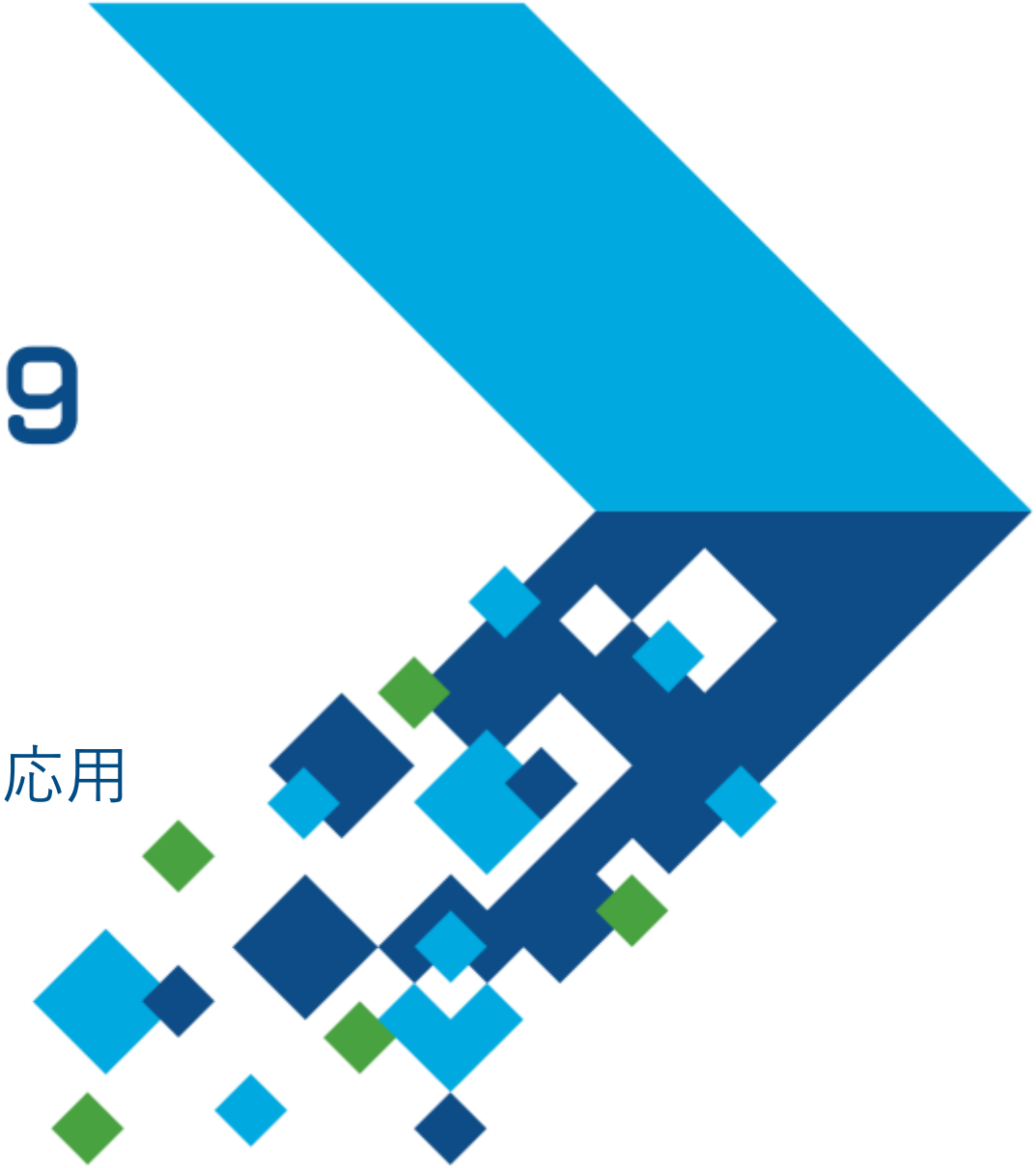
MATLAB EXPO 2019

強化学習

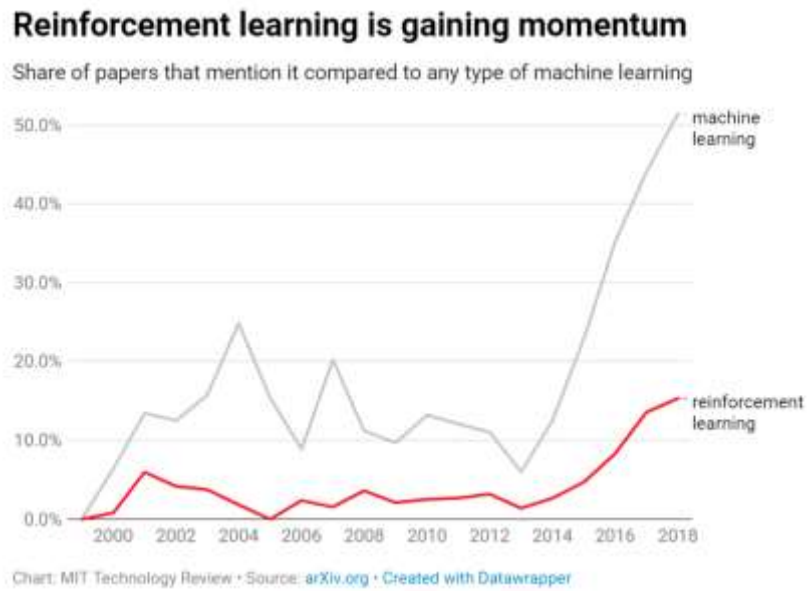
最適制御のためのディープラーニングの応用

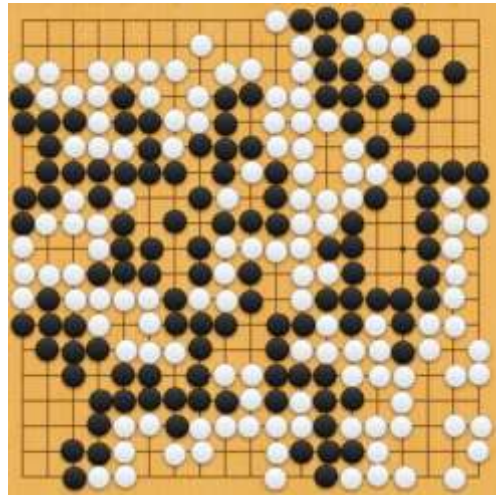
吉田 剛士



はじめに

強化学習 = Reinforcement Learning





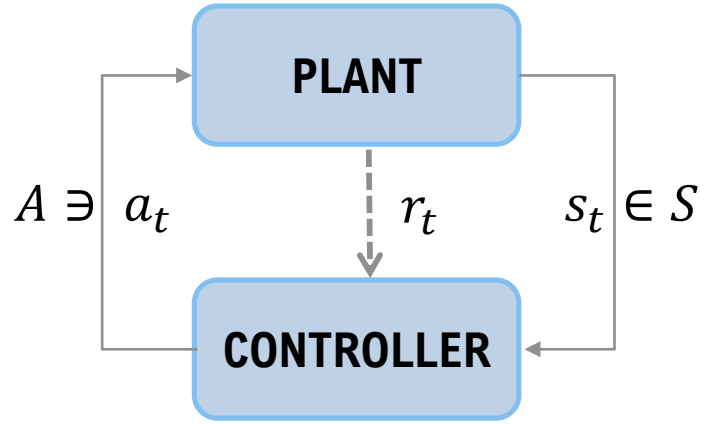
AlphaGo がプロ棋士に勝利 (2015)
そして、人類を超える (2017)

- 強化学習の特徴
  - 自律的に学習し賢くなっていく

Agenda :
強化学習 ~ 最適制御のためのディープラーニングの応用 ~

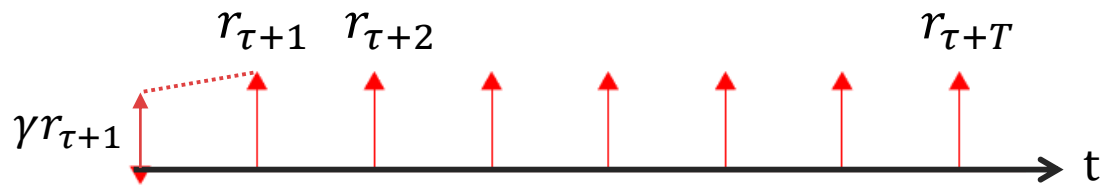
- 強化学習とは
- MATLAB による強化学習
- まとめ

強化学習とは?



