203-4in

近似モジュールの分割／統合による強化学習の実状態空間表現

A Real State Representation for Reinforcement Learning
by Dividing and Uniting Local Approximation Modules

濱口 大樹　　佐野 彰
HAMAGUCHI Hiroki　SANO Akira

龍谷大学大学院 理工学研究科 数理情報学専攻
Department of Applied Mathematics and Informatics, Graduate School of Science and Technology, Ryukoku University

We propose a self-organized on-line approximate method for real state functions. In our method, a real state function is approximated by a finite set of local linear function modules which are learned from individual input-output pairs. The local function modules are newly created by approximation error that exceeds the threshold, and united into one module by similarity between the neighboring local function modules. In this paper, we have confirmed approximation capability of the method on two learning tasks. One is the approximation problem on a nonlinear discontinuous function. Another is applying to real state representation of the action-value function on a reinforcement learning problem in continuous state space.

1. はじめに

強化学習は、環境からの報酬をもとに、エージェントが最適な行動を経験的に獲得していく機械学習の枠組みである。

初期の強化学習では、ボードゲームのような離散状態、離散行動をもつ有限マルコフ決定過程として記述される問題を対象としてきたが、近年ではロボットの行動制御のような連続的な入出力をもつ問題にも応用の幅を広げている。

連続的な空間では有限個のパラメータでどのように実状態空間を表現するかが問題となる。連続的な入出力を対象とする強化学習のパフォーマンスは、学習空間の表現能力、すなわち近似能力に大きく影響を受けるため、学習空間の表現方法は重要な課題の一つである。

実状態空間の表現方法としては、これまでにルックアップテーブル形式や線形和による関数近似手法が提案されている。ルックアップテーブル形式は、学習空間を任意の大きさのメッシュで区切りルックアップテーブルで表現する方法である。これは最も単純で容易に実装できる方法であるが、学習空間の次元が拡大するにつれ、計算コストが爆発的に増加する問題がある。また、テーブルの粒度が学習に大きく影響を及ぼすため、一般的にメッシュの大きさを定めるには設計者の経験に基づくチューニングが必要となる。

他方の線形和による関数近似方式は、学習空間をいくつかの基底関数の重み付き線形和で表現する方法がある。強化学習で用いられることの多い基底関数として、放射基底関数 (RBF :Radial Basis Function) を用いた RBF ネットワーク [釜谷 06] や、正規化ガウシアンネットワーク (NGnet: Normalized Gaussian Network)[関野 05] が代表的な手法である。

釜谷らは、学習過程においてユニットを逐次的に増やしていくことで、関数近似器の近似能力を補っている。しかし、ボトムアップ的なユニットの追加は、学習後期で結果的に無駄なユニットとなる場合があり、特に高次元で複雑な目的関数を持つ問題に適用する場合、計算機メモリを圧迫する原因となる。また、関数近似器の出力や基底関数パラメータの更新には、全基底を参照する必要があり、目的関数が複雑であるほど基底の増加に伴う計算コストが増す。

連絡先: hama@slab.math.ryukoku.ac.jp / 濱口 大樹，龍谷大学大学院理工学研究科

本論では、それぞれが独立に機能する局所的な関数近似モジュールでボトムアップ的に生成・配置しつつ、不要となったモジュールを統合・削減することで、記憶容量と計算量を抑制可能な関数近似器を提案する。

2. 部分空間近似モジュールによる関数近似器

n 次元の入力ベクトル $\boldsymbol{x} = (x_1, x_2, ...x_n) \in \boldsymbol{R}^n$ に対して未知の出力関数 $F(\boldsymbol{x}) \in \boldsymbol{R}$ によって与えられる出力を $y \in \boldsymbol{R}$ とする。ここで入出力の組 $(\boldsymbol{x}, y)$ の集合が与えられたとき、$F(\boldsymbol{x})$ の近似関数 $\hat{F}(\boldsymbol{x})$ を学習によって獲得することを考える。提案する関数近似器は, 入力空間の部分空間をそれぞれ独立に近似する関数モジュールの有限集合 $M = \{(\boldsymbol{c}_i, f_i)|i \in \boldsymbol{N}\}$ によって構成される。ここで $\boldsymbol{c}_i \in \boldsymbol{R}^n$ は i 番目の関数モジュールが近似する入力空間を指定する代表ベクトルであり、また $f_i : \boldsymbol{R}^n \rightarrow \boldsymbol{R}$ はこの部分空間上の近似関数である。近似関数 f_i には任意の関数を用いることができるが、本論では (1) 式に示す単純な線形関数を考える。

$$f_i(\boldsymbol{x}) = \boldsymbol{a}_i(\boldsymbol{x} - \boldsymbol{c}_i) + b_i \tag{1}$$

