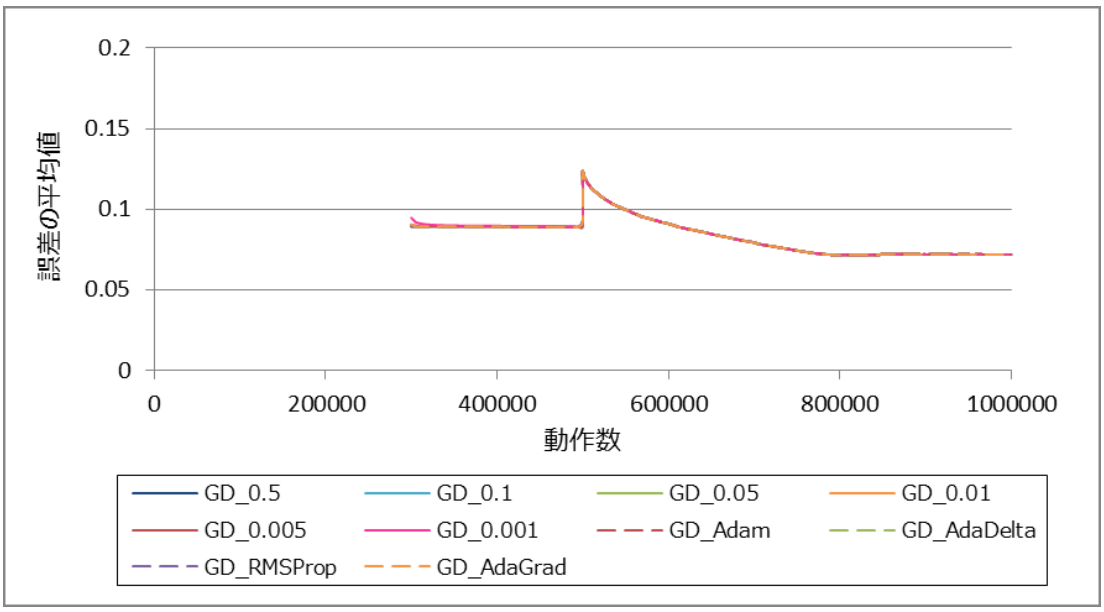

図 6.3: 従来手法の学習の正確度（大規模な例題，環境 1 →環境 2 ）

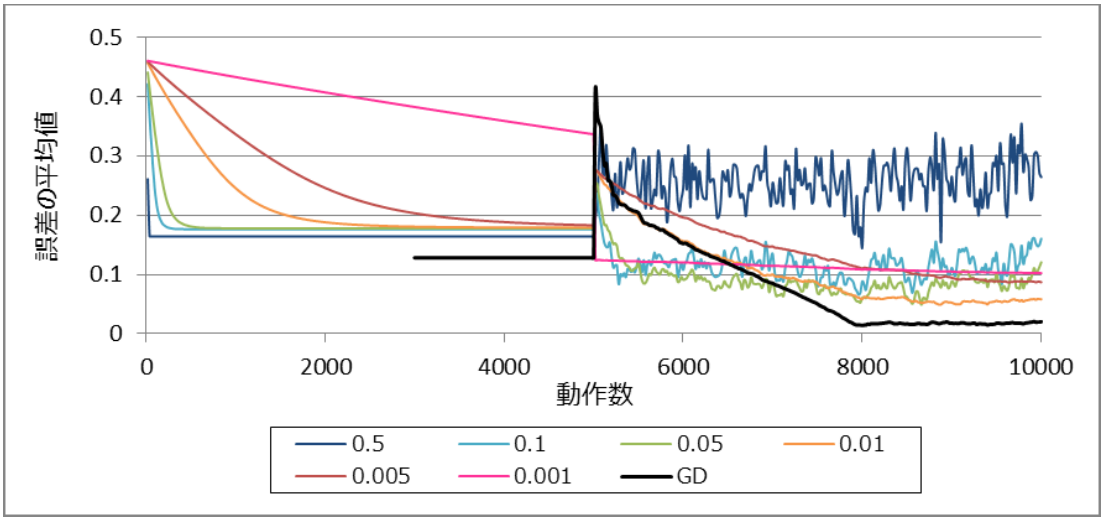

図 6.4: 一定の学習率を用いた差分学習の正確度（小規模な例題，環境 1 →環境 2 ）

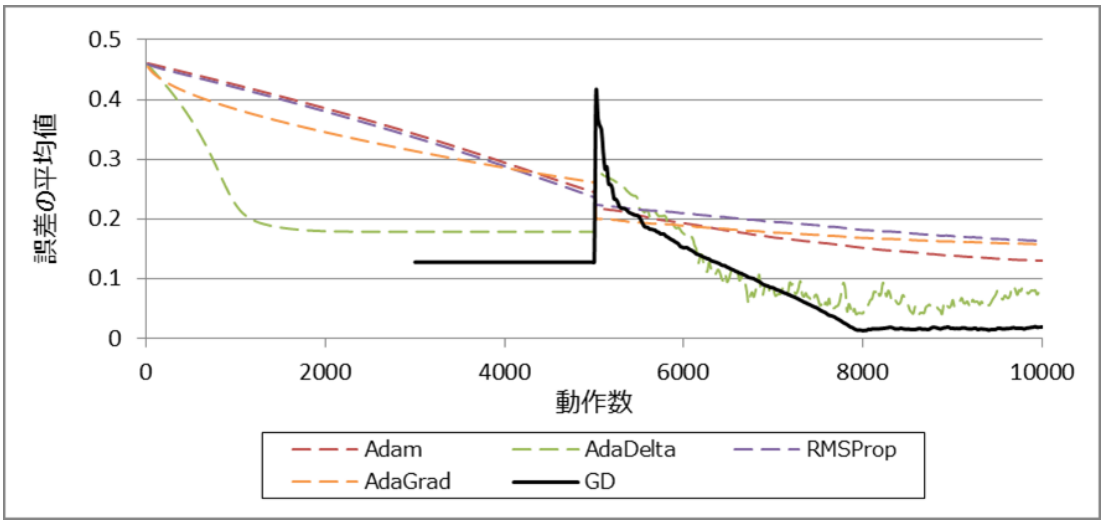

図 6.5: 既存手法を用いた差分学習の正確度（小規模な例題，環境 1 →環境 2 ）

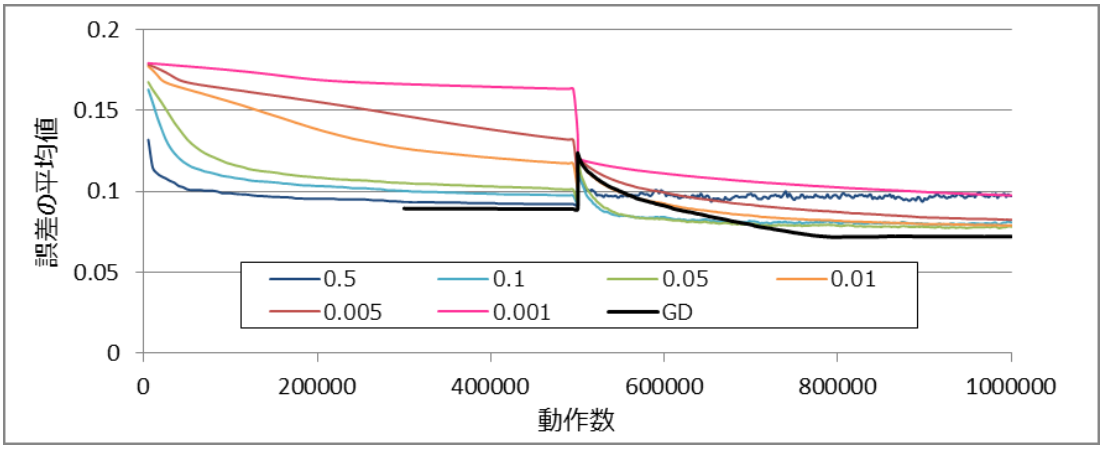

図 6.6: 一定の学習率を用いた差分学習の正確度（大規模な例題，環境 1 →環境 2 ）

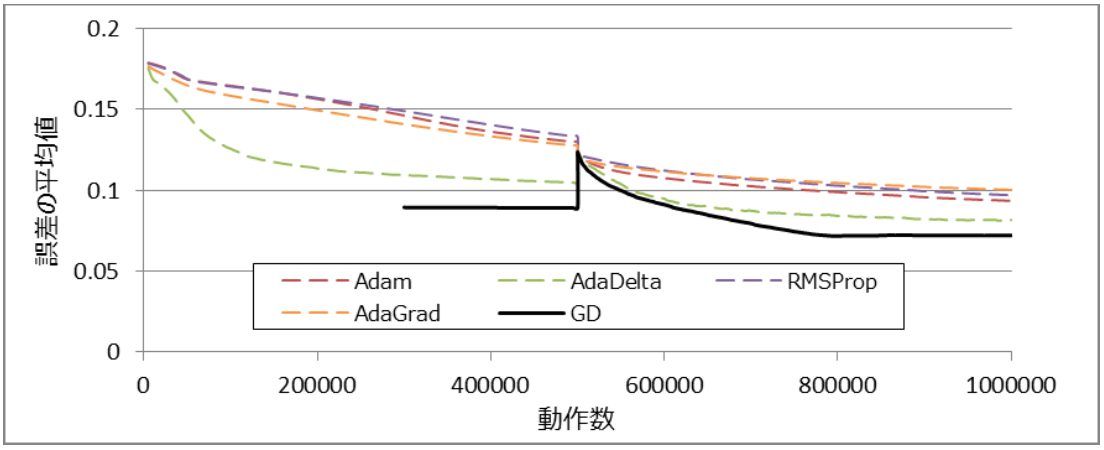

図 6.7: 既存手法を用いた差分学習の正確度（大規模な例題，環境 1 →環境 2 ）

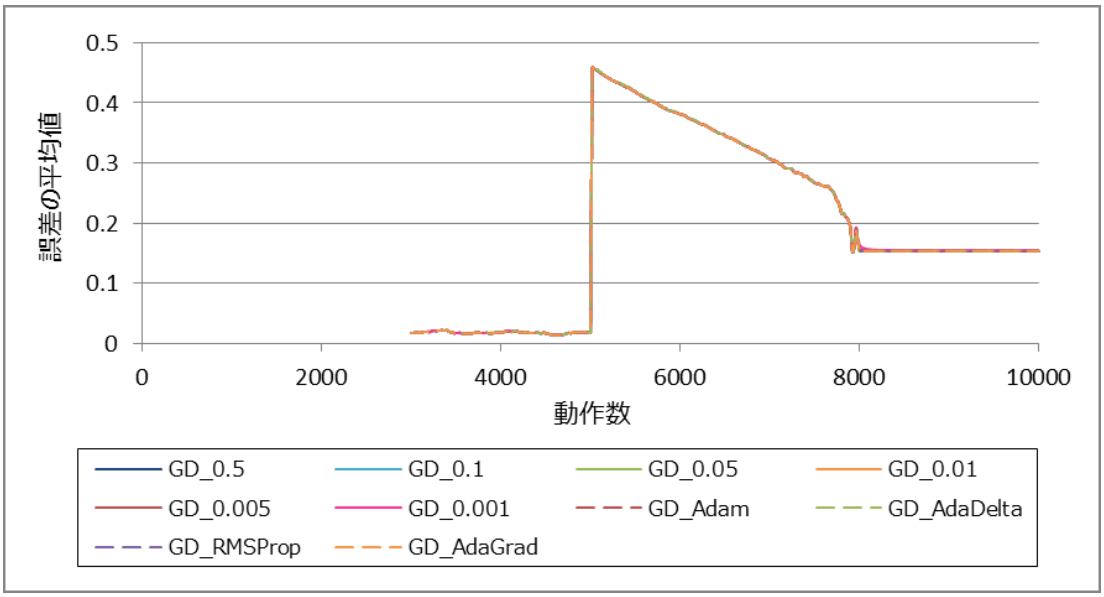

図 6.8: 従来手法の学習の正確度（小規模な例題，環境 2 →環境 1 ）

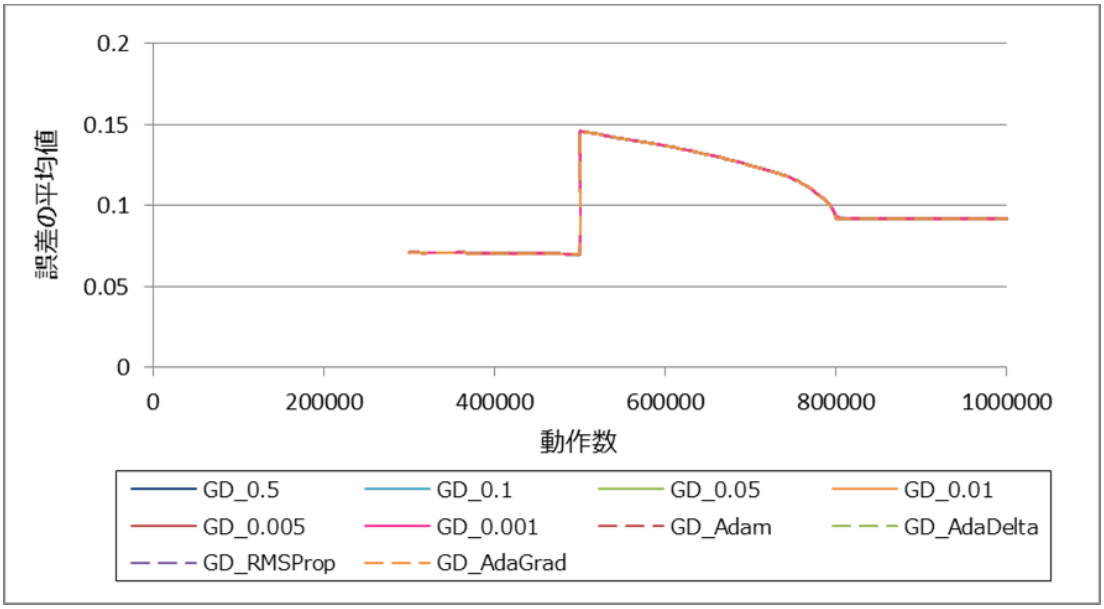

図 6.9: 従来手法の学習の正確度（大規模な例題，環境 2 →環境 1 ）

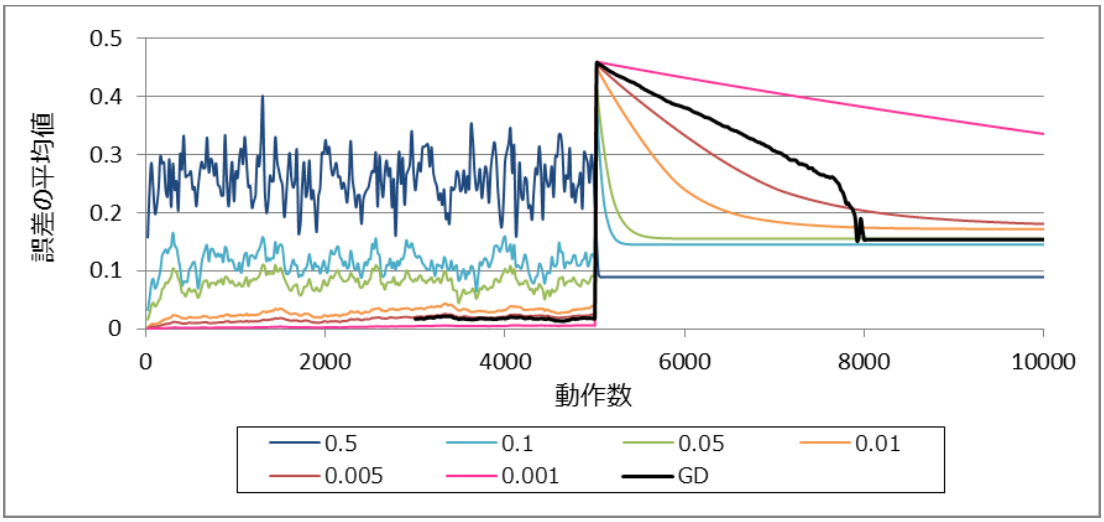

図 6.10: 一定の学習率を用いた差分学習の正確度（小規模な例題，環境 2 →環境 1 ）

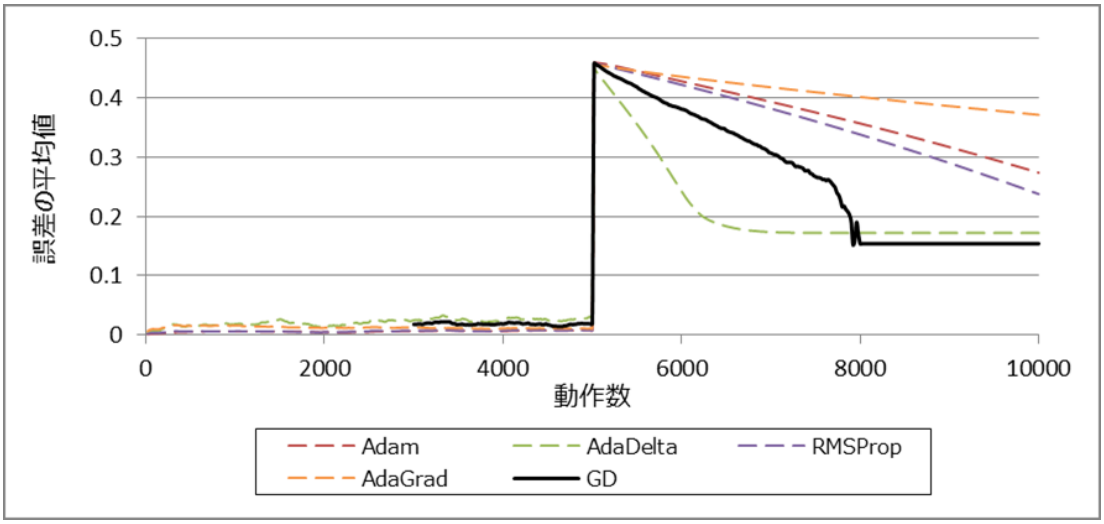

図 6.11: 既存手法を用いた差分学習の正確度（小規模な例題，環境 2 →環境 1 ）

