論文

深層強化学習のパラメータ最適化手法の提案

Proposition of A New Searching Method for Deep Reinforcement Learning

今福　啓[*1]
Kei Imafuku

Email: imafuku@dokkyo.ac.jp

本論文では、深層強化学習で必要となるパラメータ探索を行う新しい手法を提案する。通常、深層強化学習は勾配法によるパラメータ学習が行うが、膨大な乗算と加算を相当数くり返すため、GPU なしでの計算時間は非常に長い時間を要する。提案手法はパラメータを直接探索するため計算量を減らすことができ、CPU のみを搭載するコンピュータでも実用的な計算時間で深層強化学習を使用できる。この手法では行列計算で探索する解軌道を決定するため、固有値を変更することでその軌道を変更できる。また提案手法では 1 回の探索において大域的探索と局所探索を実行するため、効率的に望ましい解を得ることができる。提案手法を用いたシミュレーション結果から、その有効性を示す。

In this paper, we propose a new searching method for deep reinforcement learning (DRL). In general, DRL calculates the parameters by using the gradient method, however the method requires a lot of multiplication and addition, therefore the calculation time becomes extremely long without GPU. A proposed method searches the parameters directly, so it reduces the calculation time and we can use DRL on the computer that has only CPU. In the proposed method, the trajectory of the solution is calculated by the matrices, and we can get the various trajectory by changing the eigenvalues of the matrices. In addition, since this method executes the global and the local search when searching the optimal solution in one time, the proposed method can search the desirable solution effectively. Some simulation results reveal the effectiveness of the proposed method.

[*]1：獨協大学経済学部経営学科

1. はじめに

人工知能が実社会のさまざまな場面で活用されている。人工知能とよばれる手法に含まれるのは主に機械学習と強化学習であるが、本研究では強化学習に着目し、新たな手法を提案する。

人が扱うべき複雑な問題において、コンピュータが強化学習を用いて望ましい行動を学習できるようになったのは、多層ニューラルネットのような深層構造を導入し、安定した学習を実現したDQN（Deep Q-Network）の影響が大きい[2]。その後、行動の価値を最大化するように学習する手法としてDouble DQN[3]、A2C（Advanced Actor-Critic）[4]、政策を最適化するように学習する手法としてTRPO（Trust Region Optimization）[5]、PPO（Proximal Policy Optimization）[6]と、多くの手法が提案されている。いずれの手法もニューロンの重みを更新する際、出力結果に含まれる誤差を小さくする方向にパラメータを更新する、勾配を用いた計算が必要となる。その際、パラメータの値が急激に変わり学習が不安定とならないよう学習率を非常に小さい値として、多くの更新回数が必要となる。結果として計算回数が膨大なものとなり、特に乗算と加算をくり返すことから、実用的な時間で計算を終えるには数千のコアを持ち、並列に大量の加算と乗算を計算できるGPU（NVIDIA社製のもの[1]）を用いることが不可欠となっている。

また、深層強化学習で得られる解は、常に1つの最良解に収束するわけではない。結果が局所解となって望ましい結果に至らないことがあり、何度も学習をくり返す中で、良い解が得られたならばそれを使用する。このように、強化学習では複数回の学習を行うことは一般的である。

現在、プログラミング言語Pythonを利用することで強化学習をきわめて容易に扱うことができる[1]が、あらゆるコンピュータにGPUが搭載されているわけではない。手持ちのコンピュータを利用し、実用的な計算時間の範囲で簡単に実行できる手法の構築も必要である。

GPUに頼らないアルゴリズムの一つは、進化計算により直接的に望ましいパラメータを見つける手法である。OpenAI[7]は、進化計算によりパラメータ探索を行うことで、勾配法を用いるTRPOと比較して、より速く望ましい結果が得られることを示している。このように、勾配法を使わず直接的に必要なパラメータを獲得する手法にも、深層強化学習をより効率的に進められる可能性がある。

そこで本研究では、パラメータの探索を行う新たな手法を提案する。提案手法では行列計算により暫定的な最良解を更新するが、固有値をさまざまな値に設定して探索する軌道を変更するため、大域的探索から始まり最終的に局所探索となるよう均衡のとれた探索を実行することが可能となり探索の効率化が期待できる。そして提案手法を用いたシミュレーションを行い、手法の有効性を示す。

2. 提案手法

2.1 OpenAIの手法

OpenAIが提案する手法[7]では、1回のイテレーションで行動を決定するために用いる n 次元の値から構成される重みに平均0、標準偏差 σ の異なる正規乱数を加えた新たな重みを複数作成し、それを用いた深層学習器により与えられた環境で行動する。そして得られた報酬の合計値を標準化し、その値を使って加重平均により新たな重みを作成する。その時点で得られている最良の重み（以下、これを暫定解という）を $W(k) \in R^n$、重みに加えるノイズを $\Delta W_1(k),\cdots,\Delta W_m(k)$、新たな重み $W(k)+\Delta W_1(k),\cdots,W(k)+\Delta W_m(k)$ を使って行動して得られた報酬の合計を $r_1,\cdots,r_m$ としたとき、新たな重みは次式により計算される。これを指定したイテレーション数だけくり返す。なお、k は1回のイテレーション内で計算をくり返したステップ数である。

$$\Delta W_i(k)=\sigma\begin{bmatrix}\mathrm{rand}n_1(k)\\ \vdots \\ \mathrm{rand}n_n(k)\end{bmatrix} \tag{1}$$

$$W(k+1)=W(k)+\frac{\alpha}{m\sigma}\left(\frac{r_1(k)-\bar{r}(k)}{\sigma_r(k)}\Delta W_1(k)+\cdots+\frac{r_m(k)-\bar{r}(k)}{\sigma_r(k)}\Delta W_m(k)\right) \tag{2}$$