ただし、$\boldsymbol{a}_i \in \boldsymbol{R}^n$ 及び $b_i \in \boldsymbol{R}$ は、それぞれ線形関数の勾配とバイアスのパラメータとする。

2.1 出力値の算出

関数近似器の出力 $\hat{F}(\boldsymbol{x})$ は、入力ベクトル $\boldsymbol{x}$ と、代表ベクトル $\boldsymbol{c}_i$ が最も近い最近傍モジュールの近似関数 $f_p(\boldsymbol{x})$ によって近似される。ここで、p は入力ベクトル $\boldsymbol{x}$ の最近傍モジュールの番号であり、

$$p = \arg\min_i d(\boldsymbol{x}, \boldsymbol{c}_i) \tag{2}$$

とする。ただし、$d(\boldsymbol{x}, \boldsymbol{c}_i)$ は $\boldsymbol{x}$ と $\boldsymbol{c}_i$ 間の距離を示す距離関数とし、本論ではユークリッド距離を用いる。したがって、関数近似器の出力 $\hat{F}(\boldsymbol{x})$ は、

$$\hat{F}(\boldsymbol{x}) = f_p(\boldsymbol{x}) \tag{3}$$

となる。

2.2 モジュールの追加と更新

M 上の各関数モジュールは、近似誤差 δ を用いて逐次的に追加／更新される。時刻 t での入力 $\boldsymbol{x}_t$に対する 近似関数 $f_p(\boldsymbol{x}_t)$ の近似誤差 δ を

$$\delta = |y_t - \hat{F}(\boldsymbol{x}_t)| = |y_t - f_p(\boldsymbol{x}_t)| \tag{4}$$

とする。

2.2.1 モジュールの更新

近似誤差 δ がモジュール追加に関する閾値 ε_{add} に対して、(5) 式を満たすとき、最近傍モジュールの近似関数 $f_p(\boldsymbol{x})$ のパラメータ $\boldsymbol{a}_p$、b_p を (6) 式で更新する。

$$\delta < \varepsilon_{add} \tag{5}$$

$$\boldsymbol{a}_p \leftarrow \boldsymbol{a}_p + \eta\,\delta\,(\boldsymbol{x} - \boldsymbol{c}_p), \quad b_p \leftarrow b_p + \eta\,\delta \tag{6}$$

ただし、$\eta (0 \leq \eta \leq 1)$ を学習率とする。

2.2.2 モジュールの追加

近似誤差 δ が閾値 ε_{add} に対して (7) 式を満たすとき、入力ベクトル $\boldsymbol{x}$ を代表ベクトルとする新たな関数モジュールを M に追加する。

$$\delta \geq \varepsilon_{add} \tag{7}$$

追加されるモジュールは、

$$(\boldsymbol{c}_{new}, f_{new}) = (\boldsymbol{x}_t, y_t) \tag{8}$$

とする。また、近似関数 f_{new} のパラメータ $\boldsymbol{a}_{new}$、b_{new} は次式で与える。

$$\boldsymbol{a}_{new} = (0, 0, \cdots 0), \quad b_{new} = \delta + f_p(\boldsymbol{x}) = y_t \tag{9}$$

強化学習では、一般的に目的関数の値域を推測することはできない。そのため本論では、ε_{add} の値を小さめに設定し、積極的にモジュールを追加する。

2.3 モジュールの統合

強化学習における学習初期では、$(\boldsymbol{x}, y)$ の入力によって目的関数 $F(\boldsymbol{x})$ の値が不連続に変化することがある。提案手法ではこのような不連続な値の変化をモジュールの追加によって表現している。一方で、学習の進展に伴って、隣接する関数モジュールが同一関数を表現することが考えられる。そこで、隣接する類似モジュールを統合することにより、モジュール数の削減を行い、近似に必要な記憶容量を抑制する。

