制御システム論分野研究会資料

2020 年 9 月 24 日

M2 竹内 維吹

進捗報告 9.25

1　前提

入力アフィン系のセルフトリガー制御を考える.

$$\dot{s} = f(s) + g(s)a \tag{1}$$

1.1　倒立振子による実験

倒立時の振子の角度を $\theta = 0$ とし, 加えられる入力が $A = [-10\mathrm{N}\cdot\mathrm{m}, 10\mathrm{N}\cdot\mathrm{m}]$ と制限されるような倒立振子を考える. この倒立振子のダイナミクスは, 以下のように与えられる.

$$\frac{\mathrm{d}}{\mathrm{dt}}\begin{pmatrix}\theta\\ \dot{\theta}\end{pmatrix} = \begin{pmatrix}\dot{\theta}\\ \frac{3g}{2l}\sin\theta + \frac{3}{ml^2}a\end{pmatrix} \tag{2}$$

コンピュータで強化学習を行う場合, これを離散化したシステムについて計算を行う必要がある. 上記の状態方程式を離散化すると以下のようになる.

$$\theta_{t+1} = \theta_t + \dot{\theta}_t\delta_t + \frac{3g}{2l}\sin\theta_t\delta_t^2 + \frac{3}{ml^2}a\delta_t^2 \tag{3a}$$

$$\dot{\theta}_{t+1} = \dot{\theta}_t + \frac{3g}{2l}\sin\theta_t\delta_t + \frac{3}{ml^2}a\delta_t \tag{3b}$$

ただし, δ_t は離散化定数である.

2　現状確認

2.1　セルフトリガー制御

図 1 のような制御系を考える.

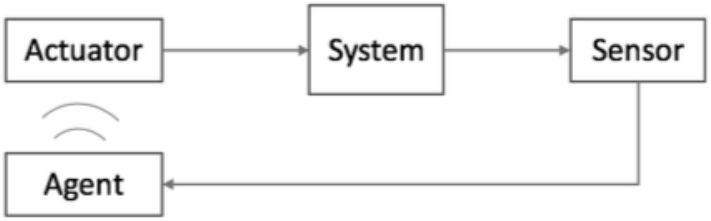

図 1　制御系

これに対するフィードバック制御を考える. 状態変数 s を観測してアクチュエータに入力信号を送信することを「インタラクション」と呼ぶと, セルフトリガー制御では, 連続的なインタラクションは行わずに, 次のイ

ンタラクションを何秒後に行うかをエージェントが決定する．それを数式上で表すため, エージェントの制御則 $\pi(s)$ は 2 つの要素からなるベクトル値関数であるとし, 1 つ目の要素はアクチュエータに送信する入力 a, 2 つ目の要素は次にインタラクションを行うまでの時間間隔 τ (s : 秒) を表すものとする. また, 次のインタラクションを行う時刻までは 1 つ前のインタラクションで送信した入力 a を加え続けるものとする (ZOH 制御).

2.2 目標点の確認

研究を通しての目標は「安全性を確保しながら, 最適セルフトリガー制御則 π^* の強化学習を実現させること」である。ここで

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, J(\pi) \tag{4}$$

$$J(\pi) = \mathbb{E}_{s_0 \in d_0}[V^\pi(s_0)] \tag{5}$$

$$V^\pi(s_0) = \sum_{i=0}^{\infty} \gamma^i C_i^\pi \tag{6}$$

$$C_i^\pi = -\int_{T_i}^{T_{i+1}} s(t)^\top Q s(t) \mathrm{d}t + \tau_i a_i^\top R a_i + \lambda \tau_i,\ T_i = \sum_{l=0}^{i} \tau_l \tag{7}$$

であり, π_1, π_2 は π の第 1, 第 2 成分である．また, i はインタラクションの回数を示し, a_i, τ_i はそれぞれ i 回目のインタラクションでの方策 π の出力である．

さて, 一般的に強化学習では, 1 ステップ 1 ステップの行動の良し悪しを評価して方策を更新していく．インタラクションとインタラクションの間の区間を「インターバル」と呼ぶと, 式 (6) より, この問題は各インターバルを 1 ステップとした強化学習問題であると考えることができる．

以下では方策 π を θ でパラメトライズし, $\theta^* = \underset{\theta}{\operatorname{argmax}}\, J(\pi_\theta)$ を解くことによって $\pi^* = \underset{\pi}{\operatorname{argmax}}\, J(\pi)$ を得るものとする．方策勾配を用いた強化学習では $\nabla_\theta J(\pi)$ を用いて θ^* を求める．その際方策勾配 $\nabla_\theta J(\pi)$ の近似のため, 実環境とのインタラクションによって得られたデータ組 $\{s, a, r, s'\}$ を用いる．「学習中の安全」という言葉を, 「このデータ組の収集を決められた安全領域 $\mathcal{C}$ の内部でのみ行うこと」と定義する．