$r_i(k)(i=1,\cdots,m)$:
重み$W(k)+\Delta W_m(k)$で得られた報酬の合計
$\bar{r}(k)$: $r_1(k),\cdots,r_m(k)$の平均
$\sigma_r(k)$: $r_1(k),\cdots,r_m(k)$の標準偏差
α: 学習率
$\mathrm{rand}n_i(k)\ (i=1,\cdots,n)$:
平均0、標準偏差1の正規乱数

この手法では、元の重みに $\Delta W_i(k)\ (i=1,\cdots,m)$

1 GPUを利用した強化学習用のPythonモジュールでは、CUDAというNVIDIA社のGPUを利用するツールキットを内部で利用する。CUDAは他社のGPUでは利用が困難であるため、強化学習や機械学習の分野では、NVIDIAのGPUを使うことが一般的になっている。

を加えることで得られた報酬 $r_i(k)$ が全体の平均 $\bar{r}(k)$ から離れている場合は $(r_i(k) - \bar{r}(k))/\sigma_r(k)$ の絶対値が大きな値となり、元の重みが大幅に変更される。報酬が平均に近い場合には、更新の際に与える影響が小さくなる。

しかし、この手法で重みを更新することは必ずしも良い結果につながるとはいえない。得られた報酬の大きさに応じて、元の重みにノイズを加算することが良い結果を生じるとは断言できない。

本論文では提案手法を用いたシミュレーションでビデオゲームの強化学習を扱うが、このような環境で望ましい解を得る際に困難な点は、関数の最適化のように最良解が決まっており、それを見つければよい、とはならない点である。本研究で扱う強化学習では、最適化の目的が「ビデオゲームを出来るだけ高得点が得られるように行動する」といった形で与えられる。これを実現するための方法は無数に存在し、唯一の答えがあるわけではない。またビデオゲームでは登場するキャラクターの行動が常に一定ではなく、常に特定の操作だけを行ったとしても高い得点は得られない。そのためOpenAIの手法で行われる、暫定解を得た際に局所探索を行って解の精度を高める、といった手法は必ずしも有効ではないと考えられる。

2.2 提案手法

本論文では1回のイテレーションで解の探索を行う際に、暫定解から離れた点から開始し、徐々に暫定解に近づくよう探索を行う手法を提案する。このような手法を用いることで、暫定解から離れた点を大域的探索することと、暫定解の近傍を局所探索することを混在させることが可能となる。

解の探索では最初ランダムに重みを作成し、行列計算を行ってその時点で得られている最良の重みまでの解軌道を計算する過程で、各点における解の評価を行うことで望ましい解を探索する。提案手法では1つの重みを更新する際、初期値として同じ重みを2つ並べたベクトルを作成し、2次元行列を使って更新する。n 次元の重みのうち $i\ (=1,\cdots,n)$ 番目を更新する際、初期値 w_{i0} から0に収束させるのであれば、以下の式（3）から（5）の計算により時刻 t におけるベクトル $w_i(t)$ が0に十分近づくまでくり返し更新する[2]。なお、解の探索時には $w_i(t)$ が0に近づくまで式（4）から構成される行列 A_i を使用し、重みを更新した後に新たな A_i を作成する。

$$\dot{w}_i(t) = A_i w_i(t) \tag{3}$$

$$w_i(t) = \begin{bmatrix} w_{i1}(t) \\ w_{i2}(t) \end{bmatrix}, A_i = \begin{bmatrix} a_{i1} & a_{i2} \\ a_{i3} & a_{i4} \end{bmatrix} \tag{4}$$

$$w_{i1}(0) = w_{i2}(0) = w_{i0} \tag{5}$$

$$a_{i1}, a_{i2}, a_{i3}, a_{i4} \text{ は実数}$$

行列 A_i は、すべての固有値の実部が負となるならば、ベクトル $w_i(t)$ は時間とともに0に近づく。しかし以下の図で示す通り、固有値によって $w_{i1}(t) = w_{i2}(t) = 0$ に近づく際の軌道や速度は異なるものとなる。なお、図1ではベクトル $w_i(t)$ のノルムが0.1以下となるまで計算を行っている（以下の図2、図3も同様である）。

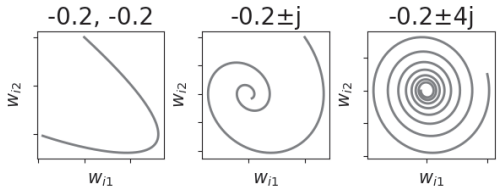

図1　固有値による軌道の違い

図上の値は固有値、初期値 $w_{i0} = 1$ として計算

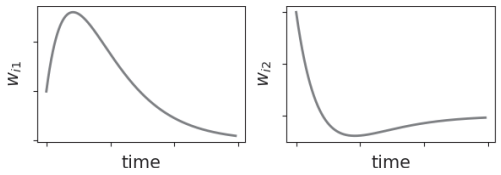

図2　固有値（重根）における時間応答

$w_{i1}(t), w_{i2}(t)$ のそれぞれの時間応答の傾向は大きく異なる。固有値を -0.2 の重根とした際の $w_{i1}(t), w_{i2}(t)$ の時間応答は図2のようになる。どちらも初期値1からの時間応答を表すが、$w_{i1}(t)$ は一度値が増加してから収束に向かうが、$w_{i2}(t)$ は減少してから収束する応答となっている。他の固有値でも同様である。

