AutoEncoderを用いた状態次元削減による強化学習の効率化

○桑原泰裕　小野典彦　永田裕一　（徳島大学）

Efficiency Improvement of Reinforcement Learning by State Dimension Reduction using AutoEncoder

∗Y. Kuwahara , N. Ono and Y. Nagata (University of Tokushima)

Abstract− Reinforcement learning (RL) is widely used to learn actions of robots for performing a given task in unknown environments. However, it takes a lot of time for learning good policies for the task when the state and action space is high-dimensional and continuous. In this research, we proposed a method of reducing the dimension of the state by using AutoEncoder to promote the efficiency of RL.

Key Words: 強化学習, ニューラルネットワーク, AutoEncoder

1　序論

現在，工場などの生産作業で活躍するような，決められた作業をこなすロボットの制御規則を作成することは人間にも可能である．しかし，ロボットに未知の環境で目的を遂行させるためには，固定された制御規則を用いるだけではなく，ロボット自身にその環境に適した制御規則を学習させることが必要になる．そこで，ロボットの制御規則の学習に強化学習を使用することにより，この問題を解決する試みが多くなされている．強化学習では状態入力に対して正しい行動出力を明示的に示すのではなく，環境から得られる報酬を手掛かりに，その報酬が最大になるような方策を学習するため，未知の環境に対して報酬を設計するだけでタスクを遂行するための行動を学習することができる．しかし，複雑なロボットの制御を学習するためには，連続で膨大な状態空間および行動空間を探索する必要があり，このような問題に強化学習を適用することは困難であるのが現状である．

本研究では，AutoEncoder[1] を用いて状態の次元を削減することにより，より効率的に強化学習を行うことを目的とする．AutoEncoder とは，ニューラルネットワークにおいて，教師データとして入力と出力が同じものを用いて教師あり学習させたものである．この時，中間層の次元を入力，出力の次元よりも小さく設定することにより，入力データの情報圧縮をすることができる．AutoEncoder を用いて状態次元削減を行う強化学習手法としては Deep Fitted Q-Iterations(DFQ)[2] が存在するが，DFQ では制御対象の画像を AutoEncoder により次元圧縮していたのに対して，本研究では制御対象のダイナミクスを AutoEncoder により次元圧縮することにより，強化学習を効率化することを目的としている．

2　対象とする強化学習

本研究では状態，行動がともに連続空間であるロボットの制御問題を対象としているため，AutoEncoder によって状態の次元削減を行う強化学習手法として，森本らによって提案された Actor-Critic 法における状態価値関数，行動関数を NGnet により表現した手法 [3] を用いた．本章ではその手法について簡単に解説していく．

2.1　収益

ある方策 π にしたがって，環境中で行動を実行した際の時間ステップ 0 から未来にわたって得られる報酬の期待値を収益と呼ぶ．特に，遠い未来に得られる報酬ほど価値割引率 γ で割り引いた収益を割引期待報酬と呼び，次式で表される．

$$J(\pi) = \underset{p_0, P_T,\ \pi}{E}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right] \tag{1}$$

- $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$：状態 $\boldsymbol{s}$ において行動 $\boldsymbol{a}$ を実行したときに得られる報酬
- γ：将来の報酬における割引率．$0 < \gamma < 1$
- $\underset{p_0, P_T,\ \pi}{E}$：初期状態分布 $p_0(\boldsymbol{s})$，状態遷移確率 $P_T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$，方策 $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ によって確率分岐する期待値を表す．

2.2　状態価値関数

エージェントが時間に依存しない方策 π をとるとき，状態 $\boldsymbol{s}$ を初期状態とみなして，将来にわたり得られる収益を状態価値関数 $V^\pi(\boldsymbol{s})$ と呼ぶ．状態価値関数は次式で定義される．

$$V^\pi(\boldsymbol{s}) = \underset{P_T,\ \pi}{E}\left\{\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \middle| \boldsymbol{s}_0 = \boldsymbol{s}\right\} \tag{2}$$