- 最適制御問題の解を求めるためのアルゴリズムの１つ
  - 収益(累積報酬)の期待値の最大化

$$\mathrm{E}\left[\sum_t \gamma^t r(s_t, a_t, s_{t+1})\right]$$

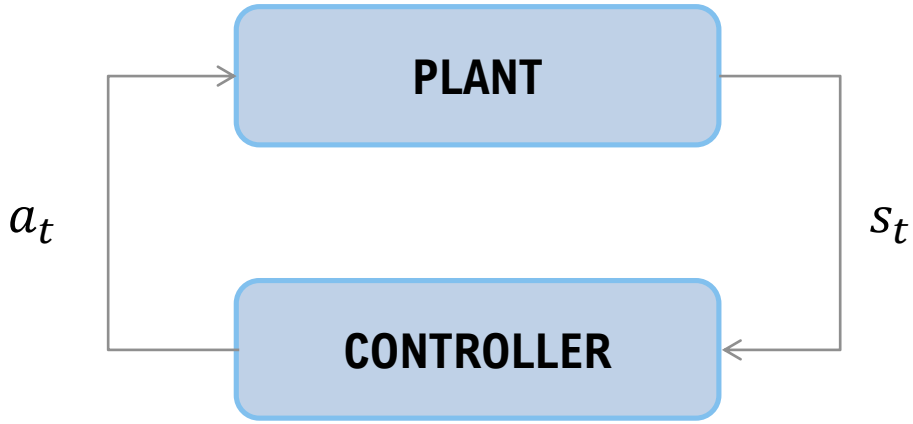


  - 適用条件
    - 制御対象(環境/プラント)の出力である状態 s_t が、マルコフ決定過程 (MDP) で表現できること
      Markov Decision Process
      - » 状態遷移に関する条件で、時刻 t において状態 s の場合に行動 a を選択した時、次の時刻において状態 s' となる確率は時刻 $t-1$ 以前の状態および行動には依存しない、つまり次の関係が成立する

$$\Pr(s_{t+1} = s' | s_t = s, a_t = a) = \Pr(s_{t+1} = s' | s_t = s, a_t = a, s_{t-1}, a_{t-1}, \cdots s_0, a_0)$$