提案手法では、1つの重みを更新する際、あるイテレーションでは $w_{i1}(t)$ に沿って変化する値を使用するが、別のイテレーションでは $w_{i2}(t)$ に沿って変化する値を使用することで、異なる解軌道のもとで探索を行うことができる。さらには固有値を変えることでも軌道は大きく異なる。そこで、1回の探索ごとに固有値を変更すると同時に $w_{i1}(t)$, $w_{i2}(t)$ のどちらを使用するか選択することで、重みをさまざまに変化する軌道にしたがって更新でき、多様な探索が可能となる。

なお、連続系を使って $w_{i1}(t), w_{i2}(t)$ の更新を行うと、ルンゲ・クッタ法を用いて連立微分方程式

[2] ここでは単純化して0に収束する問題としているが、任意の目標値に収束させることは可能である。この点については後述する。

の解を求める必要があり計算量が増大する。そこで、本論文では連続系を時刻ごとにサンプリングした離散系に変換して計算に使用する。連続系への変換は、サンプリング時刻を T としたとき、以下の計算により計算した行列 P_i を使用する。P_i により次ステップの $w_i(k)$ を計算すると、現在時刻から T だけ後の値となる。

$$w_i(k+1) = P_i w_i(k) \tag{6}$$

$$P_i = e^{A_i T} = I + \sum_{j=1}^{\infty} \frac{A_i^j}{j!} T^j \tag{7}$$

$t = kT$, I は単位行列 , k はサンプリング回数

重みは複数あるため、全ての重みを 1 次元状にならべたベクトルおよび行列を構築して更新する。新たな重みベクトル W は以下の式により計算される。

$$W(k+1) = PW(k) \tag{8}$$

$$P = \begin{bmatrix} P_1 & 0 & \cdots & 0 \\ 0 & P_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & P_n \end{bmatrix} \tag{9}$$

$$W(k) = \begin{bmatrix} w_{11}(t) \\ w_{12}(t) \\ w_{21}(t) \\ w_{22}(t) \\ \vdots \\ w_{n1}(t) \\ w_{n2}(t) \end{bmatrix} \tag{10}$$

1 つの重みの更新に 2 つの重みを使用するため、$P \in R^{2n \times 2n}, W(k) \in R^{2n}$ となっている。たとえば n 個ある重みのうち 1 番目を更新する場合は、部分行列 P_1 と $[w_{11} \ w_{12}]^T$ が計算に使用される。

$$\begin{bmatrix} w_{11}(k+1) \\ w_{12}(k+1) \end{bmatrix} = P_1 \begin{bmatrix} w_{11}(k) \\ w_{12}(k) \end{bmatrix} \tag{11}$$

なお、$W(k)$ の目標値を 0 ではなく任意のベクトル Q とする場合は、次式により更新する。

$$W(k+1) = P\big(W(k) - Q\big) + Q \tag{12}$$

本論文では、目標値 Q は 1 回のイテレーションにおける重みの暫定解となる。i ($= 1, \cdots, n$) 番目の重みの暫定解を q_i^b とすると、Q は $q_1^b(t), q_2^b(t), \cdots, q_n^b(t)$ が 2 つずつ並んだ次の $2n$ 次元ベクトルとなる。たとえば $i = 1$ とすると、$w_1 = [w_{11} \ w_{12}]^T$ は式 (12) により $w_{11} \rightarrow q_1^b, w_{12} \rightarrow q_1^b$ となるように更新される。

$$Q = \begin{bmatrix} q_1^b \\ q_1^b \\ q_2^b \\ q_2^b \\ \vdots \\ q_n^b \\ q_n^b \end{bmatrix} \tag{13}$$

式 (12) で得られる W は $2n$ 次元となるが、ニューラルネットの重みは n 次元である。たとえば、式 (12) では W の 1 つ目の重みとして式 (11) のように $[w_{11}, w_{12}]$ の 2 つを計算するが、そのうちの 1 つだけを使用すれば良いため、計算後に 50% の確率でいずれか 1 つを選択してニューラルネットに設定する。2 つ目以降の重みも同様に、n 次元の重みを設定する。

提案手法により 1 回のイテレーションで探索する手順をまとめると、以下のようになる。この手順を指定したイテレーション数くり返すことで、探索は終了する。

提案手法の手順

1. 探索を開始する初期の重みを作成するために使用する標準偏差 σ を決定する。
2. 式 (10) で表される現時点の暫定解 W に平均 0、標準偏差 σ にしたがって発生させた正規乱数を加えた値を、1 回のイテレーション開始時の重みとする。

$$W \leftarrow W + \sigma \begin{bmatrix} \mathrm{rand}n_1(\sigma) \\ \vdots \\ \mathrm{rand}n_{2n}(\sigma) \end{bmatrix}$$