2.3 実現可能性の検証:サンプル値系での実験

上記の目標を達成する見込みがあるのかを検証するために, サンプル値系での実験を行う．サンプル値系では, セルフトリガー制御と同様に連続的なインタラクションは行わない．セルフトリガー制御との違いは, インタラクションの間隔がエージェントによって状態 s 依存で決定するのではなく, 制御問題の設定として定数 t_{int} で与えられる点である. したがってサンプル値系での制御方策 π_{sample} は, アクチュエータに送信する入力信号 a のみを出力する関数として与える．

サンプル値系での実験により, $t_{\mathrm{int}} = 0.001(s)$ のサンプル値系での最適方策

$$\begin{cases} \pi_{\mathrm{sample},1} = -\underset{\pi_{\mathrm{sample}}}{\operatorname{argmax}} \displaystyle\sum_{i=0}^{\infty} \int_{it_{\mathrm{int}}}^{(i+1)t_{\mathrm{int}}} s(t)^\top Q s(t) \mathrm{d}t + t_{\mathrm{int}} a_i^\top R a_i \\ a_i = \pi_{\mathrm{sample}}(s(it_{\mathrm{int}})) \end{cases} \tag{8}$$

を初期方策として, $t_{\mathrm{int}} = 0.002(s)$ のサンプル値系での最適方策

$$\begin{cases} \pi_{\mathrm{sample},2} = -\underset{\pi_{\mathrm{sample}}}{\operatorname{argmax}} \displaystyle\sum_{i=0}^{\infty} \int_{it_{\mathrm{int}}}^{(i+1)t_{\mathrm{int}}} s(t)^\top Q s(t) \mathrm{d}t + t_{\mathrm{int}} a_i^\top R a_i \\ a_i = \pi_{\mathrm{sample}}(s(it_{\mathrm{int}})) \end{cases} \tag{9}$$

を学習中の安全性を満たしながら学習できるかを検証する．

2.4　セルフトリガー制御への発展

前節での検証によって, インタラクション間隔を大きくしても安全強化学習を行うことが可能であることを確認できたとする. サンプル値系での制御則は入力信号 a のみを出力する関数であったので, 入力信号 a とインタラクション間隔 τ の二つの要素を出力する必要があるセルフトリガー制御の初期方策として方策 $\pi_{\mathrm{sample},1}$ をそのまま用いることはできない.

そこで代替策として,

$$\begin{cases} \pi_1(s) = \pi_{\mathrm{sample},1}(s) \\ \pi_2(s) = 0.001 \end{cases} \tag{10}$$

とする方策 π_{init} をセルフトリガー制御の強化学習のための初期方策として用いる.

3　安全性の定義

3.1　インタラクション間隔 τ の安全性

ECBF(後から書きます)

3.2　入力信号 a の安全性

強化学習ではデータの収集に環境とのインタラクションを行う必要がある. DDPG と呼ばれるアルゴリズムは方策オン型の強化学習とよばれ, データの収集方策に学習中の暫定最適方策を用いる. したがって, 学習初期の方策では安全性が保証されないことがしばしばある. この課題を解決するために, 制御バリア関数を用いる.

関数 $h(s)$ が以下の条件を満たす時, システム (1) に対する制御バリア関数であるという.

$$\sup_{a \in A} \left\{ \frac{\partial h}{\partial s}(f(s) + g(s)a) + K(h(s)) \right\} \geq 0 \tag{11}$$

ただし, $K(s)$ はクラス K 関数である.

さて, 2.2 節にて登場した安全領域 $\mathcal{C}$ を

$$\mathcal{C} = \{ s \in S \mid h(s) \geq 0 \} \tag{12}$$

として与える. このとき $h(s)$ が制御バリア関数であるならば, 状態 $s \in C$ を初期状態とした時, それ以降の全時刻において, 状態 s が $s \in C$ を満たすようにする入力が存在することを保証する. そのような入力集合は現時刻での状態 s に依存し,

$$U(s) = \left\{ a \in A \mid \frac{\partial h}{\partial s}(f(s) + g(s)a) + K(h(s)) \geq 0 \right\}, \forall s \in C \tag{13}$$

としてその集合を与える.

学習中の安全性を確保するために, 図 2 のようにエージェントの出力を $U(s)$ の要素に射影するレイヤーを設ける.

当面は, エージェントの出力 a_π に最も近い $U(s)$ の元への射影を考える.

4　今後の方針

4.1　シミュレーション環境の構築

さて, 制御バリア関数による安全性保証は連続システムに対して行われるものである. また, セルフトリガー制御則の第 2 成分 τ の出力ロジックを勾配によって更新できるように, τ を連続値として扱う必要がある. しかし, コンピュータ上でシミュレーションを行うには (1) を時間に関して離散化を行わなくてはならない.