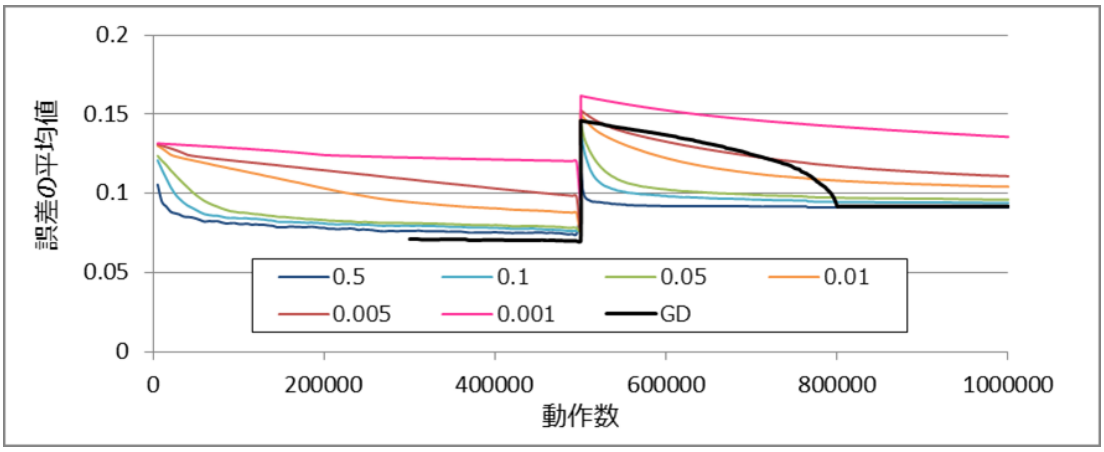

図 6.12: 一定の学習率を用いた差分学習の正確度（大規模な例題，環境 2 →環境 1 ）

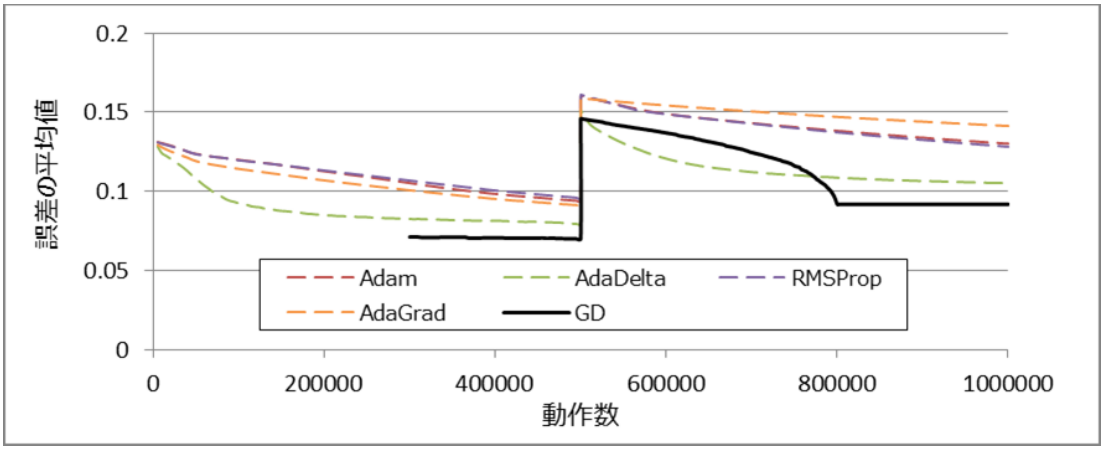

図 6.13: 既存手法を用いた差分学習の正確度（大規模な例題，環境 2 →環境 1 ）

6.4 研究課題２：正確度の収束性

本節では，環境が変化した際の正確度の収束性について，各手法の比較評価を行う．

6.4.1 収束性の比較

まず，6.3 節の変化後の環境における学習結果に着目する．図 6.4 から図 6.7 より，環境 1 から環境 2 へ変化した場合は，学習率が 0.1，0.05 のとき，例題の規模に関わらず，正確度が従来手法よりも素早く収束していることがわかる．また前節の図 6.10 から図 6.13 より，環境 2 から環境 1 へ変化した場合も同様の結果が得られている．

従来手法では，勾配の計算時に過去の環境における観測結果の影響を大きく受けてしまう．それに対し，提案手法では計算時点で得られた観測結果のみを考慮していることから，過去の環境における観測結果の影響を受けにくい．また，前節で述べたように，提案手法では学習率が大きいほど新しく得られた観測結果が学習結果に大きく反映されやすい．これらのことから，大きな学習率を用いた場合の提案手法の学習結果において良い結果が得られたと考えられる．

今回の実験では，環境を一度だけ変化させて実験を行っているが，実際には環境の変化は度々発生するものである．その際，従来手法では学習結果の収束に時間がかかるために，学習結果の収束前に新たな環境の変化が発生する場合が多く存在することが考えられる．そのような場合，変化前の環境における観測結果の影響により，実際の環境を正確に表現するモデルの学習が困難になってしまうことが予想される．度々変化する環境下で実際の環境を正確に表現するモデルを実行時に構築するためには，学習結果が素早く収束することが重要であると考えられる．