強化学習の目的は全ての状態 $\boldsymbol{s}$ において状態価値関数 $V^\pi(\boldsymbol{s})$ を最大にする最適方策関数 π^* を求めることである．

2.3　TD 誤差

タイムステップ t において，行動を実行した結果，エージェントが状態 $\boldsymbol{s}_t$ から状態 $\boldsymbol{s}_{t+1}$ に遷移して報酬 r_t を得たとする．この時，状態価値関数の定義から，$V(\boldsymbol{s}_t)$ と $r_t + \gamma V(\boldsymbol{s}_{t+1})$ の期待値は同じになるはずであるが，状態価値関数が真の値でない場合には両者は同じ値とはならない．そこで TD 誤差 $\delta(t)$ は次の式で定義される．

$$\delta(t) = r_t + \gamma V(\boldsymbol{s}_{t+1}) - V(\boldsymbol{s}_t) \tag{3}$$

TD 誤差を用いることで，状態価値関数は次式のように更新される．ただし， α は学習率である．

$$V(\boldsymbol{s}_t) = V(\boldsymbol{s}_t) + \alpha\delta(t) \tag{4}$$

2.4 適格度トレース

TD 学習は $R_t = r_t + \gamma V(\boldsymbol{s}_{t+1})$ を目標値として $V(\boldsymbol{s}_t)$ をこの値に近づけていく学習法である．すなわち，この手法は $V(\boldsymbol{s}_t)$ の学習を 1 ステップ後の観測情報を元に行っていることになる．この手法の拡張として，$V(\boldsymbol{s}_t)$ の学習を n ステップ後の観測情報（途中経過も含む）に基づいて行う方法がある．具体的には目標値

$$\begin{aligned} R_t^{(n)} = r_t + \ \gamma r_{t+1} + \ \gamma^2 r_{t+1} + \ldots + \\ \gamma^{n-1} r_{t+n-1} + \gamma^n \ V(\boldsymbol{s}_{t+n}) \end{aligned} \tag{5}$$

に対して $V(\boldsymbol{s}_t)$ の学習を行う． ここで，状態価値関数の定義から，状態価値関数が真の値を持つ場合，$V(\boldsymbol{s}_t)$ と $R_t^{(n)}$ の期待値は等しくなることに注意されたい．さらに，$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$ という $R_t^{(n)}$ に重み付けした量を定義しても，状態価値関数が真の値を持つ場合には，R_t^{λ} と $V(\boldsymbol{s}_t)$ の期待値は等しくなる．ただし λ は $0 < \lambda < 1$ の定数である．

$$(\boldsymbol{s}) = \begin{cases} \lambda e(\boldsymbol{s}) + 1 & (\boldsymbol{s}_t = \boldsymbol{s}) \\ \lambda e(\boldsymbol{s}) & (otherwise) \end{cases} \tag{6}$$

- λ：適格度の減衰率を調整するパラメータ

この式では，各ステップにおいて，全ての状態に対する適格度を λ 分減衰し，標本された状態の適格度に 1 を加えている．これにより，最近頻繁に標本された状態の適格度は大きくなり，あまり標本されない状態の適格度は小さくなる．

適格度トレースを使った価値関数の更新式は次のようになる．

$$V(\boldsymbol{s}) = V(\boldsymbol{s}) + \alpha e(\boldsymbol{s})\,\delta(t) \tag{7}$$

この式は式 (4) と違い，全ての価値関数 $V(s)$ について更新する．

2.5 正規化ガウス関数ネットワーク

制御対象の状態や行動が連続空間である場合，連続空間を離散化して状態価値関数や行動関数をルックアップテーブルで表現することも可能であるが，連続状態のまま強化学習を行う手法として，本研究では正規化ガウス関数ネットワーク (NGnet) を用いる．NGnet は Fig. 1 に示すような 3 層のネットワークで，中間素子は正規化されたガウス関数である．