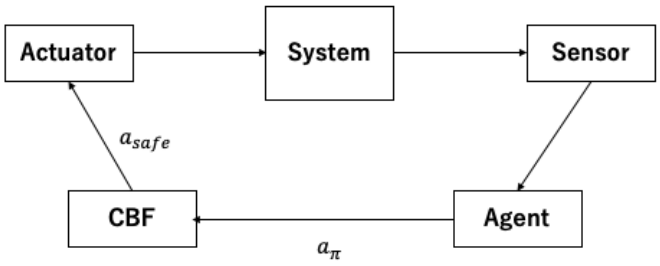

図 2　制御系

そこで, インタラクション間隔 τ を整数個に等間隔に分割し, その時間幅を用いて離散化を行う. この時, 離散化幅は $0.001(s) \sim 0.005(s)$ になるように分割数を調整する. δ_t を上から抑える理由は離散化誤差を抑えるためである. (τ を) 下から抑えるのは, コンピュータ上で $\frac{1}{\infty} = 0$ となってしまうからで, それを回避するためである.

ここで, 式 (7) のインターバル報酬 C_i^π が定積分を用いて表されているため, これをシミュレーション環境で近似する手法を考える. ダイナミクス (1) の離散化幅 δ_t の離散近似システムが

$$s_{t+1} = f_d(s_t, \delta_t) + g_d(s_t, \delta_t)a_t \tag{14}$$

と書かれているとする. インタラクション間隔 τ を N 分割した時, $\delta_t = \frac{\tau}{N}$ を用いて

$$C_i^\pi \approx -\delta_t \sum_{k=0}^{N} s_{n_i+k}^\top Q s_{n_i+k} + \tau_i a_i^\top R a_i + \lambda \tau_i \tag{15}$$

と近似する. ここで s_{n_i} は i 回目のインタラクションを行った瞬間の状態変数 $s(T_i)$ と同じ値が代入されるものとする.

4.2　サンプル値系での安全性確保ロジックの構築

2.3 節で記述した通り, t_{int} が $0.001(s)$ の最適方策 $\pi_{\text{sample},1}$ を初期値として, $0.002(s)$ の最適方策 $\pi_{\text{sample},2}$ を安全強化学習できるのか検証する. 本節ではその安全性の確保方法についてもう少し掘り下げて議論する.

ECBF を用いることによって, 入力 a に対するインタラクション間隔 τ の限界を与えることができる. その値を $\tau_{\max}(a)$ と書く. もし $\tau_{\max}(a) < 0.002$ であるなら, CBF を用いて $U(s)$ の元 a_{safe} を選び $\tau_{\max}(a_{\text{safe}}) \geq 0.002$ となれば, 次のインタラクションまで a_{safe} を加え続けても状態変数が $\mathcal{C}$ を出ていくことはない. しかし $\forall a \in U(s)$ に対して $\tau_{\max}(a) < 0.002$ であるなら, 次のインタラクションを $0.001(s)$ 後に行うことで, 安全性を確保する必要がある. $(0.001 \leq \tau_{\max}(a) < 0.002$ を仮定)

ここまでの議論を整理すると, 「サンプル値系における安全性保証」とは「1. 入力信号の安全性, 2. サンプル間隔の安全性」を保証することになる. これらが行われることを回避する学習方法については今後検討する.

4.3　セルフトリガー制御の強化学習

2.4 節で記した初期方策 π_{init} から, 安全性を保証しながら最適セルフトリガー制御則 π^* の学習を試みる. サンプル値系とは異なり, インタラクション間隔はエージェントが決定する. したがって, 方策関数の出力 $\pi(s) = \begin{bmatrix} a & \tau \end{bmatrix}$ に対して, $\tau > \tau_{\max}(a)$ となった時には入力信号 a を変更する方法と, 通信間隔 τ を変更する方法の二種類の選択があり, どちらを行うべきか考える必要がある.

5　(先行して) セルフトリガー制御の強化学習の実験

現状, ECBF についてまだ深い考察と実装を行っていないので, セルフトリガー制御の安全性の確保は行うことはできない. しかしながら π_{init} から π^* を強化学習するための環境は整っているため, 安全性については考慮せずに実験を行ってみた.

実験環境は Open-AI Gym の pendulum で, 初期状態が $\theta \sim N(0, 0.1), \dot{\theta} \sim U(-1, 1)$ となるような環境で行った. また, 10 秒間の制御を 1 エピソードと定義する.

5.1　λ による学習過程の変化

式 (7) より, τ を大きくするモチベーションは λ に大きく依存するため, それを様々な値にすることでど学習過程にどのような変化が見られるのかを確認してみた. ここでは, λ を $0.05, 0.5, 5, 50$ に設定し, それぞれ 3 回, 200000 ステップ (インタラクションの回数) の強化学習を行った. また, τ は $0.001 \sim 0.1$ の範囲に制限している.