6.4.2 収束性によるシステムの実行への影響度

次に，収束性によるシステムの実行への影響を調査するため，以上の実験とは異なる環境の変化を与えて，学習結果の比較を行う．大小のそれぞれの例題において,「エリア間が商品で塞がれて通行不可能となる」という環境の変化にシステムが直面した場合を想定し，実行トレースを用意した．このような環境の変化に直面した場合，システムの実行に支障をきたさないよう，素早くこの変化を認識し，代替動作に切り替えることが求められる．そこで，環境の変化に直面してから代替動作に切り替えるまでに要する動作数をもとに，システムの実行への影響度を比較評価する．

小規模な例題では，図 2.1 においてエリア m とエリア w の間が通行不可能となった場合を想定し，実験を行った．このときシステムは,（1）ロボットがエリア m

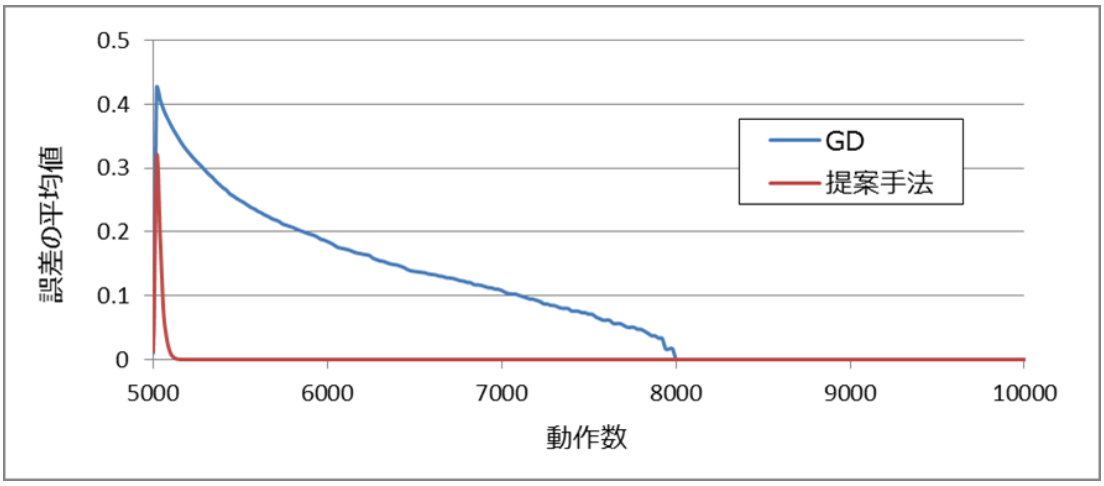

図 6.14: 一部の規則の学習結果（小規模な例題）

に到達した状態（arrive.m を受理した状態）で（２）エリア w 方向に移動するように指示をする（move.w を実行する）が,（３）通行ができないためにロボットは再びエリア m に到達する（arrive.m を受理する），という動作を繰り返し行う．システムの実行開始から 5,001 動作目（2,500 アクションセット目）で環境の変化に直面した場合の，変化した規則の学習結果を図 6.14 に示す．従来手法の結果は，学習率 0.1 の場合の結果である．