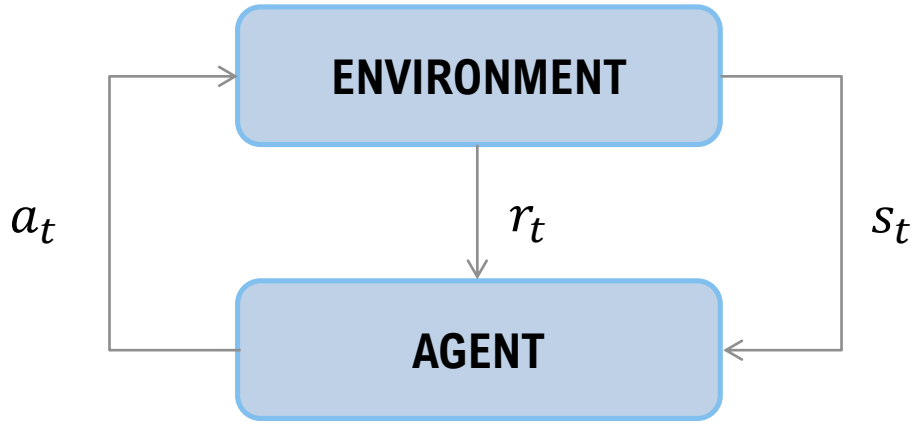
閉ループ制御 vs. 強化学習

- 閉ループ制御



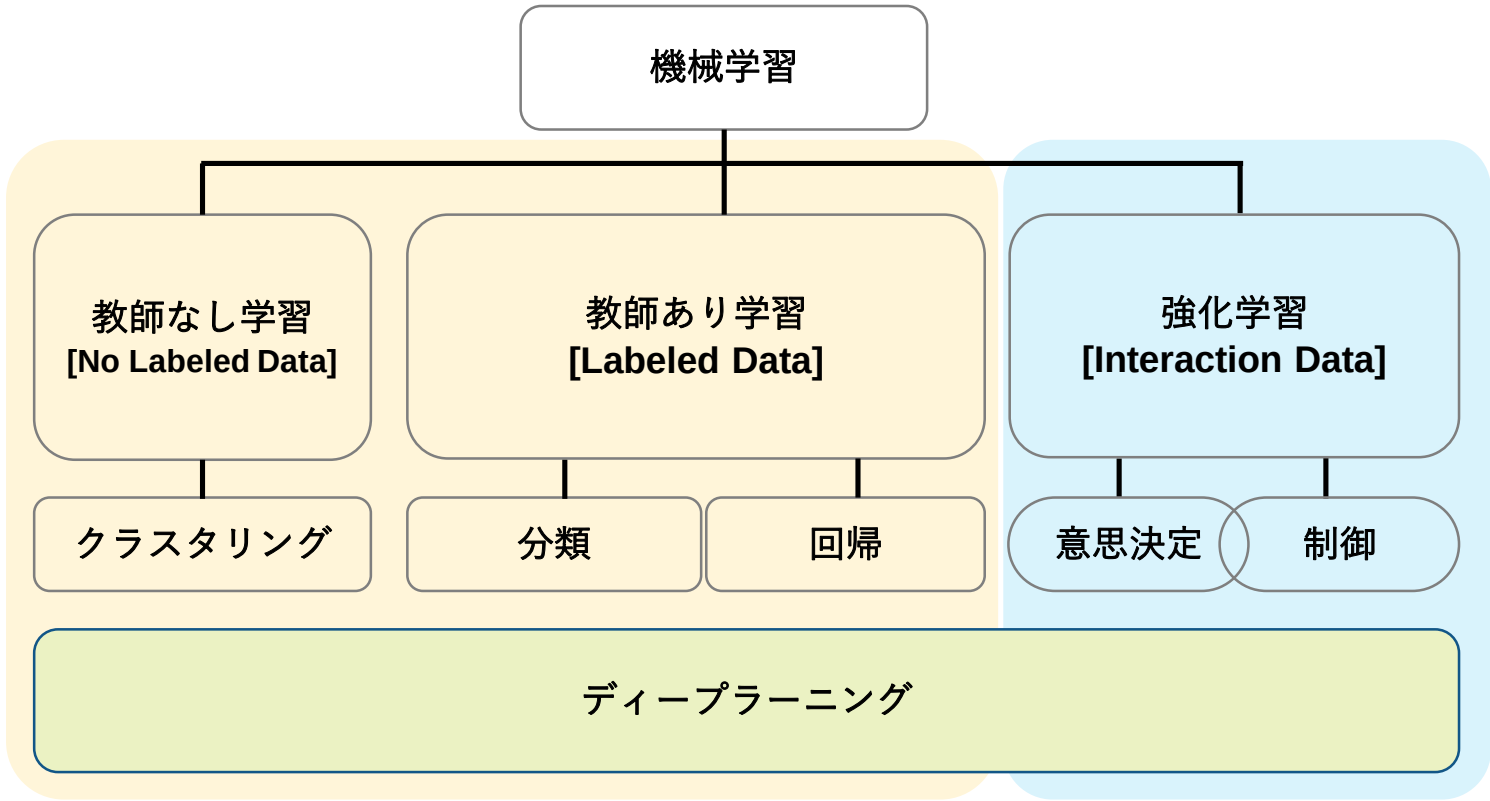
前ページに登場した収益の期待値を最大化するような方策関数 π の探索が目的

- 強化学習



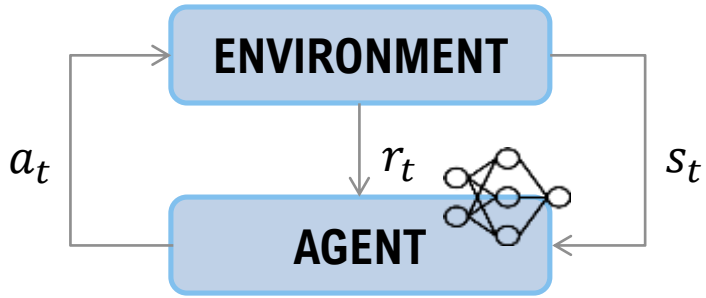
  - 時々刻々「報酬 (r_t)」を環境から受け取る
    - $r_t \coloneqq r(s_t = s, a_t = a, s_{t+1} = s')$
  - 状態 s_t は、マルコフ決定過程
  - 状態の確率変数列 $\{s_t\}_t$ は独立同分布を持つ
  - 行動の確率変数列 $\{a_t\}_t$ も独立同分布を持つ
    - 方策関数 π : 状態が s の時に、行動 a を選択する確率
      $\pi(a|s) \coloneqq \Pr(a|s)$