まずそれぞれの学習タスクの中で, 1 エピソードの間の τ の平均値の変化を見てみる.

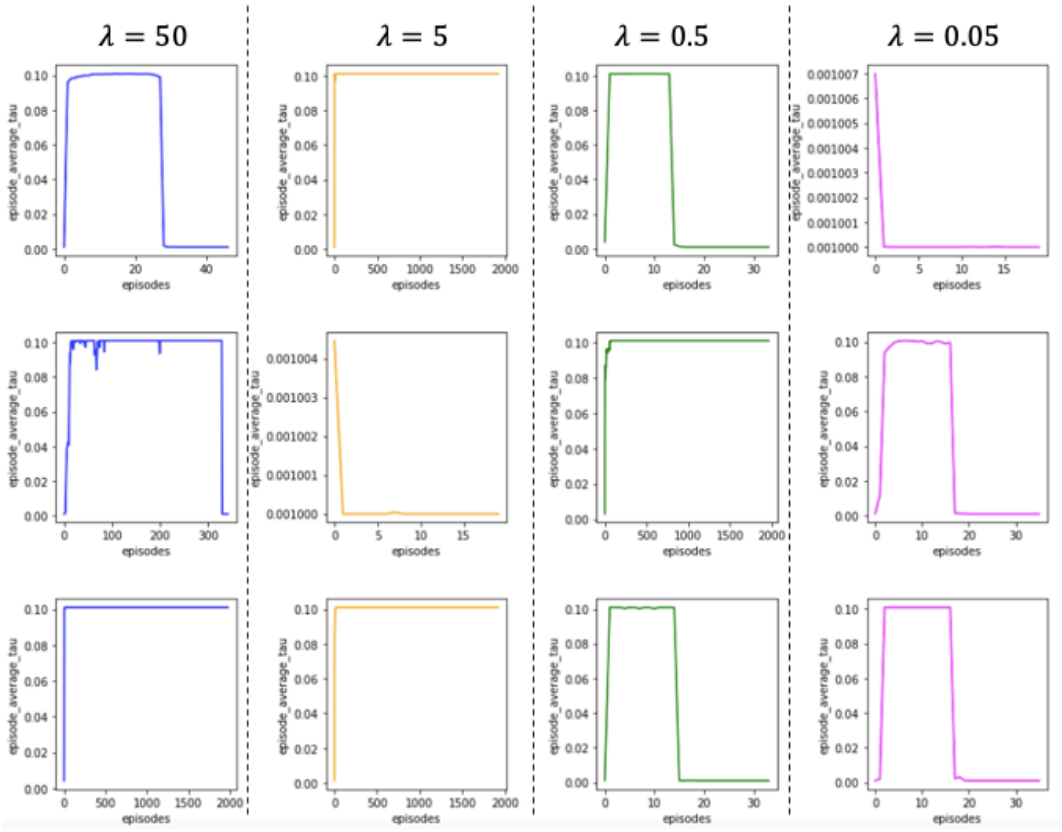

図 3　学習を通しての τ の変化

ここで注意したいのは, τ によって 200000 回のインタラクションに含まれるエピソード数が異なる点である. 例えば τ の学習が進まずにずっと $\tau = 0.001$ であった場合, 10 秒間の制御の間には 10000 回のインタラクションが行われる. したがって 200000 回のインタラクションの間には 20 エピソードしか含まれない. 逆に, 初期から τ の学習が進み $\tau = 0.1$ となることが増えてくると, エピソード数は増えてくる. 図 3 の x 軸の範囲が異なるのはそのためである.

次に, 制御性能 (倒立振子の制御) の変化を見るためこの時のそれぞれのエピソード報酬の変化を見てみる.