3. 式 (9) の行列 $P_1, \cdots, P_n$ を構築するために必要な固有値を決定して行列を作成する。
4. 式 (8)～(12) を用いて、新しい重みを計算する。
5. 4. で計算した重みを使い、環境で行動して得られた報酬の合計値を求める。
6. 4. と 5. を、指定したステップ数くり返す
7. 6. で得られた報酬が最大となった重みを、このイテレーションで得られた新たな暫定解とする。

3. シミュレーション

提案手法によるシミュレーションを実行した。使用したコンピュータは Windows 10 64bit、Core i7 − 8565U 1.80GHz、メモリ 16GB である。また、プログラムは Anaconda 3 の Python 3.7.7 にモジュー

ル Tensorflow 1.14.0、Keras 2.3.1、OpenAI Gym 0.15.7 を追加でインストールして作成した。

3.1 環境

提案手法で学習する環境は、OpenAI の Gym が提供する強化学習用のツールキットである。このキットには物理エンジンやレトロゲームなどを実際に動かして、実行内容をグラフィカルに表示できる環境が大量に用意されている。本研究では、その中から倒立振子と、レトロゲームの 1 つであるスペースインベーダを使用して強化学習し、提案手法の性能を検討する。

倒立振子とは、台車に取り付けられた自由に回転する棒を、台車を左右に移動させて一定期間（200 回行動する間）棒を立てたまま倒さないことが行動の目的となる。報酬は棒を倒さず継続できたステップ数とした。そのため、報酬の最大値は 200 となる。なお、この環境では報酬が 200 となった場合と、棒が倒れた場合に行動が終了となる。

この環境を実現するプログラムでは、OpenAI Gym に含まれる CartPole-v0 を使用した。観測できる状態は 4 次元（台車の位置、台車の速度、棒の角度、棒の角速度）、実行できる行動は 1 次元（左に押す、右に押す、の 2 通りのいずれか）である。

スペースインベーダは、OpenAI gym に含まれる SpaceInvaders-v0 を使用した。報酬は保有する自機の 3 機が攻撃されて行動が終了するまでに得た得点の合計とした。このゲームでは、観測できる状態として画面に表示されている 210 × 160 ピクセルのカラー画像、実行できる行動は 1 次元（何もしない、左に移動、右に移動、弾を撃つ、左に移動しながら弾を撃つ、右に移動しながら弾を撃つ、の 6 通りのいずれか）となる。インベーダでは、できるだけ高い得点を得ることが行動の目的となる。

3.2 学習器のモデル

本研究では Python のモジュール Keras により学習器を構築し、重みを提案手法により探索する。

倒立振子の学習では、学習器は 4 層のニューラルネットから構成される。入力層のニューロン数が 4、中間層の第 1 層が 8、第 2 層が 4、出力層が 2 となっている。入力層から出力層までのニューロンは全結合、出力層はソフトマックス関数により 2 通りの行動を選択する確率が出力される。行動は、学習器からの確率に応じてルーレット選択により 1 つを決定する。探索が必要な重みの総数は、中間層の第 1 層で、第 2 層で、出力層でとなる。

スペースインベーダでは学習器への入力として画面の情報を使用するが、元画像は 210 × 160 ピクセルのカラー画像で膨大な量となる。本研究では環境ラッパーにより探索に使用する情報を減らすことで、望ましい解を探索しやすい環境に変更する。

入力となる画像に対して、探索前に適用するラッパーは以下の 4 つとした。

1. DownSample
   画像を任意のサイズに縮小するラッパーである。本研究では縦横のサイズをそれぞれ半分として、入力画像を 105 × 80 ピクセルとした。
2. RGB2gray
   画像をグレースケールに変換するラッパーである。カラー画像は 1 ピクセルあたり赤緑青の 3 つの値から構成されるが、それを 1 つの値に減らすことができる。
3. ScaledFloatFrame
   グレースケールに変換された画像の 1 ピクセルの値は 0〜255 となるため、それを 255 で割ることで、0 から 1 の範囲に収めることで学習しやすく変更するラッパーである。
4. StochasticFrameSkip
   指定したフレーム数ごとに行動を行うラッパーである。さらに、確率的に 1 フレーム後にも同じ行動を実行するラッパーである。このラッパーによりフレームの内容は同一でも行動が変わり確率的な状況が生じることから、学習時にさまざまな行動を試すことが可能となる。

学習器は 2 層で構成され、第 1 層で入力となる 105 × 80 ピクセルの画像に対して畳み込み演算を行い、第 2 層では最初の層からの出力を 1 次元に並べる。ゲームで実行できる行動は 6 つあり、学習器からの出力をゲームでの行動とする。

第 1 層では入力画像をサイズが 5 × 5 ピクセルの 3 枚のフィルタを用意し、各フィルタを使って畳み込み演算を行った後に、により計算して出力する。畳み込み演算は、3 ピクセルずつずらしながら入力画像全体に対して行う。

$$\mathrm{ReLU}(x)=\begin{cases}0 & (x<0)\\ x & (x\geq 0)\end{cases} \tag{14}$$

ReLU はニューラルネットの学習で勾配法を用いる際に勾配消失問題を解消するために使用するものであり、本研究のように進化計算で重みの値を直接探すのであれば必ずしも使用する必要はない。提案手法において他の関数を使用することで、より望ましい結果が得られる可能性もあるが、他の関数を使用した場合との比較は今後の課題とする。