図 6.14 より，学習結果の収束性は従来手法と提案手法で大きく異なっていることがわかる．閾値 ζ が 0.1 の場合，次の変化前後の規則のように，arrive.w が観測されなくなったために制御器の更新が必要であると判断されるまでには，従来手法では環境の変化に直面してから 1,600〜1,620 動作を実行・受理した後，提案手法では 20〜40 動作を実行・受理した後であった．各動作の実行・受理に約 1 秒要するとすると，その差は約 1,600 秒（約 26 分）である．

$$\text{変化前} \quad < arrive.m, move.w, \{arrive.w, arrive.m, arrive.e\} >$$

$$\text{変化後} \quad < arrive.m, move.w, \{arrive.m, arrive.e\} >$$

大規模な例題では，図 6.1 においてエリア＜7，4＞とエリア＜8，4＞の間（上から 5 列目の左から 8 行目，9 行目のエリア間）が通行不可能となった場合を想定し，実験を行った．このときシステムは,（１）ロボットがエリア＜7，4＞に到達した状態（arrive.<7,4>を受理した状態）で（２）東方向に移動するように指示をする（move.e を実行する）が,（３）通行ができないためにロボットは再びエリア＜7，4＞に到達する（arrive.<7,4>を受理する），という動作を繰り返し行う．システムの実行開始から 500,001 動作目（250,000 アクションセット目）で環境の変化に直面した場合の，変化した規則の学習結果を図 6.15 に示す．図 6.15 より，小規模な例題と同様に，学習結果の収束性は従来手法と提案手法で大きく異なっていることがわかる．閾値 ζ が 0.1 の場合，制御器の更新が必要であると判断