集合 M 上の関数モジュール $(\boldsymbol{c}_i, f_i)$ の最近傍モジュール $(\boldsymbol{c}_j, f_j)$ について、互いの近似関数が、互いの領域を閾値 ε_{del} の誤差範囲でカバーできるとき、これらは単一の関数モジュールで表現できると判断し、2 つのモジュールを統合する。すなわち、本論では近似関数に単純な線形関数を用いているので、(10) 式を満たすとき、2 つのモジュールを統合する。

$$|f_i(\boldsymbol{c}_j) - f_j(\boldsymbol{c}_j)| + |f_j(\boldsymbol{c}_i) - f_i(\boldsymbol{c}_i)| < \varepsilon_{del} \tag{10}$$

モジュール統合では、統合される 2 つのモジュールを削除すると同時に、新たな関数モジュール $(\boldsymbol{c}_{new}, f_{new})$ を追加する。統合された新たな関数モジュール $(\boldsymbol{c}_{new}, f_{new})$ について、新たな代表ベクトル $\boldsymbol{c}_{new}$ は、

$$\boldsymbol{c}_{new} = \frac{\boldsymbol{c}_i + \boldsymbol{c}_j}{2} \tag{11}$$

であり f_{new} のパラメータ $\boldsymbol{a}_{new}$、b_{new} は、

$$\boldsymbol{a}_{new} = \frac{\boldsymbol{a}_i + \boldsymbol{a}_j}{2}, \quad b_{new} = \frac{b_i + b_j}{2} \tag{12}$$

とする。関数モジュール $F(\boldsymbol{x})$ の追加と更新が、入出力 $(\boldsymbol{x}_t, y_t)$ が与えられる度に行われるのに対して、モジュール統合はある任意の周期 τ 毎に M 上の全てのモジュールに対して処理される。

3. 不連続関数の近似実験

提案した関数近似器の有効性を確認するために、(13) 式に示す目標関数 $F(x_1, x_2)$ に対して関数近似を行う。$F(x_1, x_2)$ は連続的で緩やかな曲面と平坦な面をもちつつ、中間部分は断絶された不連続な出力をもつ。

$$F(x_1, x_2) = \begin{cases} -0.2((x_1 - 5)^2 + (x_2 - 5)^2) + 10 & (x_1 > 5) \\ 5.0 & \text{otherwise} \end{cases} \tag{13}$$

(13) 式を真の価値とし、十分な回数 (step=505000) 学習させる。以下では、提案手法において 2 つの閾値 ε_{add}、ε_{del}、及び入力 (x_1, x_2) の分布が関数モジュールの学習に与える影響を調べる。

3.1 閾値の学習への影響

モジュールの追加／統合の閾値 $\varepsilon_{add}, \varepsilon_{del}$ の値は、近似される目的関数の精度と配置されるモジュールの総数に影響を与える。そこで閾値 $\varepsilon_{add}, \varepsilon_{del}$ について以下の実験を行う。

入力変数には範囲 [0:10] の一様乱数を与える。モジュール分割の閾値 $\varepsilon_{add} = 6.0$、モジュール統合の閾値 $\varepsilon_{del} = 3.0$ とした場合、また、$\varepsilon_{add} = 0.1$、$\varepsilon_{del} = 0.01$ として提案手法による関数近似を行った。ただし、学習率を $\eta = 0.2$、モジュール統合の周期を $\tau = 10000$ とする。

図 1 は目標関数 $F(x_1, x_2)$ の、$\varepsilon_{add} = 6.0, \varepsilon_{del} = 3.0$ での近似関数である。ここで図 1 の上部は目標関数の近似値であり、下部は関数モジュールの代表ベクトル $\boldsymbol{c}_i$ の分布を表す。