強化学習と、機械学習/ディープラーニングの関係

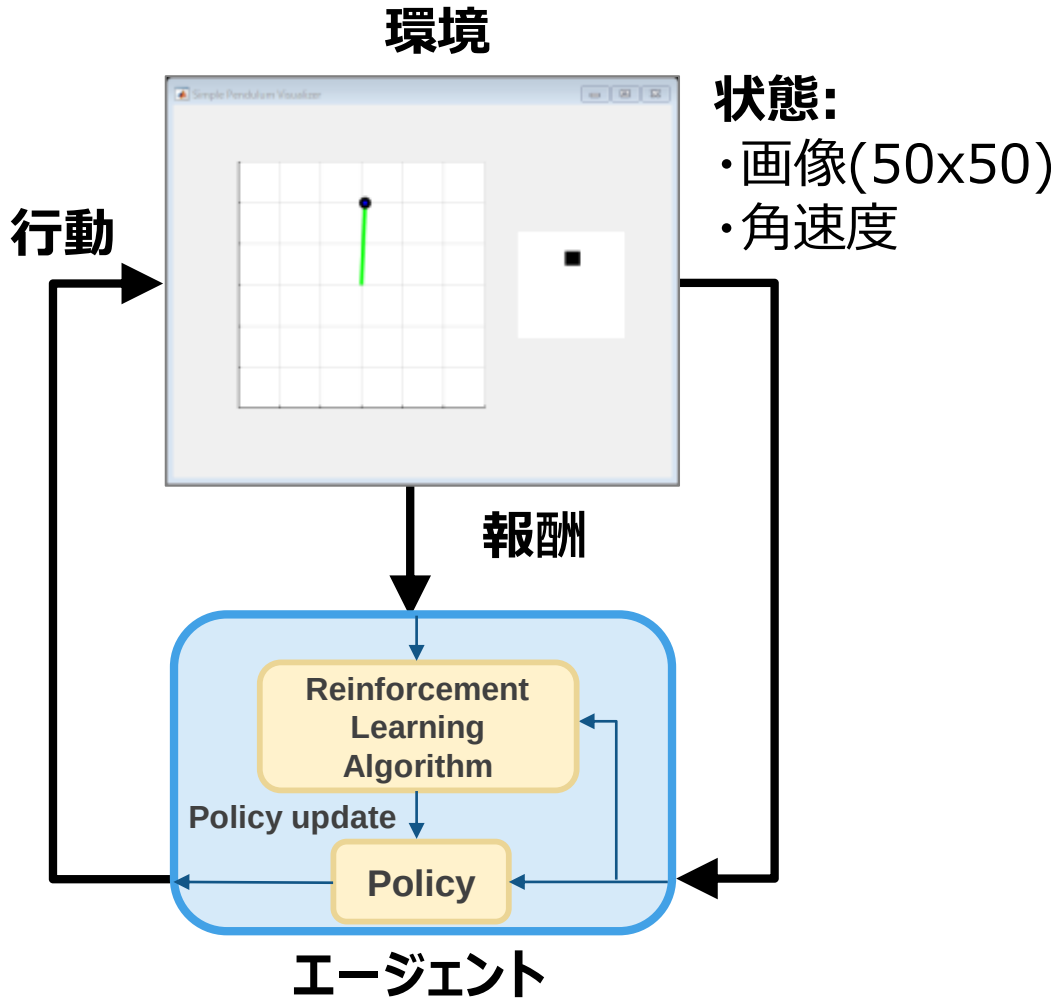


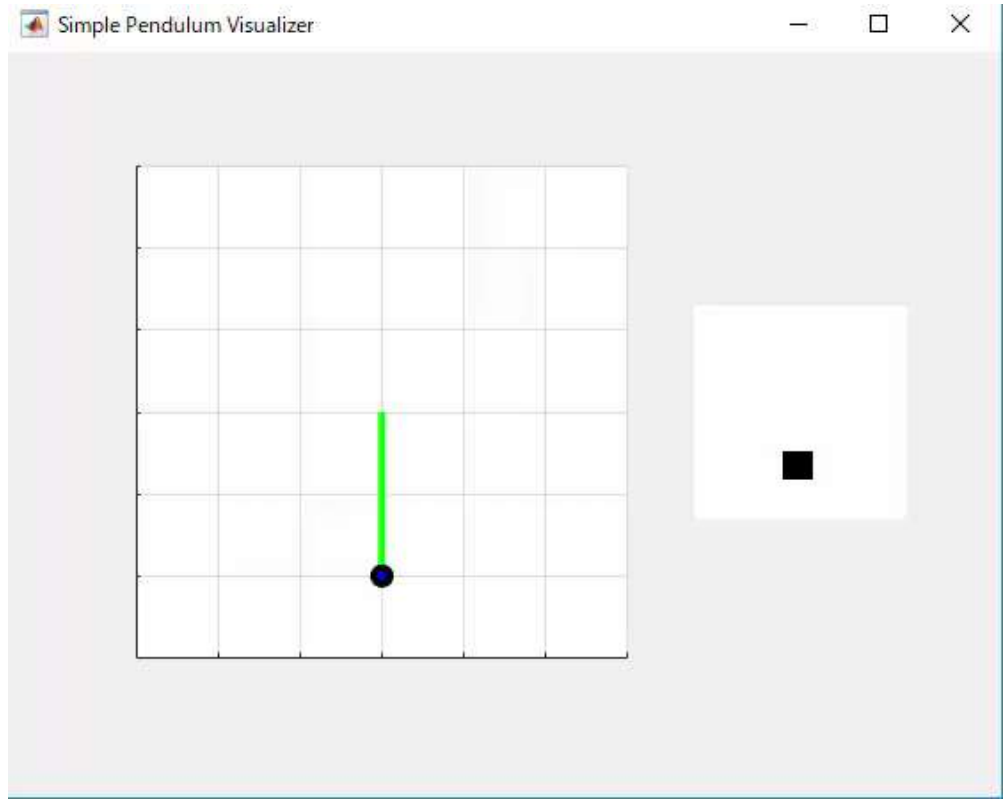
強化学習:

- 試行の繰り返しによりデータを集め収集されたデータを用いて学習を進め [*interaction*]、振る舞いの習得、またはタスクの完遂を目指す
- 複雑なシステムの制御にはディープなニューラルネットワークが活用される [*Deep Reinforcement Learning*]