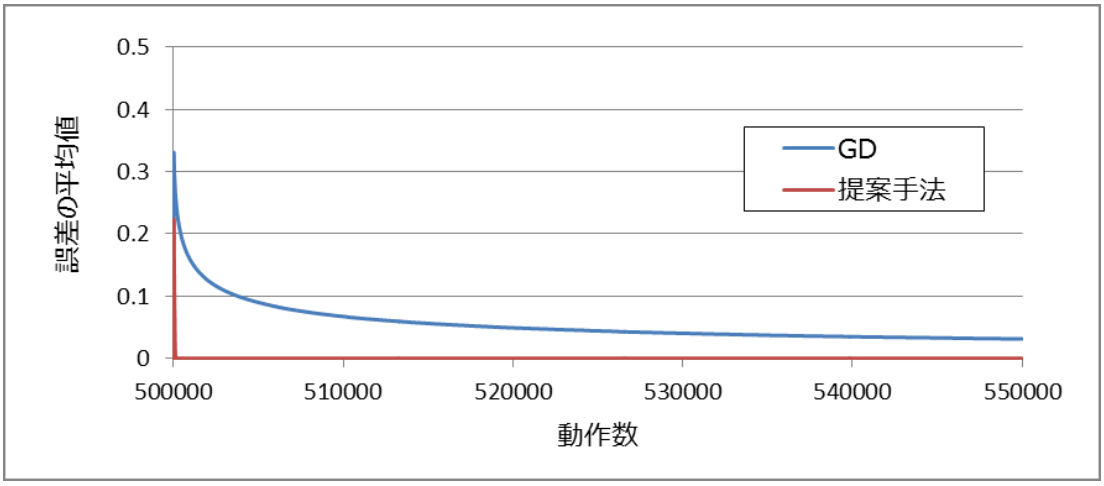

図 6.15: 一部の規則の学習結果（大規模な例題）

されたのは，従来手法では環境の変化に直面してから 30,600〜30,620 動作を実行・受理した後，提案手法では 80〜100 動作を実行・受理した後であった．各動作の実行・受理に約 1 秒要するとすると，その差は約 30,500 秒（約 8.5 時間）である．

以上の結果から，従来手法と提案手法では収束性は大きく異なっており，特に従来手法では収束性の悪さ故にシステムの実行に大きく影響を与えてしまう可能性があることがわかる．小規模な例題のように，約 26 分間システムの実行が滞ってしまうのは現実的ではなく，大規模な例題のように，約 8.5 時間システムの実行が滞ってしまうのは大きな問題となる．学習結果の収束性の良さは，システムの実行に支障をきたさないためにも重要であると言える．

6.5 研究課題３：計算時間

本節では，一度の学習に要する計算時間について評価する．

環境 1 から環境 2 へ変化した場合の計算時間について，小規模な例題に関する結果を図 6.16，図 6.17 に，大規模な例題に関する結果を図 6.18，図 6.19 に示す．縦軸は計算時間，横軸は行った動作数である．

図 6.16，図 6.18 より，従来手法ではパラメータの更新手法により計算時間が異なっている．従来手法では誤差関数が収束するまで計算を繰り返すが，その収束のしかたによって計算回数が異なるためである．Adam，AdaDelta，RMSProp，AdaGrad は誤差関数を素早く収束させるためのアルゴリズムであるが，計算時間はある一定の学習率を用いた場合と同程度，もしくはそれ以上となっている．したがって，これらの既存のパラメータ更新手法は，変化する環境下で用いるにはあまり適していないことがわかる．図 6.17，図 6.19 より，提案手法ではパラメータの更新手法により計算時間にほとんど差は存在していない．どのパラメータの更新手法を用いても計算回数が同じためである．

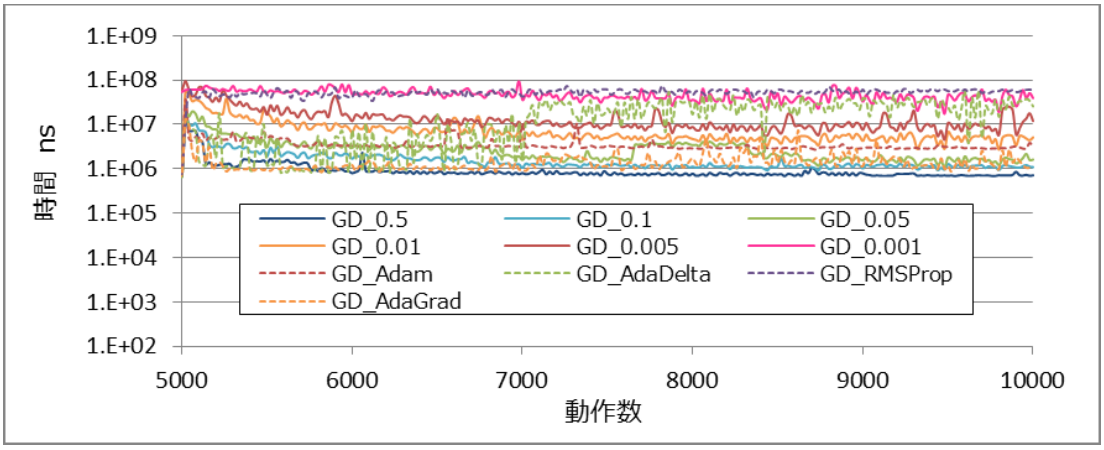

図 6.16: 従来手法の計算時間（小規模な例題）

図 6.16，図 6.17 を比較すると，小規模な例題における計算時間は，従来手法では 1〜100 ミリ秒，提案手法では 0.001 ミリ秒であり，従来手法の計算時間は提案手法の 1,000〜10,000 倍となっている．また図 6.18，図 6.19 を比較すると，大規模な例題における計算時間は，従来手法では約 10,000 ミリ秒，提案手法では 0.01〜0.1 ミリ秒であり，従来手法の計算時間は提案手法の 100,000〜1,000,000 倍となっている．これらの結果から，例題の規模が大きくなることで，計算時間は従来手法では 1,000〜10,000 倍，提案手法では 10 倍となっていることがわかる．また図 6.16 から図 6.19 の結果において，計算時間の平均増加量は，従来手法では約 12,000 ミリ秒，提案手法では約 0.12 ミリ秒となっており，従来手法では計算時間が大幅に増加していることがわかる．

一度のパラメータ更新に要する計算量は，従来手法では $O(n)$，提案手法では $O(1)$ である．従来手法では規則数の増加によりパラメータの更新回数も増加するため，システムの規模が大きくなるにつれて計算時間が大幅に増加し，提案手法では従来手法に比べて計算時間の増加量は少なくなることが予想される．前述の結果より，実際に計算時間の増加量は従来手法において大きくなっている．提案手法は従来手法と比較して計算時間の増加量は小さくなっており，システムの規模がさらに拡大した場合でも現実的な時間での学習が実現可能であると考えられる．