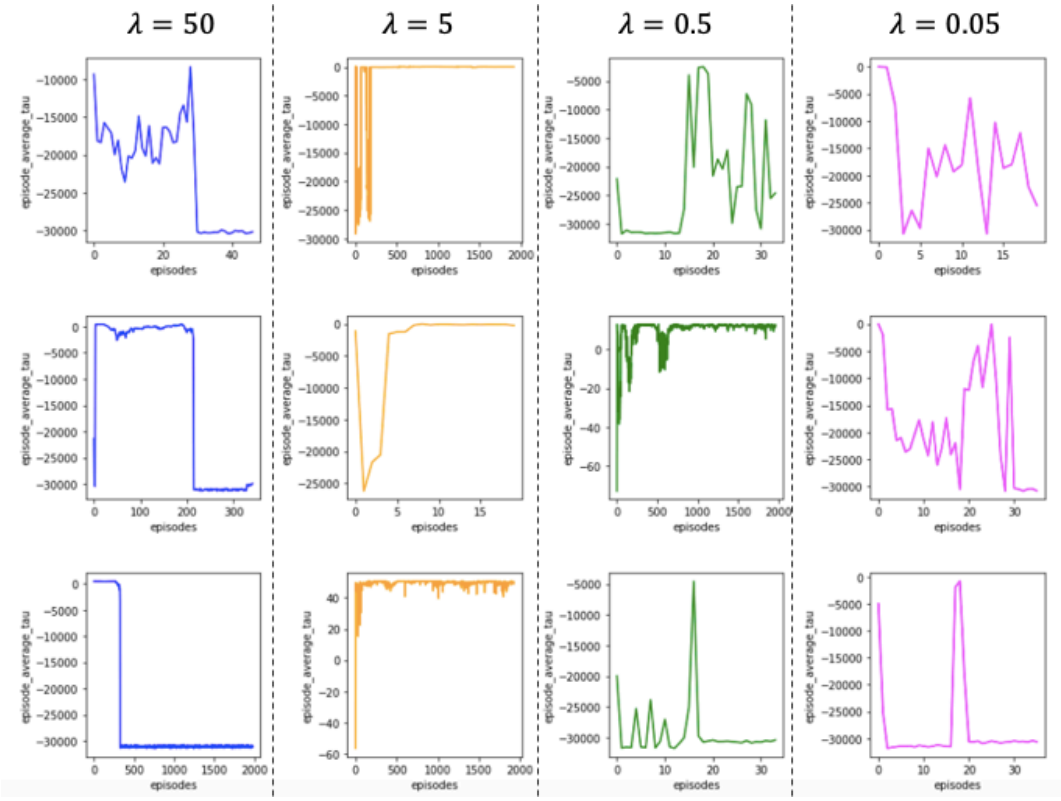

図 4　学習を通してのエピソード報酬の変化

5.2　考察

図 4 の各ブロックが表す学習の記録は図 3 と対応している．これら 2 つの図を見比べてみるといくつかの点に気付く．

1. $\lambda = 0.1$ を初期に学習できていないと, 最終的に得られる制御性能自体も悪化する.
2. λ が大きい方が τ を大きくする方向に学習が進みやすい (当然) が, 毎回そうなる訳でもない点
3. $\lambda = 0.1$ を学習できていても, $\lambda = 0.001$ に逆戻りしてしまうことがある.

1. について,reward の改善が見られないのはエピソード数が少ないためであることは予想がつくが, π_{init} は $\tau = 0.001$ において, 原点付近で線形化したシステムを安定化する方策であるため, reward が減っていくことは腑に落ちない.
3. については, reward をみると $\tau = 0.1$ を維持している学習とそうでない学習で, $\tau = 0.1$ の間の制御性能に差がある点ことがわかり, これが理由だと考える. ただし, $\lambda = 50$ の学習での逆行は理由がわからない.

5.3　学習した方策による制御性能

図 4 の 2 行 3 列目が表す学習によって得られた方策 π_a の制御を詳しくみてみる.

図 5 を見ると, おおよそ振り子を倒立させ続けられていることがわかる. しかし注目したいのは τ の変化である. 本来セルフトリガー制御は状態 s によって様々な値をとるはずであるが, ほぼ一定値となってしまっている. このことから「倒立時は長くインターバルを空け, 倒れ出した時に多くインタラクションを行う」というほども高度なことを学習できていない. また, これは果たしてリーズナブルな結果なのかも今後明らかにして行きたい.