第 2 層では、第 1 層の出力すべてを 1 次元に並べたものを出力層となる 6 次元のニューロンに全結合し、ソフトマックス関数を計算して最大の値を出力する。これがゲームへの入力（6 入力のうち 1 つ）となる。行動は、学習器からの確率に応じてルー

レット選択により 6 通りの中から 1 つを決定して実行する。

探索が必要な重みの総数は、第 1 層で、第 2 層でとなる。

3.3 探索に用いるパラメータの設定

提案手法で設定するパラメータは、サンプリング時間、1 つの重みを更新する 2 次元行列を作成するための固有値（重みの個数だけ必要となる）、目標値に収束するまでのステップ数、探索をくり返す回数となるイテレーション数である。

式（7）で必要となるサンプリング時間は、目標値に収束するまでの時間を決定する。また、イテレーション回数との組み合わせでプログラムの実行時間が決定される。本論文では後に設定するくり返し回数との計算時間を考慮し、$T = 0.5$ とした。

固有値は、図 1 に示したとおり探索の際の複雑さに影響する。本論文では事前に固有値をさまざまに変更してシミュレーションを行い、軌道および時間応答の傾向が異なる以下の値を固有値として用いる。

$$\text{固有値の実部} \quad -0.2,\ -0.4$$
$$\text{固有値の虚部} \quad 0,\ j,\ 3_j,\ 5_j,\ 6_j$$

提案手法では、上述した実部、虚部からそれぞれランダムに 1 つ選択した値を固有値とする行列 A_i を作成し、それを離散化した P_i を作成した。指定した固有値から行列 A_i を作成する方法は、別の文献に示されている [8]。

行列 A_i の固有値の実部を -0.2 として、虚部を $0, j, 3_j, 5_j, 6_j$ と変更した際の軌道および w_{i1}, w_{i2} の時間応答を図 3 に示す。初期値は $w_{i0} = w_{i1}(0) = w_{i2}(0) = 1$ とした。なお、固有値の実部を -0.4 と大きな値に変更した場合、収束までのステップ数はおおよそ半分と短くなる。

固有値が -0.2 の重根の場合、初期値から目標値までの時間応答は振動することがない。しかし固有値に虚数が含まれると、時間応答はその頻度に違いはあるものの振動する。このように固有値による時間応答の違いを利用することで、多様な探索を実行することが可能となる。

目標値に収束するまでのステップ数は、固有値が -0.2 の重根の場合のみ 60 ステップ程度となっているが、他の固有値では 40 ステップ以内である。以上の結果から、1 つの重みでの探索ステップは、初期値から目標値までおおむね収束する 50 ステップとした。

イテレーション回数は、全体の計算時間に大きく影響し、どの程度の計算時間を利用できるのかという点に依存して決定する必要がある。本論文では倒立振子で 20、スペースインベーダで 100 とした。シミュレーション結果で示すとおり、倒立振子では短いイテレーション数で報酬の最大値 200 となったことから、イテレーション数を小さい値としている。

以上の設定で倒立振子を学習するために必要な提案手法でのイテレーション 1 回の実行時間は 50〜60 秒、インベーダーを学習するための実行時間は約 3 時間 20 分である。

OpenAI の手法では、倒立振子で 200 イテレーション、インベーダで 100 イテレーションとした。学習器は倒立振子よりスペースインベーダのほうが複雑であるが、シミュレーション結果をみるとわかるとおり、スペースインベーダではイテレーションが進めば必ず結果が改善されるわけではない。この点について事前の予備的シミュレーションを実行して確認し、倒立振子より短いイテレーション数としている。式（2）で加えるノイズの総数 $m = 20$、学習率 $\alpha = 0.1$ とした。