また従来手法では，大規模な例題における計算時間が 10〜20 秒程度となっている．新しいアクションセットが得られる度に学習を行うことを考えると，一度の学習に 10 秒以上要するのは現実的ではない．自動倉庫管理システムにおいても，計算時間の増加はロボットの作業効率の低下につながってしまう．一方，提案手法では一度の学習は 1 ミリ秒以下で行うことが可能である．これはロボットの作業効率にほとんど影響を与えることなく，無視することができる値である．

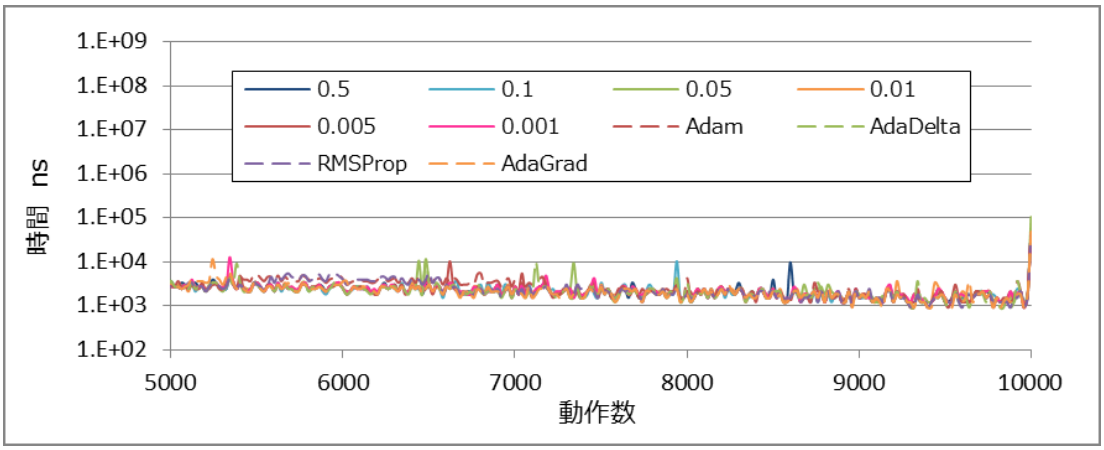

図 6.17: 提案手法の計算時間（小規模な例題）

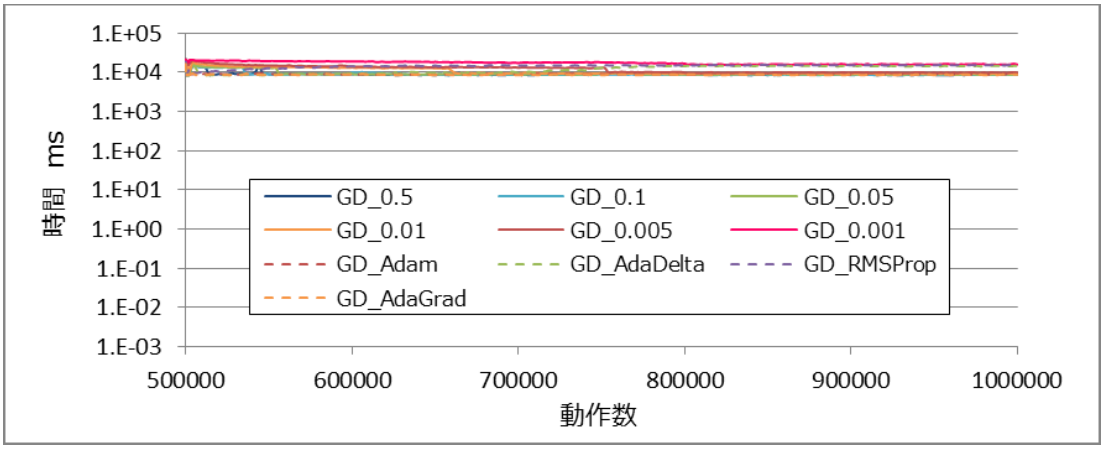

図 6.18: 従来手法の計算時間（大規模な例題）

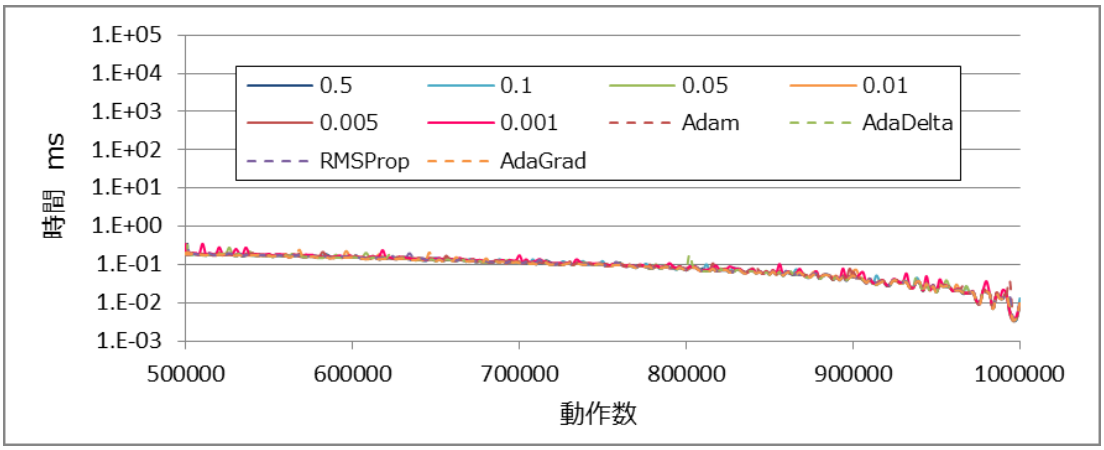

図 6.19: 提案手法の計算時間（大規模な例題）

6.6 評価結果のまとめ

3 つの研究課題について実験を行い，それぞれ次のような結果が得られた．

研究課題 1 従来手法と提案手法の学習の正確度は同程度である．

研究課題 2 適した学習率を用いた場合，従来手法よりも提案手法の方が正確度の収束性が良い．

研究課題 3 提案手法を用いることで，学習に要する計算時間は大幅に削減可能である．

提案手法の学習結果は学習率により異なっている．今回の実験の例題においては 0.05 から 0.1 の学習率が適していること，また既存のパラメータ更新手法を用いる場合は AdaDelta が適していることがわかった．

以上の結果から，変化する環境において正確かつ短時間での学習を実現するにあたり，本手法は適した学習率を用いた場合に有用であることがわかる．

6.7 本手法の有用性と限界

本手法は，変化する環境において正確かつ短時間での学習を実現するにあたり，適した学習率を用いた場合に有用である．しかしながら，適した学習率はシステムの規模や仕様によって異なることが予想され，最適なものを事前に特定することは困難である．したがって，実行時により適した学習率を特定することが求められる．実験で用いた既存のパラメータ更新手法は学習結果に応じて学習率を変化させる手法であるが，これらの手法は変化する環境下で用いるには収束性の点であまり適していなかった．そのような環境において有用なパラメータ更新手法とあわせて本手法を用いることができれば，適した学習率を事前に特定することなく，正確かつ短時間で収束性も良い学習が実現可能であると考えられる．