従来手法とは異なるアプローチ
画像を状態量とした例





適用検討分野

自動運転

- クルーズコントロール
- レーンキープアシストシステム
- 障害物回避
- パスプランニング
- 最適ドライバーモデルの構築

FA/ロボットアーム

- バラ積み把持点の自動検出
- 嵌合作業
- 自己/複数台干渉回避

FA/AGV/フィールドロボット

- 障害物回避
- 悪路での直走性獲得

メディカル

- X線照射装置

エネルギー

- 最適運用計画

航空宇宙/UAV

- 飛行経路プランニング

通信

- 動的チャネル最適化

土木建築

- エレベーターの最適運用
- 自動建築システム

経済

- 景気政策決定

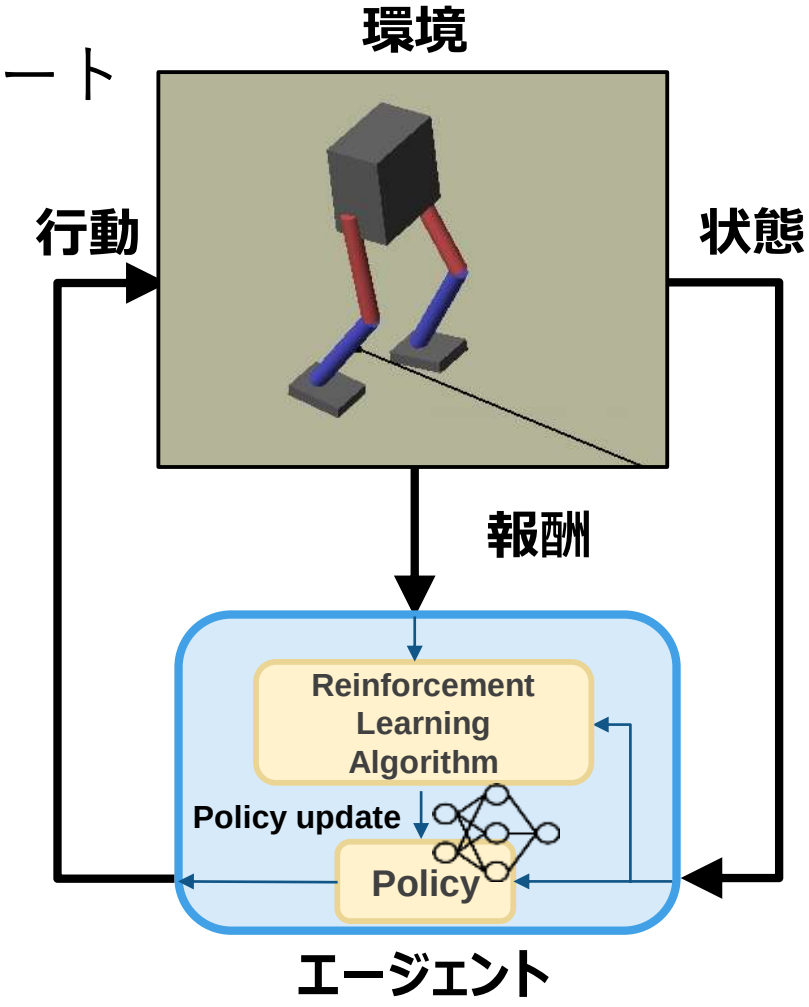
その他

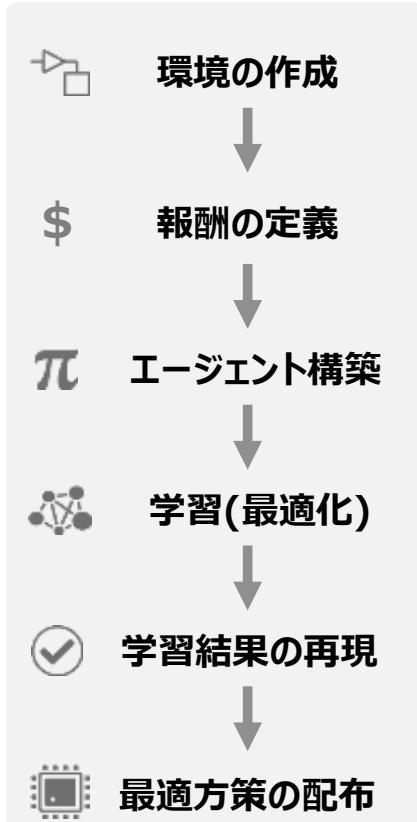
- 交通渋滞回避のための信号機制御
- 設備のリソーススケジューリング

Reinforcement Learning Toolbox™

R2019a

- 強化学習のフローを網羅的にサポート
  - MATLAB 関数 / Simulink® モデルによる環境とのインターフェース
    - 強化学習用「RL Agent」ブロック
  - エージェント作成のための ネットワーク構築環境
  - 学習アルゴリズム
    - Q-Learning
    - DQN / Double DQN
    - SARSA
    - REINFORCE*1
    - DDPG
    - A2C*1,*2
    - 並列学習 (GORILA / A3C*1,*2)
  - 配布のための最適方策の関数化