OpenAI の手法で倒立振子を学習するための計算時間は約 3〜5 分、インベーダは約 1 時間 40 分である。

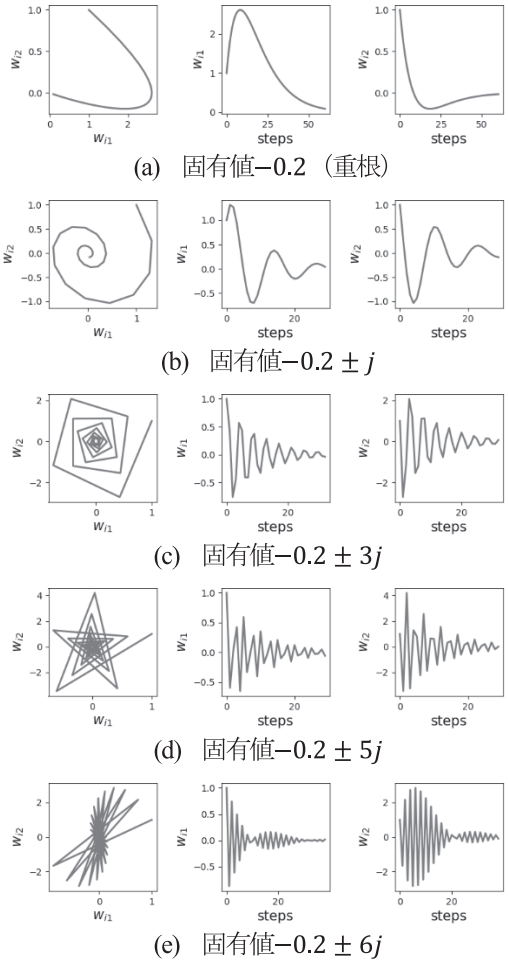

図 3 固有値の違いによる軌道と時間応答

シミュレーションは、標準偏差σを1, 5, 10と変化させて実行した。それぞれのσにおいて、100イテレーションとなるシミュレーションを1回として10回行った。

3.4 シミュレーション結果

提案手法とOpenAIの手法によるシミュレーション結果を示す。

3.4.1 倒立振子

倒立振子を用いて、σを変えて実行したシミュレーション結果を表1、表2に示す。シミュレーションは提案手法、OpenAIの手法のそれぞれで5回実行した。5回というシミュレーション回数は少ないが、事前のシミュレーションにおいて、本論文と同じパラメータでシミュレーションを行ったところ、本論文の結果と大きく異なる点はみられなかったことから結果の分析を行う上で問題のない回数と考える。

表1、表2では初めて報酬が200となったイテレーションと、報酬が200となる結果が5回連続した場合に安定的な解が獲得できたとして、その最初のイテレーションを示している。表中のNo.はシミュレーションの回数で、－となっている箇所はその結果が得られなかったことを表す。たとえば表1のσ = 0.5、No.4の結果では報酬が200となることなく、14イテレーションで報酬195となったのが最大値であったことから－と記している。またσ = 0.5、No.1では報酬が200となる結果が5イテレーション続くことがなかったため、結果が－となっている。

すべてのシミュレーション結果で提案手法、OpenAIの手法のいずれでも指定したイテレーション以内に報酬が最大値200となっている。提案手法ではσ = 1.0 , 5.0 , 10.0において報酬200となる状況が5回連続する結果がみられるが、OpenAIの手法では σ = 0.5のみでしかそのような結果が得られなかった。

図4はOpenAIと提案手法で、各イテレーションにおける暫定解で獲得した報酬の変化を表す。図示した結果は、OpenAIの手法では最も早く報酬200を獲得し、かつ5イテレーション連続して報酬200を獲得した結果を使用した。提案手法では、σ = 10 のNo.4の結果が1イテレーションで報酬の最大値を獲得し、そのまま最終イテレーションまで報酬200を獲得し続ける最良の結果であったが、この結果を図示してもイテレーションごとに獲得する報酬がどのように変化するか把握できない。その

表1　提案手法の結果（倒立振子）

σ	No.	初めて報酬が200となったイテレーション	報酬200が5イテレーション連続した結果の最初のイテレーション
0.5	1	8	－
	2	13	－
	3	17	－
	4	－	－
	5	17	－
1.0	1	5	5
	2	8	11
	3	19	－
	4	－	－
	5	13	13
5.0	1	4	4
	2	4	4
	3	6	6
	4	2	2
	5	16	16
10.0	1	1	6
	2	4	14
	3	2	2
	4	1	1
	5	3	7

表2　OpenAIの手法の結果（倒立振子）

σ	No.	初めて報酬が200となったイテレーション	報酬200が5イテレーション連続した結果の最初のイテレーション
0.5	1	40	80
	2	61	101
	3	72	134
	4	88	113
	5	40	62
1.0	1	44	－
	2	87	－
	3	24	－
	4	58	－
	5	97	－
5.0	1	29	－
	2	4	－
	3	28	－
	4	30	－
	5	5	－
10.0	1	2	－
	2	17	－
	3	27	－
	4	36	－
	5	12	－

ため、別の結果を使用した[3]。いずれの図でも、イテレーションが進むにつれて得られる報酬が高くなる結果となっていることがわかる。

また最終イテレーション時の暫定解を使って行動させたところ、常に報酬が 200 となることを確認した。

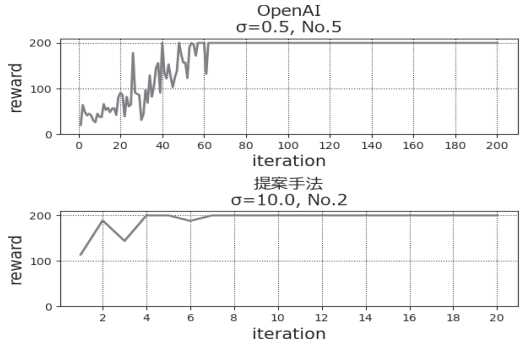

図 4　OpenAI と提案手法で暫定解が獲得する報酬

3.4.2　スペースインベーダ

スペースインベーダでは倒立振子のように報酬の最大値は存在しないため、得られた結果は獲得した得点の 100 イテレーション中の最大値と平均値を示した。3.4.1 項の倒立振子の結果をふまえて、提案手法では望ましい結果が期待できる $\sigma = 1.0, 5.0, 10.0$ としてシミュレーションを行った。OpenAI の手法も同様に、高い報酬が期待できる $\sigma = 0.5, 1.0$ の結果を示す。シミュレーションは提案手法、OpenAI の手法のそれぞれで 10 回実行した。提案手法、OpenAI の手法により得られた結果を、それぞれに表 3、表 4 に示す。

OpenAI の手法では、$\sigma = 0.5$ の No.7 で最大値が 1310 となり、提案手法を含めた全てのシミュレーションの中で最も良い値となっている。