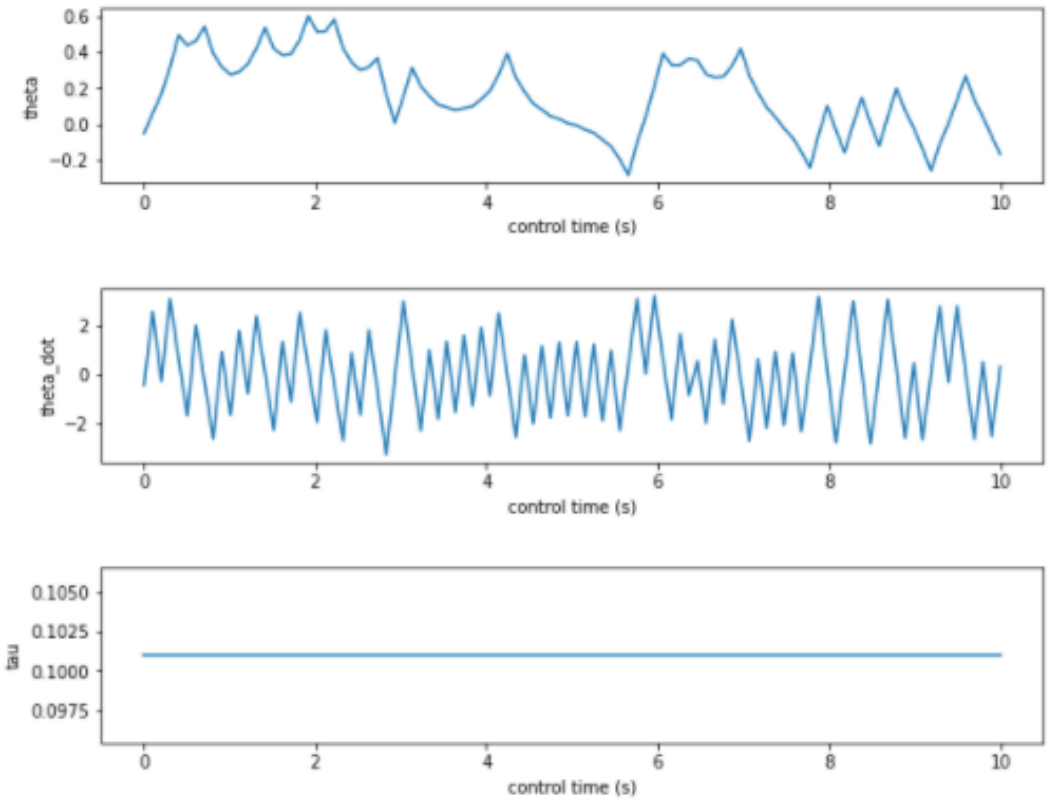

図 5　π_a による制御ログ

6　$\tau(s)$ の学習の様子

前節で, エピソード (10 秒間の制御) 毎にその間の $\tau(s)$ の平均値を追うと, 0.001 から 0.1 に急激に変化していることを確認した. また, 5.3 節では, 200000 ステップの学習によって得られた方策によって決定される $\tau(s)$ がインターバルの間ずっと 0.1 となっていることも確認した.

そこで, 本節では $\tau(s)$ を正しく s 依存の関数として扱っていて, 学習の結果このように「張り付く」ような結果になっているのか否かを確認する.

6.1　検証方法

方策関数 $\pi_\theta(s)$ のパラメータ θ は 1 ステップ (1 インタラクション) ごとに行われる. したがってその 1 要素である $\tau(s)$ という関数も 1 ステップごとに更新される. ここでは学習途中のとある 100 ステップ分の $\tau(s)$ を取り出し, それぞれの s に対する $\tau(s)$ の変化を確認する. (ただしプロットのための引数として用いる s は, $\theta \in [-\pi, \pi], \dot{\theta} \in [-8, 8]$ の範囲からランダムに 500 個選ぶ.)

6.2　$\tau(s)$ の変化

まず初めに π_{init} による $\tau(s)$ をみてみる.

図 6 は, 全状態で $\tau(s) = 0.001$ となっており, 式 (10) の通りである.

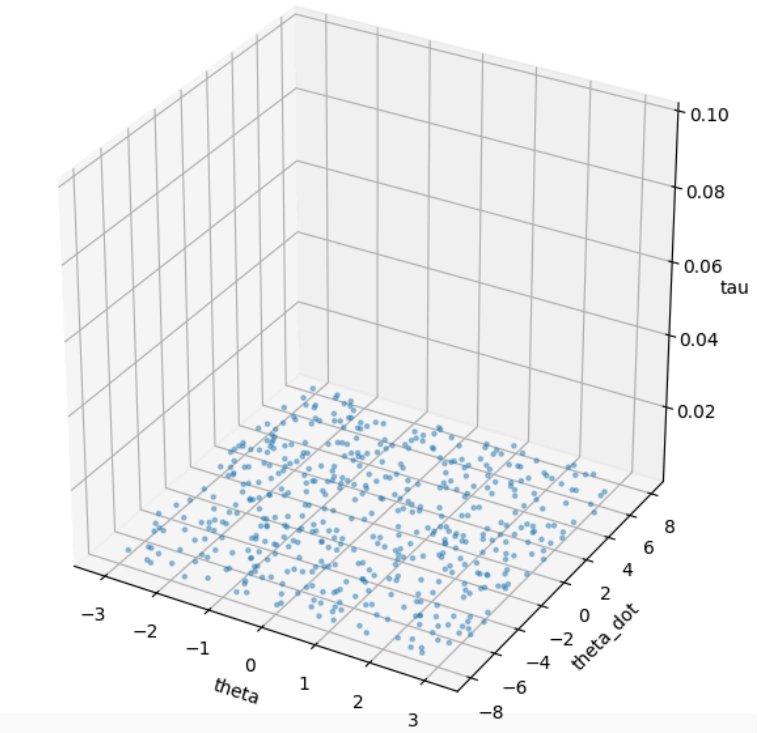

図 6　π_{init} の $\tau(s)$

では次に 5.3 節で確認した方策 π_a による $\tau(s)$ をみてみる.