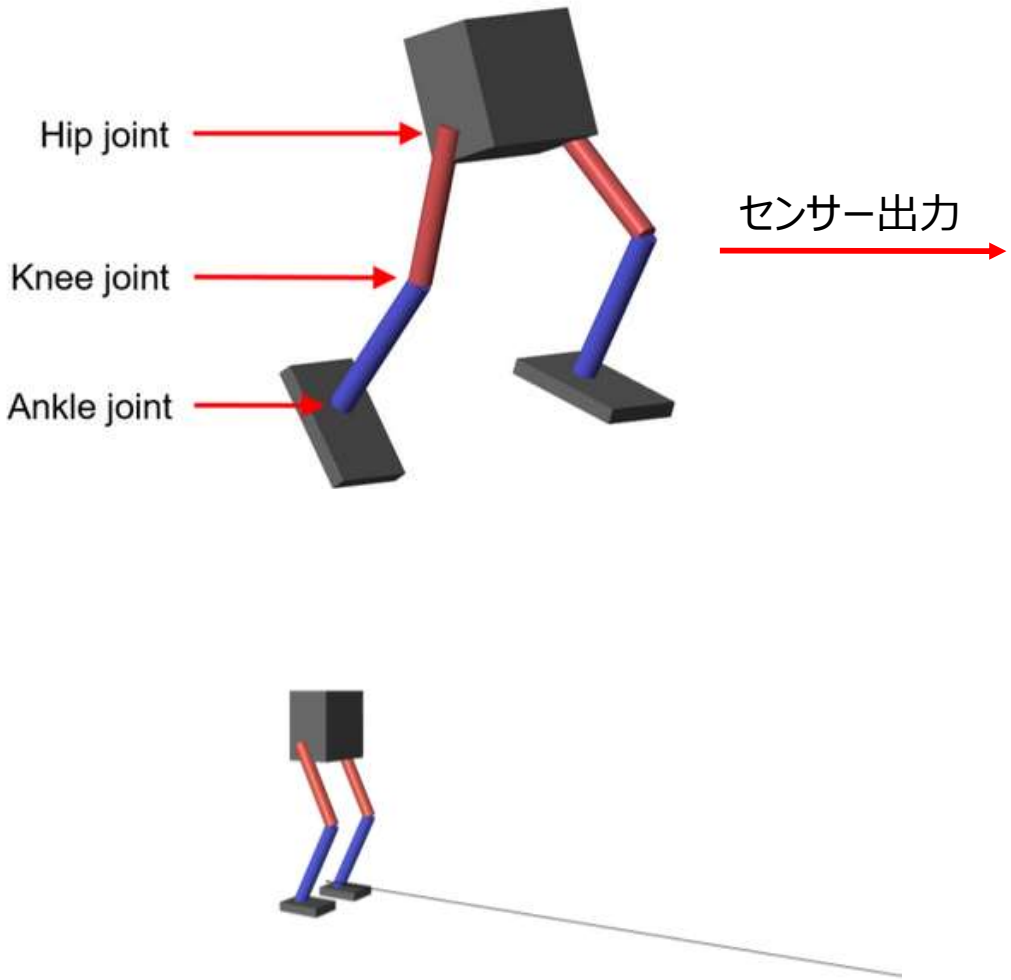


R2019a時点では
*1. 行動空間が離散であることを要求します
*2. 1 つのネットワークで状態価値関数および方策関数を表現することができません

Agenda
強化学習 ~ 最適制御のためのディープラーニングの応用 ~

- 強化学習とは
- MATLAB による強化学習
- まとめ

例題：２足歩行ロボットの歩行動作獲得

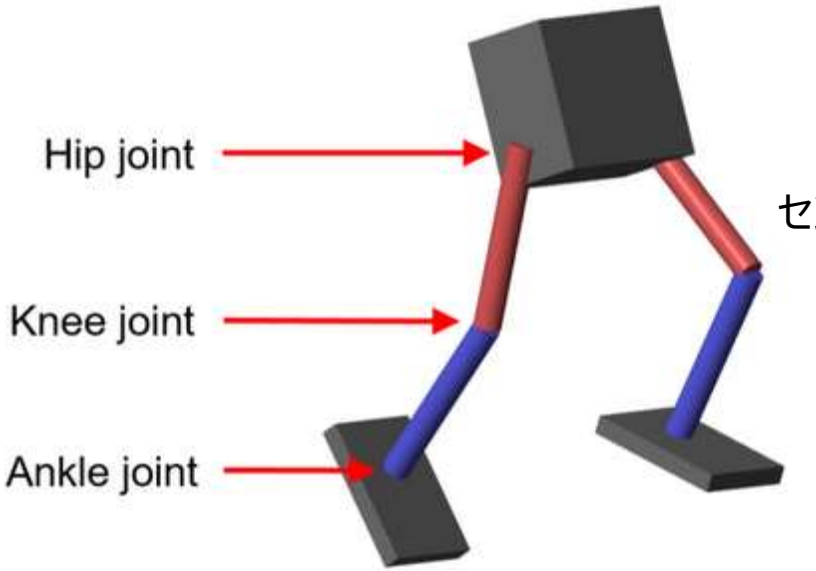


- 制御入力 (Num of Input: 6)
  - 足首、膝、付け根 x 2足=6関節トルク
  - それぞれ、-3[Nm]から3[Nm]の連続値
- センサー出力 (Num of output :23)
  - 胴体の姿勢(11個)
    - 重心位置Y(横方向)とZ(高さ)
    - 重心速度(X(進行方向),Y(横方向),Z(高さ))
    - ヨー角、ピッチ角、ロール角
    - ヨー角速度、ピッチ角速度、ロール角速度
  - 両足の関節(12個):
    - 両足の3つの関節の角度と角速度