環境モデルの構成要素である規則は，予め用意しており，規則中に存在しないアクションセットを観測した場合は学習されない．したがって，観測し得るアクションセットはシステムの開発時に洗い出し，規則として用意する必要がある．またアクションセットは，制御可能な動作，観測可能な動作を交互に実行・受理することを想定し，制御可能な動作とその前後の観測可能な動作の組としている．しかしながら，制御可能な動作を複数回実行するようなシステム，ある 1 つの制御可能な動作に対して複数の観測可能な動作を受理するようなシステムも存在すると考えられる．

また本手法により学習可能な規則は，実行された制御可能な動作に関する規則のみであり，実行されない制御可能な動作に関する規則は学習不可能である．本研究は，要求の充足が保証された制御器に基づいて動作するシステムを対象にしているため，要求が充足されなくなる可能性のある動作は実行することができな

い．したがって，環境の変化により実行しなくなってしまった動作を再び実行することができない可能性がある．例えば，自動倉庫管理システムにおいて，一度通行不可能であると判断されたエリア間は，以降，通行するという選択をされなくなってしまう．再度通行可能となった場合でも，通行するという選択をしないために，その変化を学習することはできない．

以上の実験結果は，調整するパラメータの初期値や，従来手法において用いる実行トレース長によって変化する可能性がある．既存のパラメータ更新手法を用いた場合についても，設定値を変更することで結果が変化する可能性がある．

第7章　おわりに

7.1　まとめ

自己適応システムは，環境の変化に対して，要求を充足し続けるよう，システム自身で実行時に振る舞いを変更するシステムである．振る舞いを決定する際に，離散制御器合成技術を用いることで，与えられた環境下での要求の充足が保証された振る舞い仕様を自動生成することが可能である．しかしながら，システムの実際の実行環境とシステムが想定している実行環境に差異が存在する場合，正しい振る舞い仕様の生成ができなくなってしまう．また，システムの実行環境は不確実性を持つため，事前に実行環境を想定することは困難である．従って，システムが想定している実行環境である環境モデルと，実際の実行環境との一貫性を維持するためには，実行時に実環境を正確に表現する環境モデルを学習することが求められる．

従来の環境モデルの学習手法は，システムの非実行時に用いることを想定しているため，実行時に用いるには学習に要する時間が課題となる．そこで本研究では，実行時の正確かつ短時間での学習を実現するため，実行時に得られるデータから効率よく環境モデルを差分学習する手法を提案した．確率的勾配降下法を応用することで，差分のみの学習を実現し，学習に要する時間を削減した．評価では，環境モデルの学習の正確度とその収束性，計算時間について，従来手法との比較を行い，次の３点について確認した．

1. 従来手法と提案手法の学習の正確度は同程度であること．
2. 提案手法では適した学習率を用いることで従来手法よりも正確度の収束性の良い学習が可能であること．
3. 提案手法を用いることで一度の学習に要する計算時間は大幅に削減可能であること．

変化する環境下で実際の環境を正確に表現するモデルを実行時に構築するためには，短い時間で素早く環境の変化を学習することが重要であり，本手法は有用であるといえる．

7.2 今後の課題

本手法を用いるにあたり，6.7 節でも述べたように，学習率の設定が課題となる．事前に適した学習率を特定せずに，実行時に学習率を適応させる手法，特に変化する環境下において有用な手法を考案することで，より正確かつ収束性の良い実行時学習が実現可能となる．

また，同じく 6.7 節で述べたように，本手法では，実行されない制御可能な動作に関する規則は学習不能となってしまうという課題が存在する．このような規則を学習するためには，実行されなくなった制御可能な動作を実行する必要がある．しかしながら，そのような動作の実行は要求違反につながる可能性があり，要求充足の保証を扱う本研究では実行することができない．そこで，要求分析の手法と組み合わせることで，一時的に要求緩和を行い，緩和された要求の充足を保証した中で実行されなくなった制御可能な動作を試行することができれば，学習不能となった規則の学習も可能となると考えられる．

本手法では，事前に入力された規則群についての学習を行うが，実行中に想定していなかった規則を観測した場合への対処も今後の課題である．想定外の規則も新しい規則として学習可能にすることで，開発時の規則の想定にかかる負担を軽減することができる．

評価では，自動倉庫管理システムを例題として実験を行ったが，より大規模なシステムや実在するシステムへ本手法を適用し，拡張性や実用性に関する評価にも取り組むべきであると考えられる．またその際に，4 章の図 4.1 に示したような自己適応システムを実現し，実際にシステムを稼動させたときの，学習によるシステムへの影響度についても評価すべきである．

付 録A　パラメータ更新手法の設定

A.1　AdaGradのアルゴリズム

AdaGradでは，各パラメータはそれぞれ異なる学習率を持ち，その学習率は計算をする度に更新される．パラメータθ_iに関して，g_iを誤差関数の勾配，ηとeを任意の定数，h_iを各パラメータが持つ変数（初期値＝0）とすると，パラメータの更新式は次のように表される．今回の実験では，$\eta = 0.01$，$e = 10^{-8}$として計算を行った．

$$h_i = h_i + g_i^2 \tag{A.1}$$

$$\theta_i = \theta_i - \frac{\eta g_i}{\sqrt{h_i + e}} \tag{A.2}$$

A.2　RMSPropのアルゴリズム

RMSPropでは，各パラメータは次の式のように更新される．θ_iはパラメータ，g_iは誤差関数の勾配，α，β，eは任意の定数，h_iは各パラメータが持つ変数（初期値＝0）である．今回の実験では，$\alpha = 0.001$，$\beta = 0.9$，$e = 10^{-8}$として計算を行った．

$$h_i = \beta h_i + (1 - \beta) g_i^2 \tag{A.3}$$

$$\theta_i = \theta_i - \frac{\alpha g_i}{\sqrt{h_i + e}} \tag{A.4}$$

A.3　AdaDeltaのアルゴリズム

AdaDeltaでは，各パラメータは次の式のように更新される．θ_iはパラメータ，g_iは誤差関数の勾配，β，eは任意の定数，r_i，s_i，x_iは各パラメータが持つ変数（初期値＝0）である．今回の実験では，$\alpha = 0.95$，$e = 10^{-6}$として計算を行った．

$$r_i = \beta r_i + (1 - \beta) g_i^2 \tag{A.5}$$

$$x_i = \frac{\sqrt{s_i + e}}{\sqrt{r_i + e}} g_i \tag{A.6}$$

$$s_i = \beta s_i + (1 - \beta) x_i^2 \tag{A.7}$$

$$\theta_i = \theta_i - x_i \tag{A.8}$$

A.4 Adam のアルゴリズム

Adam では，各パラメータは次の式のように更新される．θ_i はパラメータ，g_i は誤差関数の勾配，α，β_1，β_2，e は任意の定数，m_i，v_i，$\hat{m}_i$，$\hat{v}_i$ は各パラメータが持つ変数（初期値＝0），t は計算回数である．今回の実験では，$\alpha = 0.001$，$\beta_1 = 0.9$，$\beta = 0.999$，$e = 10^{-8}$ として計算を行った．

$$m_i = \beta_1 ri + (1 - \beta_1) g_i \tag{A.9}$$

$$v_i = \beta_2 x_i + (1 - \beta_2) g_i^2 \tag{A.10}$$

$$\hat{m}_i = \frac{m_i}{1 - \beta_1^t} \tag{A.11}$$

$$\hat{v}_i = \frac{v_i}{1 - \beta_2^t} \tag{A.12}$$

$$\theta_i = \theta_i - \frac{\alpha \hat{m}_i}{\sqrt{\hat{v}_i + e}} \tag{A.13}$$