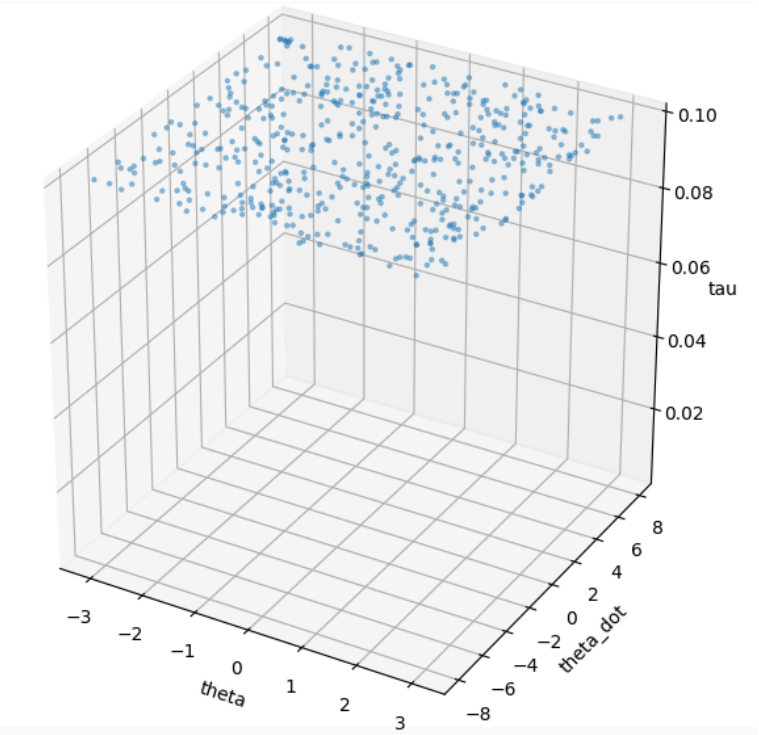

図 7　π_a の $\tau(s)$

図 7 をみると, ほとんど全ての状態で $\pi_a(s) = 0.1$ となっていることがわかる.

では, これらの途中にはどのような変化があったのかを確認する.

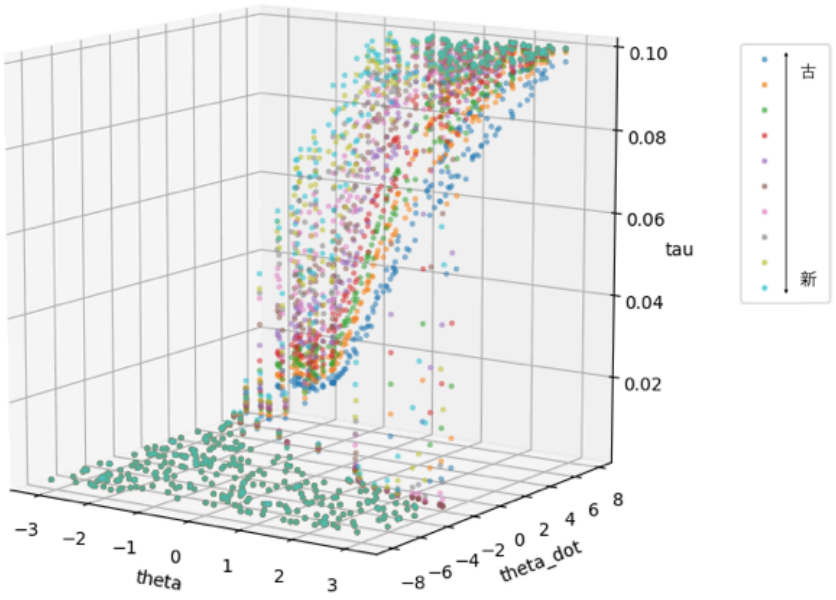

図 8　学習が進むにつれてどのように $\tau(s)$ が変わるのか

図 8 は 6.1 節で述べた, 学習中のとある 100 ステップ分の関数 $\tau(s)$ のうち, 等間隔に 10 個選んだものをプロットしたものである. 図の同じ色の点は同じ関数 $\tau(s)$ による出力を表す. 図 8 をみると, 少しづつ $\tau(s) = 0.1$ となる領域が大きくなっていることがわかる.

6.3　$\tau(s)$ の学習過程のまとめ

ここまでみてきたように, $\tau(s)$ は s 依存の関数として学習しているにもかかわらず全状態で 0.1 を出力する関数と 0.001 を出力する関数とを往復しているようである. (図 9 は逆に一度全状態で 0.1 を出力する関数になった後, 再び 0.001 に戻っていく様子である.)