NGnet では，与えられた n 次元ベクトル $\boldsymbol{x}$ に対して、k 番目のユニット (中間素子) の基底関数は次のように計算される。

$$a_k(\boldsymbol{x}) = e^{-\frac{1}{2}||\boldsymbol{M}_k(\boldsymbol{x}-\boldsymbol{c}_k)||^2} \tag{8}$$

- $\boldsymbol{c}_k$：基底関数の中心ベクトル
- $\boldsymbol{M}_k$：基底関数の形状を決定する体格行列．対角成分の値は各入力の大きさに合わせて調節する．

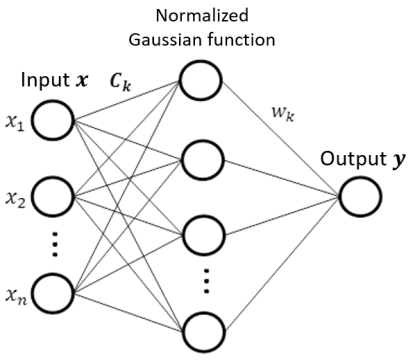

Fig. 1: Structure of NGnet

ただし，実際には $a_k(\boldsymbol{x})$ の総和が 1 になるように正規化した

$$b_k(\boldsymbol{x}) = \frac{a_k(\boldsymbol{x})}{\sum_{l=1}^{K} a_l(\boldsymbol{x})} \tag{9}$$

- K：ユニットの個数

がユニットの出力となる．

NGnet を用いて関数近似を行う場合，重みの学習は次の式のように，実際の出力と目標出力との誤差を修正するように，ネットワークの重み w_k を更新することで行われる．

$$w_k = w_k + \eta(\hat{y}(\boldsymbol{x}) - y(\boldsymbol{x}))\, b_k(\boldsymbol{x}) \tag{10}$$

- $\hat{y}(\boldsymbol{x})$：目標出力
- η：学習係数

これは，$\hat{y}(\boldsymbol{x})$ と $y(\boldsymbol{x})$ の 2 乗誤差を最小化する勾配法に相当する．

2.6 Actor-Critic 法

Actor-Critic 法とは，Critic と呼ばれる評価ネットワークと，Actor と呼ばれる制御ネットワークを用いて，状態価値関数 $V(\boldsymbol{s})$ と行動関数 $\mu(\boldsymbol{s}) = \boldsymbol{a}$ を別々に学習させる手法である．

Actor-Critic 法では下記 1. から 4. の流れを繰り返し，Critic では正しい評価値 $V(\boldsymbol{s})$ を，Actor では $V(\boldsymbol{s})$ を最大にする行動関数 $\mu(\boldsymbol{s})$ を学習する．

1. Actor が環境から制御対象の状態 $\boldsymbol{s}$ を受け取る．
2. Actor が行動関数 $\mu(\boldsymbol{s})$ より，その状態 $\boldsymbol{s}$ における行動 $\boldsymbol{a}$ を環境に出力する．
3. 環境がその行動を実行し，Critic と Actor に状態 $\boldsymbol{s}$ を，Critic に報酬 r を出力する．
4. Critic 受け取った状態 $\boldsymbol{s}$ と報酬 r より TD 誤差を計算し，その TD 誤差をもとに Critic と Actor を更新する．

本研究では Critic の出力 $V(\boldsymbol{s})$ と Actor の出力 $\mu(\boldsymbol{s})$ の表現を NGnet を用いておこなう．本来は $V(\boldsymbol{s})$ と $\mu(\boldsymbol{s})$ で同じ状態を入力としているため，同じ NGnet を用いることができるが，本研究では後述するように