謝辞

本論文の執筆にあたり，指導教員として様々なご指導を賜りました，早稲田大学大学院 基幹理工学研究科 情報理工・情報通信専攻の深澤良彰教授に深謝を申し上げます．また，ご指導，ご助言いただいた国立情報学研究所の本位田真一教授，本研究及び論文の細部にわたりご指導いただいた鄭顕志准教授，研究をするにあたり多くのご助言を賜りました清水遼先輩，相澤和也先輩にも深謝申し上げます．

また，大学院生活を通し，ともに研究に励み，様々なご協力をいただいた早稲田大学大学院 基幹理工学研究科 情報理工・情報通信専攻 深澤研究室，鷲崎研究室の皆様．東京大学大学院 情報理工学系研究科 コンピュータ科学専攻及び創造情報専攻 本位田研究室の皆様，電気通信大学大学院 情報システム学研究科 大須賀・清研究室＆田原研究室の皆様にも感謝申し上げます．

参考文献

[1] Eugene Asarin, Oded Maler, and Amir Pnueli. Symbolic controller synthesis for discrete and timed systems. In *International Hybrid Systems Workshop*, pp. 1–20. Springer, 1994.

[2] Luciano Baresi and Carlo Ghezzi. The disappearing boundary between development-time and run-time. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pp. 17–22. ACM, 2010.

[3] Léon Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, Vol. 17, No. 9, p. 142, 1998.

[4] Olivier Bousquet and Léon Bottou. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pp. 161–168, 2008.

[5] Victor Braberman, Nicolas D'Ippolito, Jeff Kramer, Daniel Sykes, and Sebastian Uchitel. Morph: A reference architecture for configuration and behaviour self-adaptation. In *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*, pp. 9–16. ACM, 2015.

[6] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pp. 1–32. Springer, 2013.

[7] Zuohua Ding, Yuan Zhou, and MengChu Zhou. Modeling self-adaptive software systems with learning petri nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. 46, No. 4, pp. 483–498, 2016.

[8] Nicolás Roque D'Ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. Synthesis of live behaviour models. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 77–86. ACM, 2010.

[9] Nicolas D'Ippolito, Víctor Braberman, Jeff Kramer, Jeff Magee, Daniel Sykes, and Sebastian Uchitel. Hope for the best, prepare for the worst: multi-tier control for adaptive systems. In *Proceedings of the 36th International Conference on Software Engineering*, pp. 688–699. ACM, 2014.

[10] Nicolas D'Ippolito, Victor Braberman, Daniel Sykes, and Sebastian Uchitel. Robust degradation and enhancement of robot mission behaviour in unpredictable environments. In *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*, pp. 26–33. ACM, 2015.

[11] Nicolás D'Ippolito, Dario Fischbein, Marsha Chechik, and Sebastián Uchitel. Mtsa: The modal transition system analyser. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 475–476. IEEE Computer Society, 2008.

[12] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, Vol. 12, No. Jul, pp. 2121–2159, 2011.

[13] Dirk Fahland and Wil MP van der Aalst. Repairing process models to reflect reality. In *International Conference on Business Process Management*, pp. 229–245. Springer, 2012.

[14] Carlo Ghezzi, Leandro Sales Pinto, Paola Spoletini, and Giordano Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *Proceedings of the 2013 International Conference on Software Engineering*, pp. 33–42. IEEE Press, 2013.

[15] Dimitra Giannakopoulou and Jeff Magee. Fluent model checking for event-based systems. In *ACM SIGSOFT Software Engineering Notes*, Vol. 28, pp. 257–266. ACM, 2003.

[16] Julio E Godoy, Ioannis Karamouzas, Stephen J Guy, and Maria Gini. Adaptive learning for multi-agent navigation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1577–1585. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[17] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference for Learning Representations, arXiv:1412.6980*, 2015.

[18] Jeff Magee and Jeff Kramer. *State models and java programs*. wiley, 1999.

[19] David Martínez, Tony Ribeiro, Katsumi Inoue, Guillem Alenya, and Carme Torras. Learning probabilistic action models from interpretation transitions. 2015.

[20] Jacob Menashe and Peter Stone. Monte carlo hierarchical model learning. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pp. 771–779. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[21] Ali Yadavar Nikravesh, Samuel A Ajila, and Chung-Horng Lung. Towards an autonomic auto-scaling prediction system for cloud resource provisioning. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 35–45. IEEE, 2015.

[22] Nir Piterman, Amir Pnueli, YanivSa'ar. Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pp. 364–380. Springer, 2006.

[23] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, Vol. 4, No. 2, p. 14, 2009.

[24] Amir Molzam Sharifloo, Andreas Metzger, Clément Quinton, Luciano Baresi, and Klaus Pohl. Learning and evolution in dynamic software product lines. In *Proceedings of the 11th International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pp. 158–164. ACM, 2016.

[25] Daniel Sykes, Domenico Corapi, Jeff Magee, Jeff Kramer, Alessandra Russo, and Katsumi Inoue. Learning revised models for planning in adaptive systems. In *Proceedings of the 2013 International Conference on Software Engineering*, pp. 63–71. IEEE Press, 2013.

[26] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop. *COURSERA: Neural networks for machine learning*, 2012.

[27] Eric Yuan, Naeem Esfahani, and Sam Malek. Automated mining of software component interactions for self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 27–36. ACM, 2014.

[28] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM transactions on Software Engineering and Methodology (TOSEM)*, Vol. 6, No. 1, pp. 1–30, 1997.

[29] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[30] 海野裕也, 岡野原大輔, 得居誠也, 徳永拓之. オンライン機械学習. pp. 28–30. 講談社, 2015.

[31] 相澤和也. 実行時要求緩和のためのゲーム理論を応用した環境モデル分析手法に関する研究. 2016.

業績

1. Moeka Tanabe, Kenji Tei, Yoshiaki Fukazawa, and Shinichi Honiden. Learning environment model at runtime for self-adaptive systems. In Proceedings of the 32nd ACM Symposium on Applied Computing (SAC '17). April 3-6, 2017, Marrakesh, Morocco, 7 pages.

2. 田邉萌香, 鄭顕志, 深澤良彰, 本位田真一, "自己適応システムのための実行時環境モデル学習" , 合同エージェントワークショップ＆シンポジウム 2016 (JAWS2016), September 15-16, 2016, pp. 278-285. 優秀論文賞 受賞.