イテレーションごとに得られる報酬の変化を図 5 に示す。倒立振子とは異なり、イテレーションが進むにつれて得られる報酬が高くなるわけではないことがわかる。また、報酬が最大となったのは 22 イテレーション目であるが、その前後の報酬は少なくなっている。これはゲームに登場する敵キャラクターが常に同じ行動を行うわけではないことが影響していると考えられる。

提案手法で得た報酬の最大値は、$\sigma = 5.0$ の No.9 で 1265 である。イテレーションごとに得られる報酬の変化は図 6 のとおりであるが、OpenAI の手法と同様に、イテレーションが進むにつれて得られる報酬が高くなってはいない。全体の結果をみると、提案手法では最大値が 1000 を超える回数が $\sigma = 5.0$ では 9 回、$\sigma = 10.0$ では 7 回と安定的であるのに対して、OpenAI の手法では $\sigma = 1.0$ では 5 回、$\sigma = 5.0$ では 4 回となっている。平均値については提案手法のすべての結果が OpenAI の手法を超えており、提案手法が有効に機能しているといえる。

最終イテレーション時の解を使ってゲームをプレイした結果を表 5 に示す。OpenAI および提案手法で、結果全体の中で報酬が最大となった結果でも、最終イテレーション時の解で得た報酬はそれぞれ 480、740 と全イテレーション中の最大値ではない。そこで、最終イテレーション時の暫定解で獲得した報酬が最大（1135）となった、提案手法の $\sigma = 10$, No.4 の暫定解を使った結果をあわせて示す。最終イテレーション時の解で得た報酬が高い結果ほど、ゲームをプレイした際の得点が高くなっていることがわかる。

表 3　提案手法の報酬（インベーダ）

σ	No.	最小	最大	平均
1.0	1	380	1200	662.25
	2	285	1050	787.85
	3	410	870	622.1
	4	410	1195	746.45
	5	315	1240	766.35
	6	440	1040	650.2
	7	410	925	738.4
	8	360	985	645.9
	9	310	995	643.85
	10	320	975	650.9
5.0	1	395	1175	725.6
	2	415	1025	639.8
	3	365	1005	640.4
	4	425	1110	669.75
	5	345	835	638
	6	435	1075	691.85
	7	445	1010	688.4
	8	380	1075	653.15
	9	335	1265	667.8
	10	360	1095	625.95
10.0	1	440	1085	746.65
	2	385	1010	694.55
	3	260	990	641.6
	4	305	1150	751.05
	5	415	845	637.25
	6	315	1285	646.8
	7	335	1130	698.3
	8	400	1050	626.25
	9	285	1080	623.0
	10	310	920	632.85

[3] 最も早く報酬 200 を獲得し、かつそのイテレーションよりも後に 5 イテレーション連続して報酬 200 を獲得した結果となる。

表 4　OpenAI の手法の報酬（インベーダ）

σ	No.	最小	最大	平均
0.5	1	290	1000	555.9
	2	285	860	523.05
	3	270	865	520.35
	4	285	895	454.05
	5	250	1095	556.1
	6	270	875	546.05
	7	260	1310	478.8
	8	285	1060	567.8
	9	215	890	439.95
	10	260	920	563.0
1.0	1	255	975	506.7
	2	280	940	529.25
	3	265	935	487.75
	4	235	895	497.2
	5	260	1005	499.5
	6	270	1050	513.75
	7	295	1045	508.9
	8	250	1105	502.6
	9	275	935	494.75
	10	285	1030	503.1

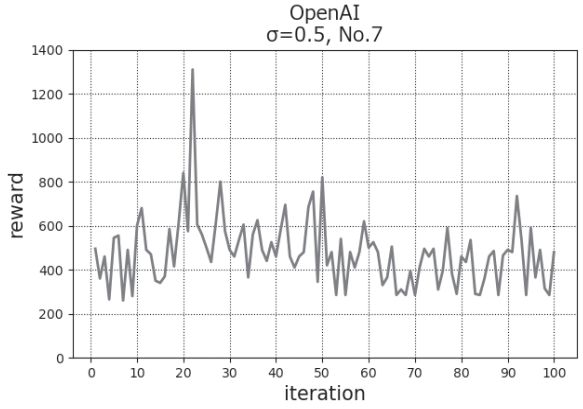

図 5　OpenAI で報酬が最大となった結果

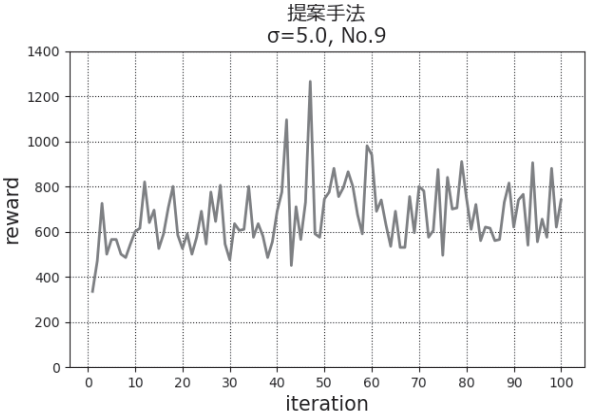

図 6　提案手法で報酬が最大となった結果

表 5　最終イテレーション時の暫定解を使ってゲームをプレイした結果
（）内は最終イテレーション時の暫定解で獲得した報酬

	OpenAI σ = 0.5 No.7 (480)	提案手法 σ = 5.0 No.9 (740)	提案手法 σ = 10.0 No.7 (1135)
最高得点	440	820	970
最低得点	75	120	110
平均	143.45	340.9	409.05