$V(\boldsymbol{s})$ と $\mu(\boldsymbol{s})$ で入力が異なるため，NGnet を別々に用意する．

NGnet を用いた Critic における出力 $V(\boldsymbol{s})$ の式は次式で表される．

$$V(\boldsymbol{s}) = \sum_{k=1}^{K} v_k b_k^C(\boldsymbol{s}) \tag{11}$$

- K：Critic における中間素子の個数
- v_k：Critic におけるネットワークの重み
- $b_k^C(\boldsymbol{s})$：Critic における中間素子の出力

ネットワークの重み v_k は次式で更新する．

$$\Delta v_k = \alpha e_k \delta(t) \tag{12}$$

- α：Critic における学習係数
- e_k：適格度

適格度の更新式は次式で表される．

$$e_k = b_k^C(\boldsymbol{s}) + \lambda e_k \tag{13}$$

式 (13) は NGnet を用いた連続空間の強化学習における適格度の更新式を表している．離散空間では，適格度が状態 $\boldsymbol{s}$ それぞれに割り振られていたのに対して，こちらは NGnet の基底関数 b_k それぞれに割り振られているといえる．適格度 e_k は 1 ステップにつき λ の割合で減衰していき，基底関数が活性化している場合，活性化している大きさに合わせて大きくなる．つまり，離散時間においては最近頻繁に標本された状態の適格度ほど大きくなるのに対して，NGnet を用いた連続空間においては，最近頻繁に活性化した基底関数の適格度ほど大きくなる．

Actor の j 番目の出力式は次式で定義される．

$$\mu_j(\boldsymbol{s}) = \mu_j^{max} g\left(\sum_{l=1}^{L} w_{lj} b_l^A(\boldsymbol{s}) + \sigma n_j(t)\right) + \mu_{bj} \tag{14}$$

- L：Actor における中間素子の数
- w_{lj}：Actor におけるのネットワークの重み
- b_l^A：Actor における中間素子の出力
- σ:探索のためのノイズの大きさを決めるパラメータ
- $n_j(t)$：探索のためのガウスノイズ
- $g(x)$：シグモイド関数
- μ_j^{max}：出力の幅
- μ_{bj}：出力のバイアス

Actor を更新する際は，NGnet の出力にノイズを加え，そのときの TD 誤差 $\delta(t)$ により，ネットワークの重みを更新する方向を決める．つまり，TD 誤差が正だった場合，ノイズを加えたときの報酬が，ノイズを加えなかった場合の想定報酬よりも多くもらえたと考えることができ，その時のノイズの方向にネットワークの重みを更新し，TD 誤差が正だった場合はその時のノイズの逆方向にネットワークの重みを更新する．

ネットワークの重み w_{lj} は次式で更新される．

$$\Delta w_{lj} = \beta\ \delta\ (t) \sigma\ n_j(t)\ b_l^A(\boldsymbol{s}) \tag{15}$$

- β：Actor における学習係数

3 Deep Fitted Q-Iterations

3.1 AutoEncoder

AutoEncoder とはニューラルネットワークの一種で，訓練データとして入力データと出力データが同じものを用いて教師あり学習させたものである．この時，中間層の次元が入力層，出力層よりも大きい場合は，入力をそのままコピーすることで入力と同じ値を出力できるようになるが，中間層の次元を入力層と出力層の次元よりも小さく設定した場合はそのままコピーすることができないため，入力データを何らかの形で圧縮する必要がある．そのため，次元が小さくなっている中間層では入力データの特徴を学習することができる．また，AutoEnocder は特徴抽出のほかにデータのノイズ除去や異常検知，クランスタリングなどにも用いられる．

3.2 Deep Fitted Q-Iterations

AutoEncoder を強化学習に応用した手法として Deep Fitted Q-Iterations(DFQ) が存在する．DFQ は Fitted Q-Iteration[4] と AutoEncoder を組み合わせた学習モデルである．強化学習に用いる状態入力として，環境から得た状態入力をそのまま用いるのではなく，AutoEncoder により次元削減したデータを用いる．