図 1 より、モジュール分割／統合の閾値を大きなの値に設定すると、関数近似器は目的関数を粗く近似することがわかる。なぜなら、モジュールの分割は (7) 式より消極的に行われ、かつ、統合は (10) 式より積極的に行われるためである。逆に、モジュール分割／統合の閾値を小さなの値に設定すると、関数近似器は状態空間を精度よく近似する (図 2)。このとき、目的関数近似のための近似モジュールの総数は、大きく増加していることがわかる。

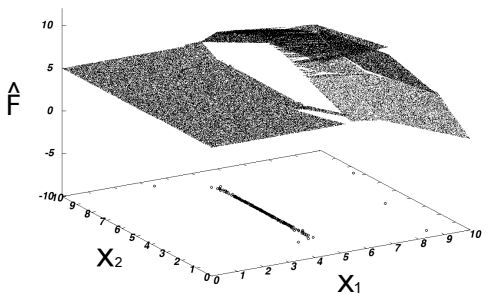

図 1: $F(x_1, x_2)$ の近似関数とモジュール分布 ($\varepsilon_{add} = 6.0, \varepsilon_{del} = 3.0$)

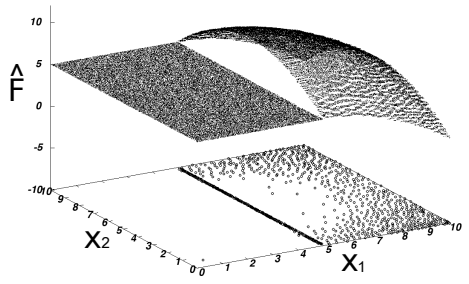

図 2: $F(x_1, x_2)$ の近似関数とモジュール分布 ($\varepsilon_{add} = 0.1, \varepsilon_{del} = 0.01$)

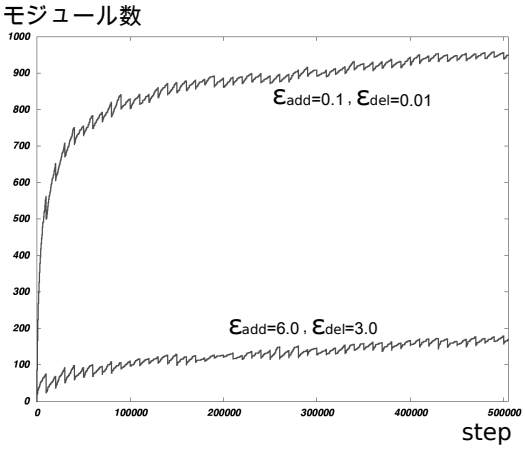

図 3: 近似モジュール数の時間変化 (閾値)

3.2 入力分布の学習への影響

強化学習では、入力変数が張る空間全域を一様に探索することはない。強化学習の性質上、エージェントはその時々で価値が高い部分を重点的に探索する。すなわち、入力変数に入ってくる入力値は一様ではなく、偏りが生まれる。関数近似器は、その部分空間を必要な精度で近似できればよい。これを想定して、入力の分布に偏りをもたせることを考える。

そこで平均値 5.0、標準偏差 1.0 の正規分布に従う乱数を区間 [0:10] で生成し、これを入力変数 x_1, x_2 の値とする。

ただし、関数近似のパラメータは、$\eta = 0.2$、$\varepsilon_{add} = 0.1$、$\varepsilon_{del} = 0.01$、$\tau = 10000$ とした。

図 4 は入力分布に偏りをもたせた場合の、目的関数の近似結果である。前節の、一様な入力分布で目的関数を近似した結果と比較する。また、step 毎の近似モジュール数の変移を図 5 に示す。

実験に用いた非一様な入力分布は、座標の中央 $(x_1 = 5, x_2 = 5)$ 付近で高頻度の入力がある。図 4 より、偏りのある入力分布では近似される関数形が異なることがわかる。ここで重要なのは、高頻度な入力ベクトル付近の目的関数がうまく近似されているかという点である。入力分布に偏りのある場合、中央付近では近似精度がよく、離れるにつれて精度の悪い近似になっていて、学習に必要な箇所を近似するように近似モジュールが配置されていることがわかる。