3.5　考察

倒立振子では、提案手法の σ = 10.0、No.4 の結果において 1 イテレーション目で報酬の最大値 200 となり、最終イテレーションまで最大値のままとなる結果がみられた。σ=10.0 の他の結果でも早い時点で 5 イテレーションにわたり報酬 200 を獲得する結果となっている。このことから、提案手法の探索が有効に機能していることがわかる。これは、σ = 5.0 とした場合でも同様の傾向となっている。そのため、σ を大きな値とすることで、大域的探索と局所探索により素早く環境に適応する行動が獲得できるといえる。

逆に σ = 1.0 , 0.5 と小さい値では、局所探索が主体となり 5 イテレーション連続して報酬 200 が続く結果が得られないケースがある。特に σ = 0.5 では、すべての結果で 5 イテレーションにわたり報酬が 200 と安定的に獲得する結果が得られていない。このことから、提案手法は大域的探索と局所探索の両方を実行できるよう、σ を大きな値とすることが有効と考えられる。

OpenAI の手法では σ ≥ 1.0 とした場合、報酬 200 を 1 回の入れテーションに限って得ることはできるが、5 イテレーション連続する結果はない。このことから、OpenAI の手法は σ を小さい値として局所探索を行う場合であれば良い結果が得られるが、大域的な探索手法として使う場合には機能しないと考えられる。

スペースインベーダでは、シミュレーション全体で最良の結果は OpenAI の手法で σ = 0.5 の No.7 である。しかし、OpenAI の手法は全ての結果で平均値が提案手法を上回ることがなく、1000 点を超える得点を得た回数も提案手法の方が多いことから、提案手法がより望ましい形で探索を行っているといえる。また、提案手法では 1000 点を超える得点となった回数は σ = 5.0 が 9 回と最も多く、σ = 1.0 では 5 回、σ = 10.0 では 7 回であることから、倒立振子と同様に提案手法では σ を大きな値とすることが望ましいといえる。

スペースインベーダでは登場キャラクターの行

動によって自らの行動も変わることから、最終イテレーションで得られた結果が必ずしも最良解にはならない。表5からわかるとおり、最終イテレーション時の解で得た報酬の高い結果が、ゲームをプレイした際に得る得点を最大とする行動を実現している。このことから、スペースインベーダにおいてはOpenAIおよび提案手法ではイテレーションが進むことで重みが必ず改善されるわけではないといえる。これは、現在の暫定解が次のイテレーションでプレイしたゲームでも同じ報酬を得られるわけではないことから生じると考えられる。過去に獲得した報酬の最大値を記憶しておき、その最大値を超える報酬となる解が得られた際にそれを新たな暫定解とする、といった方法で結果を改善できる可能性がある。

4. おわりに

本論文では深層強化学習のパラメータ更新を、多くのアルゴリズムが採用している勾配法の代わりに、値そのものを直接的に望ましい値に更新する手法を提案した。提案手法では、自由系により計算される軌道にしたがってパラメータを更新する。また自由系では固有値によって軌道を変えられるため、多様な軌道を実現する固有値を準備することで探索の効率化を図っている。

勾配法ではパラメータの更新に膨大な回数の行列計算が必要となりGPUを用いた計算が不可欠であるが、提案手法ではパラメータの値を直接求めることから、実用的な時間で望ましい解を求めることができる。また提案手法を用いたシミュレーション結果より、進化計算を用いる従来手法よりも高い性能を得られることを示した。

今後の課題として、現代のCPUは複数のコアを持つことからプログラムをマルチプロセス化することで、より高速に解を求められるようにすること、自由系の軌道を決定する固有値やサンプリング時間を調整し、さらに探索を効率化することがあげられる。

謝辞

本研究の一部は、情報科学研究所研究の助成によるものです。記して謝意を表します。

参考文献

(1) 布留川英一、“OpenAI Gym/Baselines 深層学習強化学習人工知能プログラミング実践入門”、ボーンデジタル（2020)

(2) V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, L. Antonoglou, D. Wierstra and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning”, arXiv preprint arXiv:1312.5602 (2013)

(3) H. v. Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-learning”、arXiv:1509.06461 (2016)

(4) OpenAI “OpenAI Baselines: ACKTR & A2C”, https://openai.com/blog/baselines-acktr-a2c/ (2017)

(5) J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust Region Policy Optimization”, arXiv:1502.05477 (2015)

(6) J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Proximal Policy Optimization”, arXiv:1502.05477 (2017)

(7) OpenAI, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning”, https://openai.com/blog/evolution-strategies/ (2017)

(8) 今 福 啓、“Quaantum-Inspired Evolutionary Algorithm における新たな解探索手法の提案”, Vol. 8, pp.5–12、獨協大学情報学研究（2018）

(9) 青野雅樹、“Keras によるディープラーニング”、森北出版（2019）

(10) 久保隆宏、“Python で学ぶ強化学習 [改定第2版]”、講談社（2019）