DFQ の学習の基本的な流れを次に示す．

1. 現在の行動関数に従い制御対象の 1 エピソード分の状態遷移を記録する．
2. 1. で記録した状態を訓練データとして AutoEncoder の教師あり学習を行う．
3. 1. で記録した状態遷移を用いて強化学習を行う．この時，状態として AutoEncoder により圧縮された状態を用いる．
4. 終了条件を満たすまで 1. から 3. を繰り返す．

また，DFQ では 6 × 6 マスの迷路問題対象としており，状態入力として 6 × 6 マスを 30 × 30pixel の画像データで表したものを用いていた．

4 提案手法

本研究では，DFQ を一般的に迷路問題よりも難しいとされている倒立振子運動に適用する．Fitted Q-Iteration では行動が連続空間である場合に適用できないため，前述したように本研究では森本らによって提案された Actor-Critic 法における状態価値関数，行動関数を NGnet により表現した手法を用いる．

4.1 提案手法における工夫点

迷路問題と比較して，倒立振子問題のような制御問題では取り得る状態が膨大であるため，DFQ で用いるような AutoEncoder の学習法をそのまま適用して強化学習を行ってもうまく学習することができない．その問題を解決するため，本研究では次の 3 点の工夫を行った．

(a) AutoEnocder による状態の圧縮を行動関数に用いた場合，AutoEncoder 学習後の状態の特徴空間の変化により状態遷移を観測する際の行動が大きく変化してしまうため，行動関数の入力には AutoEnocder により圧縮されていない元の状態を用いた．

(b) DFQ では入力とする 1 エピソード分の状態を復元するような AutoEncoder の学習を，誤差が十分少なくなるまで学習していたが，それでは 1 エピソード毎に状態の特徴空間が大きく変化していまい，強化学習を行う際に状態価値関数がうまく適応できなくなってしまう．そこで，本研究では 1 エピソード毎の学習は訓練データに対して 1 度だけ学習を行い，少しずつ学習するようにした．

(c) DFQ では強化学習の 1 エピソード毎に，そのエピソードで得られた状態のみを用いて AutoEncoder の学習を行っていた．しかし，倒立振子問題においては，直前の 1 エピソードでは本来探索すべき状態空間のほんの一部しか経験しないため，この方法では AutoEncoder のオーバーフィッティングが生じてしまう．そこで本研究では AutoEncoder の汎化性能を高め，より多くの状態の特徴を抽出できるようにするため，Deep Q-Network[5] や Deep Deterministic Policy Gradient method[6] などの深層強化学習で用いられている ReplayBuffer の考え方を元に，直前の 1 エピソードだけでなく，過去数エピソード分の状態を状態バッファに保存しておき，学習する際はその状態バッファに保存されている状態のいくつかを訓練データとして学習を行うようにした．

4.2 提案手法のアルゴリズムの流れ

本研究における学習の流れを Fig. 2 に示す．Fig. 2 中の状態遷移サンプルでは，TD 誤差を求めるのに必要な状態 $\boldsymbol{s}_t$，報酬 r_t, 次ステップの状態 $\boldsymbol{s}_{t+1}$ と行動関数 $\mu(\boldsymbol{s})$ を更新するのに必要な探索のためのノイズ $\sigma\boldsymbol{n}(t)$ を 1 エピソード分保存しておき，強化学習に用いる．そして，状態バッファでは状態 $\boldsymbol{s}$ を数エピソード分保存しておき，AutoEnocder の教師あり学習に用いる．

Fig. 2 の詳細は以下の通りである．

1. 状態バッファを初期化する．
2. 状態遷移サンプルを初期化し，シミュレーションの状態をリセットする．
3. Actor が環境からステップ t における状態 $\boldsymbol{s}_t$ を受け取る．
4. Actor が行動関数 $\mu(\boldsymbol{s}_t)$ により状態 $\boldsymbol{s}_t$ における行動 $\boldsymbol{a}_t$ を環境に出力する．