図 5 は近似モジュール数の時間変化を示したものである。入力が一様な場合と非一様な場合では、生成される近似モジュールの総数が異なることがわかる。断絶的な目的関数を最近傍モジュールの近似関数で表現するためには、境界付近に多くの近似モジュールを配置する必要がある。入力分布の偏りは、座標中央に集中しており、近似モジュール総数の差異は、目的関数の断絶部分を表現するために多くの近似モジュールが配置されたためと考えられる。

4. 実験と考察

3 章では、目的関数 $F(\boldsymbol{x})$ は時間に関して一定でありかつ、入出力の分布も一定で変化しない。強化学習では、$F(\boldsymbol{x})$ や入出力の分布は非一様であり、学習の進行に伴って変化する。提案する関数近似器が強化学習特有の性質のもとでうまく働くかを具体的な強化学習の問題に適用して調べる。

4.1 MountainCarTask

提案する関数近似器を用いて強化学習の学習システムを構築する。強化学習の問題としては MountainCarTask[Sutton 98] を取り上げる。これは、出力不足の台車で山を登ることを想定した問題である。台車は山の頂上に到達することを目的とするが、台車自身の出力だけでは急な坂を登りきることができない。そのため、あえて頂上と逆方向の山に進み、その山を下る推進力を利用して頂上を目指す必要がある。

台車が観測できる状態は、台車の座標上での位置 $x(-1.2 \leq x \leq 0.5)$ と、台車の速度 $v(-0.07 \leq v \leq 0.07)$ であり、次式で更新される。

$$v \leftarrow v + 0.001\,a - 0.0025\,\cos(3x) \tag{14}$$

$$x \leftarrow x + v \tag{15}$$

台車が選択できる行動 a は離散的で 3 種類であり、$a \in \{-1, 0, 1\}$ とする。また、報酬として 1step ごとに -1 の報酬を与え、台車が山の頂上に到達したとき、強化学習の 1 回の試行が終了する。台車の初期速度と初期位置は、それぞれの値の取り得る区間内で一様乱数を用いて初期化する。

4.2 関数近似器の設定

本論では、状態空間 S のみが連続で、行動空間 A は離散な場合を考え、強化学習でしばしば用いられる Q-Learning で MountainCarTask を検証する。

Q-Learning では、状態行動空間 $S \times A = \{s, a\}$ を行動価値関数 $Q(\boldsymbol{s}, \boldsymbol{a})$ で推定することで学習を行う。よって、関数近似器は行動価値関数 $Q(\boldsymbol{s}, \boldsymbol{a})$ を近似する。強化学習において時刻 t における目標関数の出力 y_t は、$Q(s_t, a_t)$ の近似であればよいので次式で与えられる。

$$y_t = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'_{t+1}) \tag{16}$$

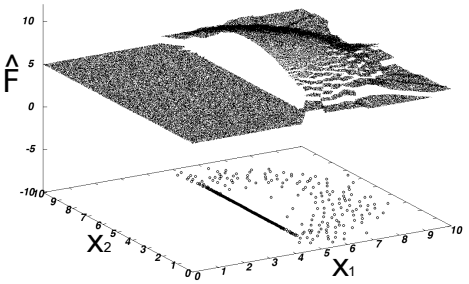

図 4: 非一様入力での近似関数とモジュール分布 ($\varepsilon_{add} = 0.1, \varepsilon_{del} = 0.01$)

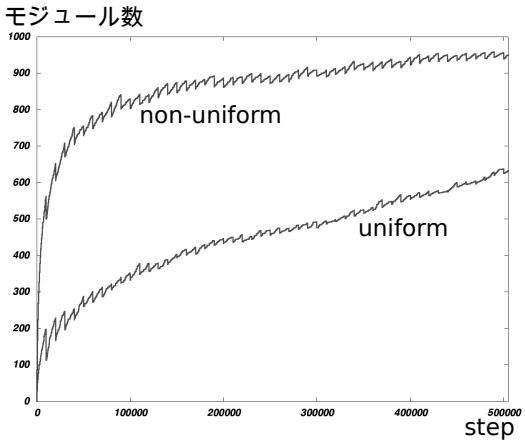

図 5: 近似モジュール数の時間変化 (入力分布)