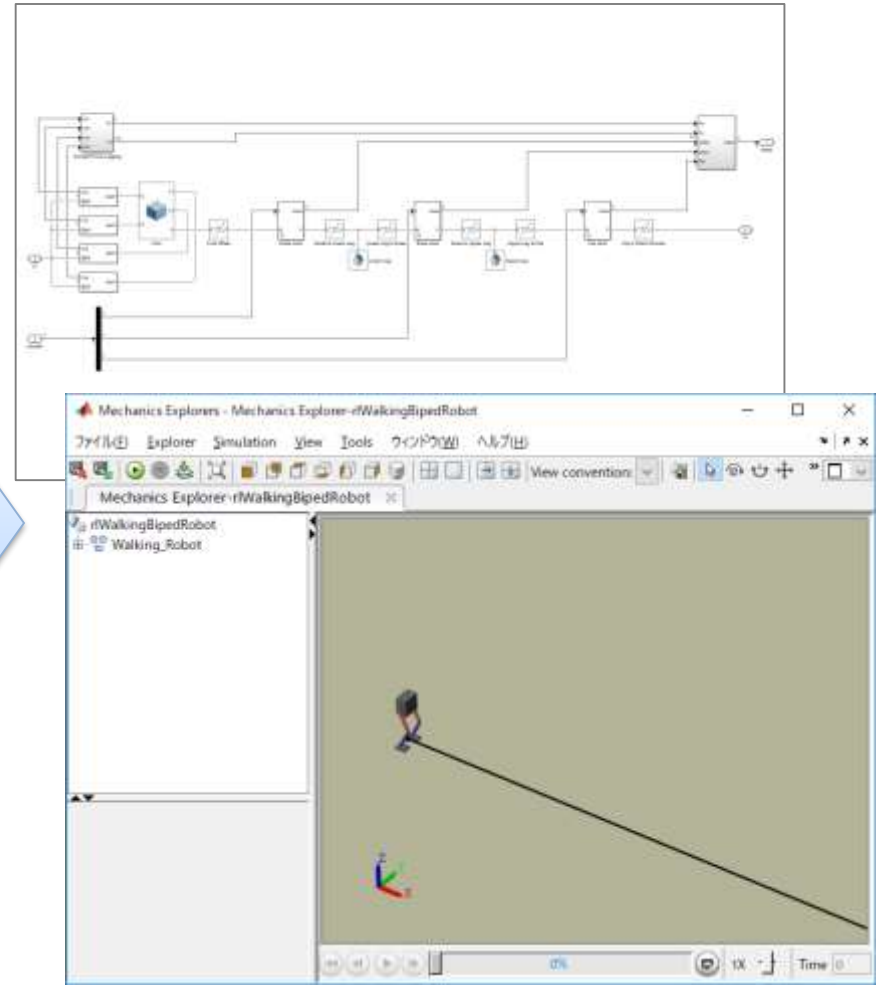
ロボットが直線上を歩くように制御するには？

1. 環境の作成

- 制御対象であるロボットを含む「環境」のモデル化



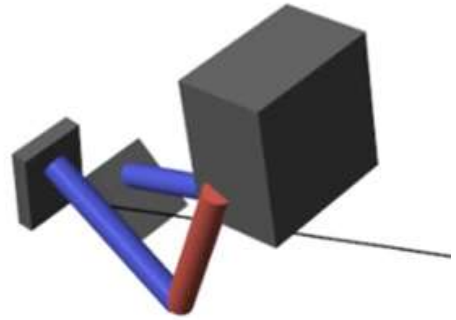
センサー出力



環境の作成
↓
報酬の定義
↓
エージェント構築
↓
学習(最適化)
↓
学習結果の再現
↓
最適方策の配布

環境に「失敗」を設計

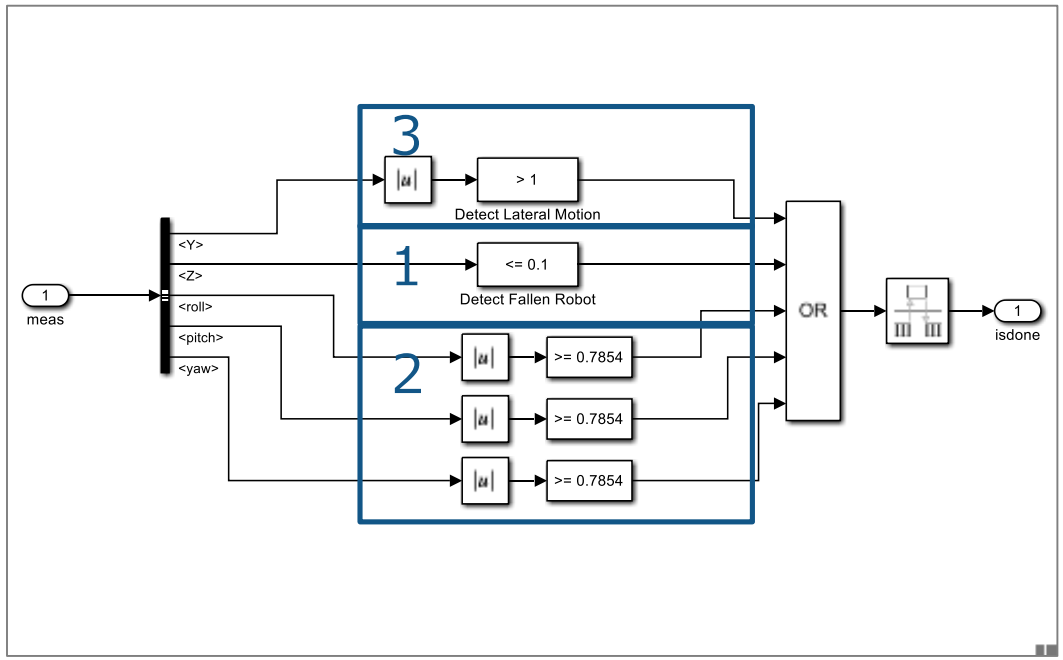
シミュレーション早期終了条件の定義



1. 転倒
   - 胴体の高さ(Z)が0.1m以下になる

2. 転倒の予兆で復帰不能
   - 胴体のロール角、ピッチ角、ヨー角が45度以上になる

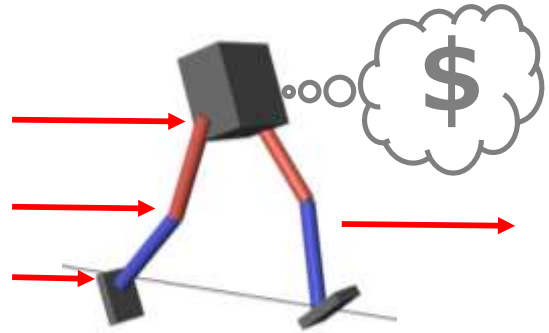
3. 軌道から大幅にずれる
   - 胴体の横方向位置(Y)が1m以上直線から離れる



- 環境の作成
- 報酬の定義
- エージェント構築
- 学習(最適化)
- 学習結果の再現
- 最適方策の配布

2. 報酬の設計

4. 「力の大きさ」を減点
各トルク値の２乗和
を罰則として採用



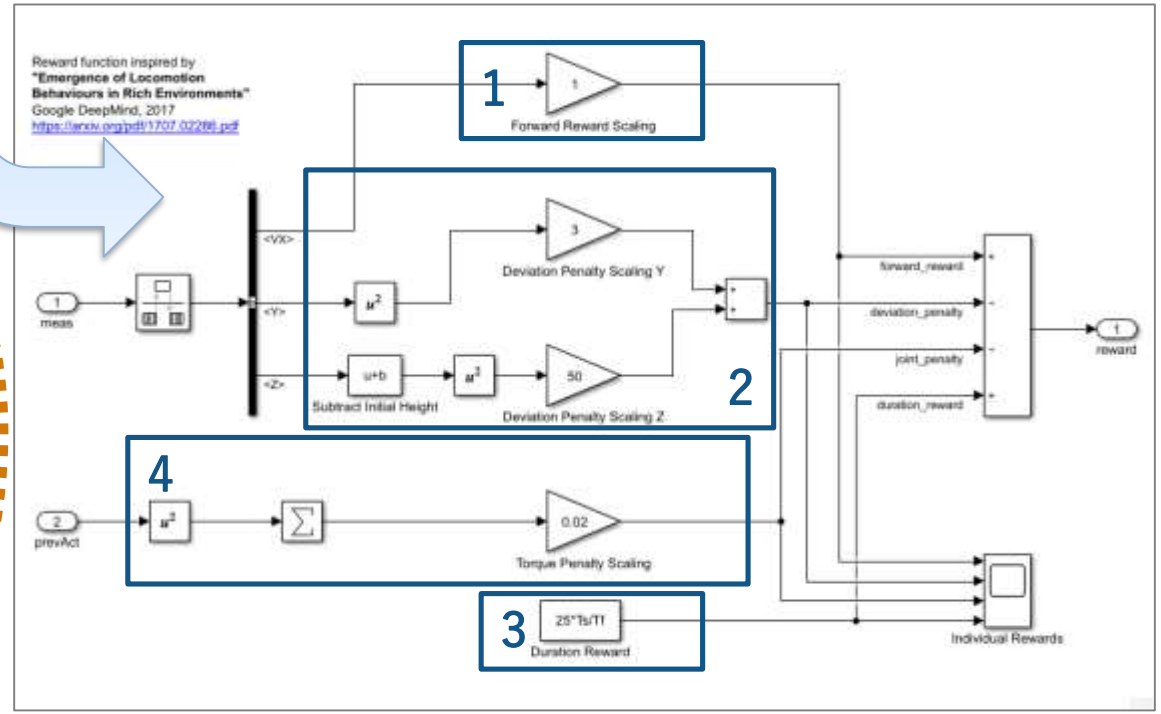
- 報酬関数の定義

$$r_t := v_t - \{3y^2 + 50(z - z_0)^2\} + 25\frac{Ts}{Tf} - \frac{1}{50}\sum_{i}^{6} {u_{i,t-1}}^2$$