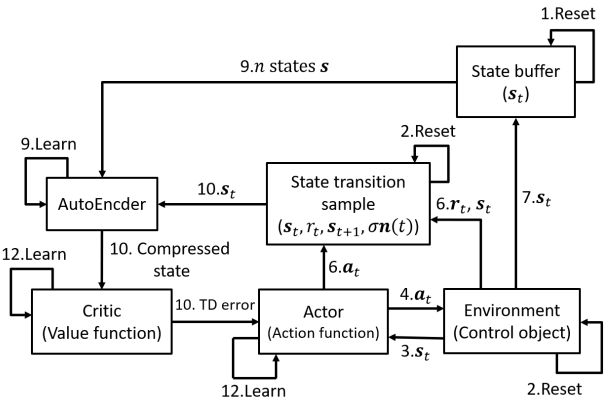

Fig. 2: Flow of the proposed method

5. 環境は実際に行動 $\boldsymbol{a}_t$ を実行して次の状態 $\boldsymbol{s}_{t+1}$ へ移行し，報酬 r_t を計算する．
6. ステップ t における状態遷移 $(\boldsymbol{s}_t, r_t, \boldsymbol{s}_{t+1}, \sigma\boldsymbol{n}(t))$ を状態遷移サンプル保存する．
7. 状態 $\boldsymbol{s}_t$ を状態バッファに保存する．この時，状態バッファの大きさが規定サイズを超えている場合，1 番最初に保存された状態は破棄する．
8. 3. から 7. を 1 エピソード分繰り返す．
9. 状態サンプルからランダムに n 個の状態を取り出し，それを訓練データとして 1 度だけ AutoEnoder の学習を行う．
10. 状態遷移サンプルから 1 ステップ分の状態遷移 $(\boldsymbol{s}_t, r_t, \boldsymbol{s}_{t+1}, \sigma\boldsymbol{n}(t))$ を取り出し，それを用いて状態価値関数 $V(\boldsymbol{s})$ と行動関数 $\mu(\boldsymbol{s})$ の更新値 $\Delta V(\boldsymbol{s}), \Delta\mu(\boldsymbol{s})$ をそれぞれ式 (12) と式 (15) により求める．
11. 10. を 1 エピソード分繰り返す．
12. 状態価値関数 $V(\boldsymbol{s})$ と行動関数 $\mu(\boldsymbol{s})$ を更新値 $\Delta V(\boldsymbol{s}), \Delta\mu(\boldsymbol{s})$ により更新する．
13. 2. から 12. を終了条件を満たすまで繰り返す．

5 実験設定

提案手法で示した 3 点の工夫の有用性を確かめるため，以下の設定で倒立振子問題の学習結果を比較する．

- AutoEncoder を用いない場合
- AutoEncoder を用いたがどの工夫も適用しなかった場合
- AutoEncoder を用いて工夫 (a) のみを適用した場合
- AutoEncoder を用いて工夫 (a) と (b) を適用した場合
- AutoEncoder を用いて工夫 (a)，(b)，(c) 全てを適用した場合

実験設定は以下に示す．

- モデルの状態の取得，行動の出力，報酬の取得は全て 0.01 秒ごとに行い，1000 ステップごとにシミュレーションをリセットする．これを 1 エピソードとし，2000 エピソード終了した時点で学習を終了する．
- 状態空間を台車の x 軸方向位置 x，x 軸方向速度 x'，振り子の角度 θ，振り子の角速度 θ' の 4 次元，行動空間を台車に x 軸方向に加える力 F の 1 次元とした．
- 振り子の高さを-1 から 0 に正規化したものを報酬とした．ただし，台車の x 軸方向位置 x と振り子の角速度 θ' が一定値を超えている時はペナルティとして報酬は-1 となる．また，1 エピソード間における各ステップの報酬を合計したものを 1 エピソードの合計報酬とした．
- 状態次元の圧縮には Fig. 3 のような 5 層の AutoEncoder を用いた．入力層と出力層が同じになるようにミニバッチ方式のバックプロパゲーションにより学習させ，次元数が入力層と出力層よりも小さい 3 次元となっている 3 層目で次元を圧縮する．