ここで $\gamma(0 \leq \gamma \leq 1)$ は割引率であり、r_{t+1} は、エージェントが状態 s_t から s_{t+1} に遷移するときに得られる報酬である。また、近似誤差 δ には次式で定義される TD(temporal-difference) 誤差を用いる。

$$\delta = y_t - \hat{Q}(s_t, a_t) \tag{17}$$

4.3 実験結果

学習率を $\eta = 0.2$、割引率 $\gamma = 0.9$ とする。また、モジュールの追加／統合の閾値をそれぞれ、$\varepsilon_{add} = 0.1, \varepsilon_{del} = 0.01$ とする。行動選択には ε-greedy 方策を用い、$\varepsilon = 0.1$ とする。ε-greedy 方策では、行動選択は ε の確率でランダムに行動することで状態空間探索し、$1 - \varepsilon$ の確率で現時点で最善の行動方策をとる。

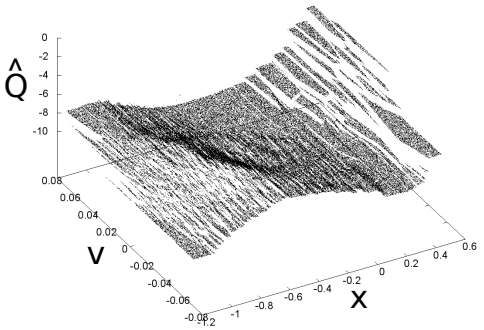

図 6: 近似された行動価値関数 $\hat{Q}(s, a)$

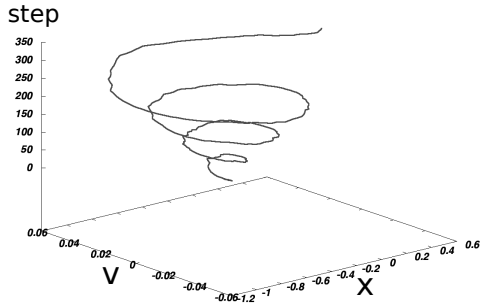

図 7: 学習によって得られた状態遷移ルート

図 6 はある状態 s における可能な行動 a 中で、最大の行動価値関数 $\hat{Q}(\boldsymbol{s}, a)$ を示したものである。1 回の試行を最大 3000step とし 1000 試行学習した。山の頂上付近の価値は高く、逆に遠い地点の価値は低くなっており、適切に学習が行われているとわかる。

図 7 は、1000 試行終了後に学習獲得した状態遷移ルートを示したものである。状態遷移ルートは、状態 s_t における最善の行動方策に従って行動した結果得られた状態遷移ルートをトレースしたものである。時間経過に伴い、台車が両側の山を往復しながら頂上に登る様子がわかる。

5. おわりに

本論では、部分空間を独立に近似する関数近似モジュールを用い、強化学習への適用を想定した関数近似器を提案した。この関数近似器は、任意の目的関数を局所関数モジュールの自己組織的な分割／統合によって近似表現できることを確認した。また、Q-Learning における状態行動価値関数を提案した関数近似器で近似表現し、実状態空間上強化学習問題である MountainCarTask に対して学習実験を行い、この近似器が強化学習上で有効に機能することを確認した。

提案手法の近似能力と強化学習での有効性を示す為には、$\varepsilon_{add}, \varepsilon_{del}$ という 2 つの閾値パラメータが関数モジュールの数と近似精度に与える影響を明らかにしなければならない。また、大規模な実空間上の強化学習への適用を想定し、強化学習をターゲットとする他の近似手法と記憶容量や計算量に基づく性能比較も必要である。

参考文献

[Sutton 98] Sutton, R. S. and Barto, A. G.: Reinforcement Learning : An Introduction., MIT Press, 1998.

[釜谷 06] 釜谷博行, 北山数行, 藤村敦子, 阿部健一:連続状態空間における強化学習:計測自動制御学会東北支部 第 229 回研究集会: 229–11, 2006.

[関野 05] 関野正志, 片山大輔, 新田克己: 基底関数間相互作用に基づく状態空間自己組織化: 第 19 回人工知能学会全国大会予稿集 1D3–04, 2005.