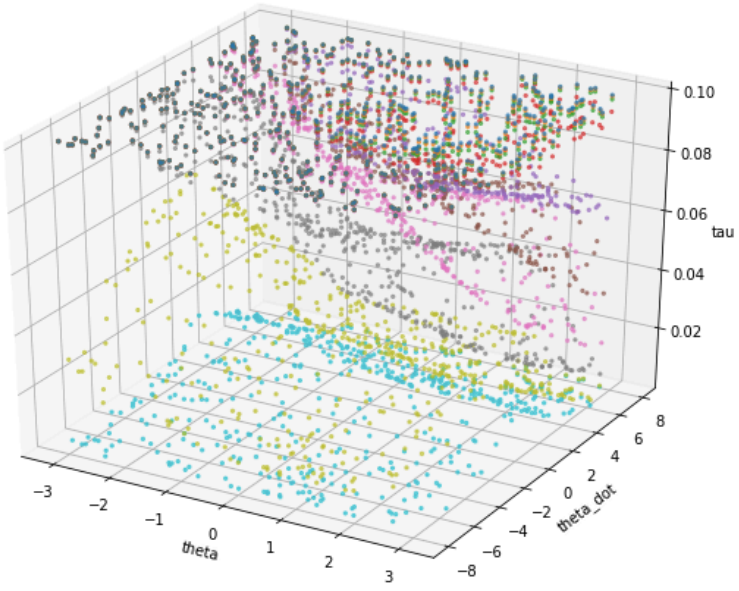

図 9　$\tau(s)$ の変化

本研究の目的は, 「状態 s によって最適な τ を出力するような方策の学習」である. したがって現状得られているような $\tau(s)$ では目的を果たせているとは言えない (原点を最大値に持つ凸関数たるべきである). この課題が生じている原因として, パラメータ更新に用いる勾配法の学習率が大きすぎるというものが考えられる.

しかしながら $\tau(s)$ は図 10 のように $\pi_\theta(s)$ を表現するニューラルネットワークの出力の 1 つとして与えているので, $\tau(s)$ の学習率のみを小さくすることはできない (入力信号 $a(s)$ の学習も遅くなってしまう).

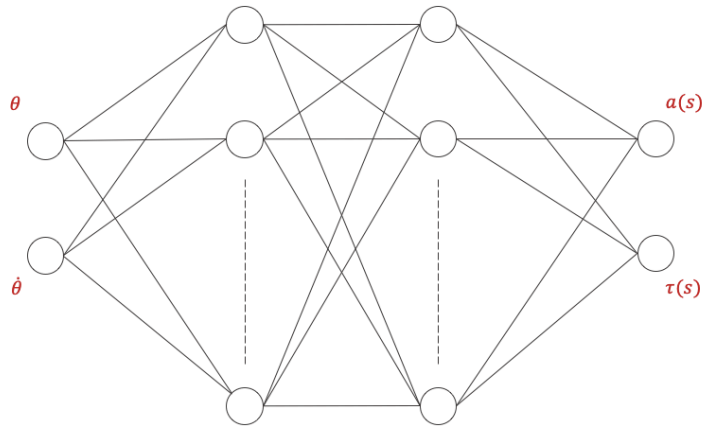

図 10　方策 $\pi_\theta(s)$

そこで図 11 のようにニューラルネットワークを分離して, $a(s)$ と$\tau(s)$ をそれぞれ別のニューラルネットワークで表現してみようと考えている.

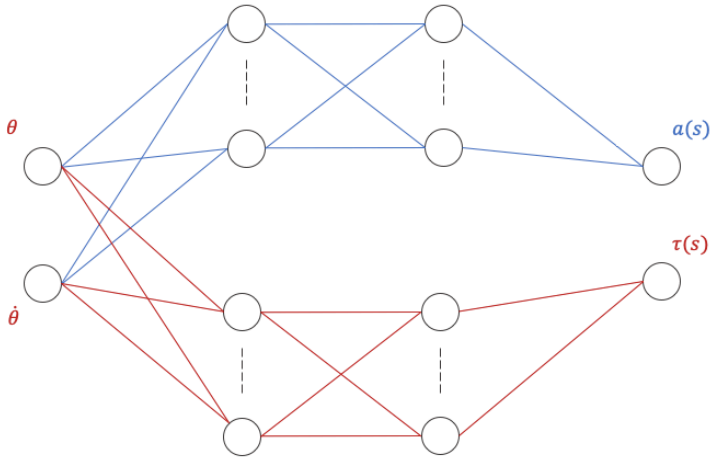

図 11　方策 $\pi_\theta(s)$ の分離

こうすることで, 入力信号 $a(s)$ とインタラクション間隔 $\tau(s)$ の学習率を別々にすることができる.

ただし, このように「張り付く」関数になってしまうのは学習率以外の原因が大きい気がするので究明したい (試行錯誤の回数が少なすぎる (?)). また, 研究の効率化の点からもバリア関数のロジックの組み込みについても真剣に取り組み始めたい.

参考文献

[1] G. Yang, C. Belta, and R. Tron. “Self-triggered Control for Safety Critical Systems Using Control Barrier Functions.” *In American Control Conference (ACC) Philadelphia, USA*, 2019.