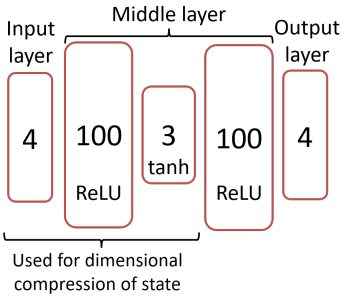

Fig. 3: Structure of AutoEncoder

- 学習直前 100 エピソード分を状態バッファに保存しておき，その中からランダムに 1000 ステップ分の状態を選択し，それを AutoEncoder の訓練データとした．
- AutoEncoder を用いない場合では NGnet の基底関数は状態空間 4 次元の各次元ごとに等間隔に 5 個ずつ，$5 \times 5 \times 5 \times 5 = 625$ 個を格子状に配置した．そして圧縮した状態空間では 3 次元の各次元ごとに等間隔に 7 個ずつ，$7 \times 7 \times 7 = 343$ 個を格子状に配置した．

6 実験結果

実験結果を Fig. 4 に示す．

グラフの横軸がエピソード数，縦軸が 1 エピソードの合計報酬，そして青点が合計報酬のプロット，茶線が前 100 エピソードの平均線を示している．

Fig. 4 より，AutoEncoder に状態の次元圧縮を用いなかった場合，基本的にエピソード数が進むごとに合計報酬が増えていっており，最終的に合計報酬-150 程度で倒立振子運動を学習できていることが分かる．

しかし，AutoEncoder を用いたがどの工夫も適用しなかった場合，ほぼ完全にランダムに行動し，倒立振子運動を全く学習することができなかった．

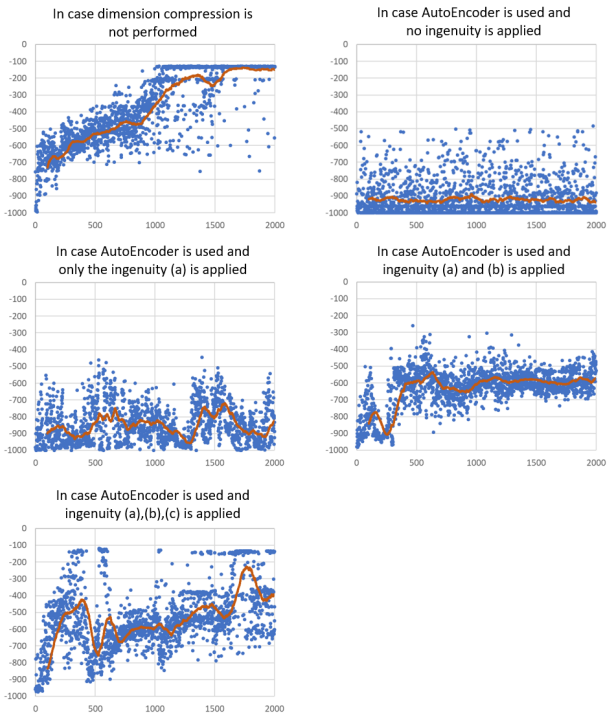

Fig. 4: Performance of the different settings

AutoEncoder を用いて工夫 (a) のみを適用した場合もどの工夫も適用しなかった場合よりは学習をしているが，安定して学習することはできなかった．

そして AutoEncoder を用いて工夫 (a) と (b) を適用した場合，1 エピソードの合計報酬が-700 から-500 になる程度の運動までは学習できているが，倒立振子運動までは学習できなかった．

最後に AutoEncoder を用いて工夫 (a)，(b)，(c) 全てを適用した場合，安定はしていないが工夫 (a) と (b) を適用した場合よりも最大合計報酬が高くなっており，学習途中で倒立振子運動を学習することができているのが分かる．

7 考察と結論

7.1 考察

AutoEncoder を用いたがどの工夫も適用しなかった場合と工夫 (a) のみを適用した場合のグラフを比較すると，AutoEncoder を用いたがどの工夫も適用しなかった場合，工夫 (a) のみを適用した場合と比べて，全く学習できていない事が分かる．状態価値関数に次元圧縮された状態を用いる場合はすでに観測された状態を圧縮するが，行動関数に次元圧縮された状態を用いる場合はまだ観測されていない未知の状態を次元圧縮することになり，AutoEncoder がその状態の圧縮の仕方を学習できていないためこのような結果になったのだと考えられる．

次に工夫 (a) のみを適用した場合と工夫 (a) と (b) を適用した場合のグラフを比較すると，工夫 (a) のみを適用した場合に比べて，工夫 (a) と (b) を適用した場合，1 エピソード毎の合計報酬が-700 から-500 になるまでは安定して学習できていることが分かる．これは 1 エピソード毎の AutoEnocder の変化が小さくなり，それにともない状態価値関数の学習が安定化したためだと考えられる．

最後に工夫 (a) と (b) を適用した場合と工夫 (a)，(b)，(c) 全てを適用した場合のグラフを比較すると，工夫 (a) と (b) を適用した場合に比べて，工夫 (a)，(b)，(c) 全てを適用した場合，安定性は低くなったが最大合計報酬は高くなっており，学習途中で倒立振子運動を学習できていることが分かる．これは AutoEncoder が特徴抽出できる状態の範囲が広くなり，様々な状態の特徴を汎用的に抽出できるようになっているためだと考えられる．また，学習が不安定になっているのは AutoEnocoder の汎用性が高くなった代わりに直近の状態に対する特徴抽出の精度が落ちてしまっているためだと考えられる．

7.2 結論

AutoEncoder を用いて状態の次元圧縮を行うことにより強化学習の効率化を行う手法である DFQ アルゴリズムを倒立振子問題に適用する際の「1 エピソード毎の状態入力が大きく変化し，それにともない AutoEncoder の出力も大きく変化する」という問題点に対して，「行動関数の入力には AutoEncoder により圧縮されていない元の状態を用いる」「AutoEncoder の 1 エピソード毎の学習は訓練データに対して 1 度だけ行う」「直前の 1 エピソードだけでなく，過去数エピソード分の状態を用いて AutoEncoder の学習を行う」という 3 つの工夫点を示し，その有用性を示すことができた．

しかし，それら 3 つの工夫を用いても倒立振子問題を安定して解くことはできておらず，安定して学習させるためにはさらなる工夫が必要になると考えられる．

7.3 今後の課題

AutoEnocder による状態の特徴抽出の精度は訓練データに依存する．そのため，安定して学習させるためには観測された状態を全て状態サンプルに保存していくのではなく，学習に必要なデータのみを厳選して保存していく必要があると考えられ，どのようにして学習に必要なデータと不要なデータを選別するのかを今後の課題としたい．

参考文献

1) Geoffrey E. Hinton, R. R. Salakhutdinov : Reducing the Dimensionality of Data with Neural Networks, Science, Vol. 313 Issue 5786, 504/507 (2006)
2) Sascha Lange, Martin Riedmiller：Deep Auto-Encoder Neural Networks in Reinforcement Learning, The 2010 International Joint Conference on Neural Networks (IJCNN) (2010)
3) 森本淳・銅谷賢治：強化学習を用いた高次元連続状態空間における系列運動学習 —起き上がり運動の獲得—，電子情報通信学会論文誌 Vol.J82-D-II No.11，2118/2131 (1999)
4) Martin Riedmiller : Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method, European Conference on Machine Learning (2005)
5) Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin A. Riedmiller : Playing Atari with Deep Reinforcement Learning, NIPS Deep Learning Workshop (2013)
6) Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra : Continuous control with deep reinforcement learning, ICLR (2016)