1. 「前進」を加点
前進速度を報酬として採用

2. 「不安定さ」を減点
Y方向およびZ方向の初期値
からの差を罰則として採用

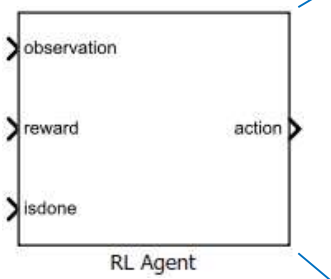
3. 「正規化したΔt」を加点
シミュレーション時間を報
酬として採用

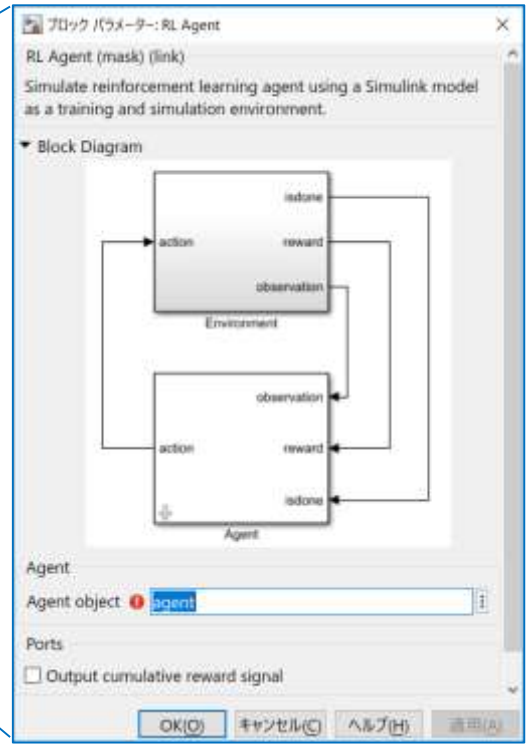


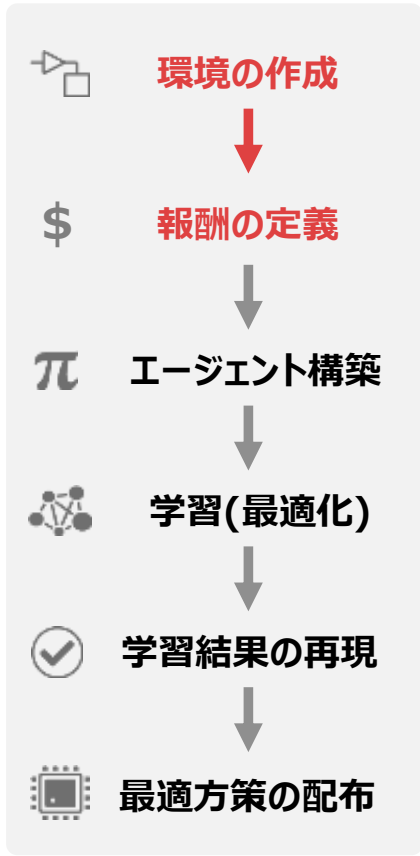
環境の作成
↓
報酬の定義
↓
エージェント構築
↓
学習(最適化)
↓
学習結果の再現
↓
最適方策の配布

RL Agent ブロックの配置

- RL Agent block
  - Reinforcement Learning Toolbox が用意する
    環境インターフェース向け Simulink モデル向けのブロック
    - 「報酬」と「(シミュレーションの)終了フラグ」をモデルに配置



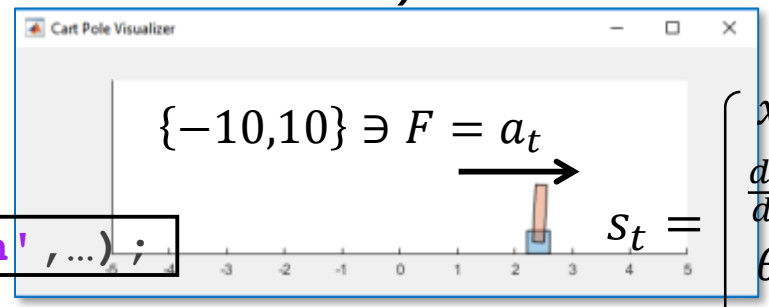





```
env = rlSimulinkEnv(mdl, blk, ...);
```

環境および報酬の用意
MATLABコードの場合

- Δt ごとの環境の挙動(時刻 t から時刻 t+1 への環境の更新)をコード化
  - 例 : Cart Pole の場合




```
env = rlFunctionEnv(...,'myStepFunction',...);
```

```
function [NextObs, Reward, IsDone, LoggedSignals] = ...
        myStepFunction(Action,LoggedSignals,EnvConstants)

Force = Action;     a_t

% Unpack the state vector from the logged signals
State = LoggedSignals.State;
XDot = State(2);
        :
```

s_t

環境の作成
↓
報酬の定義
↓
エージェント構築
↓
学習(最適化)
↓
学習結果の再現
↓
最適方策の配布

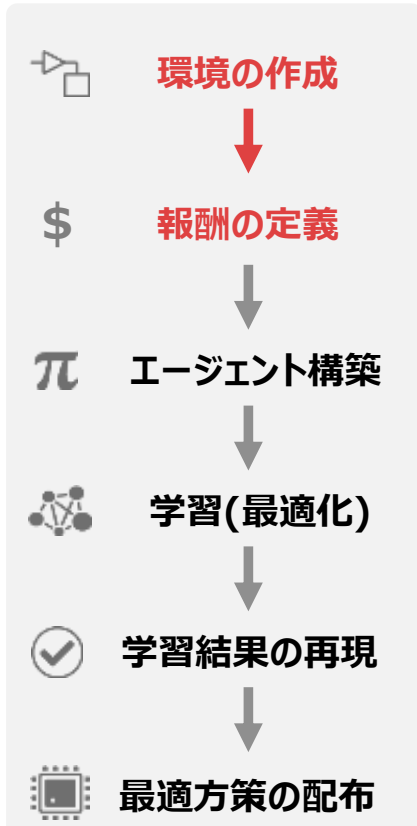
環境および報酬の用意 (続き)
MATLABコードの場合

```
% Euler integration
LoggedSignals.State = State + ...
 EnvConstants.Ts.*[XDot;XDotDot;ThetaDot;ThetaDotDot];

% Transform state to observation
NextObs = LoggedSignals.State;

% Check terminal condition
X = NextObs(1); Theta = NextObs(3);
IsDone = abs(X) > EnvConstants.XThreshold || ...
abs(Theta) > EnvConstants.ThetaThresholdRadians;

% Get reward
if ~IsDone
    Reward = 1;
else
    Reward = -5;
end
```

$$s_{t+1} = s_t + \Delta t \cdot \begin{pmatrix} \frac{dx}{dt} \\ \frac{d^2x}{dt^2} \\ \frac{d\theta}{dt} \\ \frac{d^2\theta}{dt^2} \end{pmatrix}$$

シミュレーション(早期)終了条件

r_t



3. エージェント構築

1. アルゴリズムの選択
   - R2019a でサポートされるアルゴリズム

Q-Learning
rlQAgent

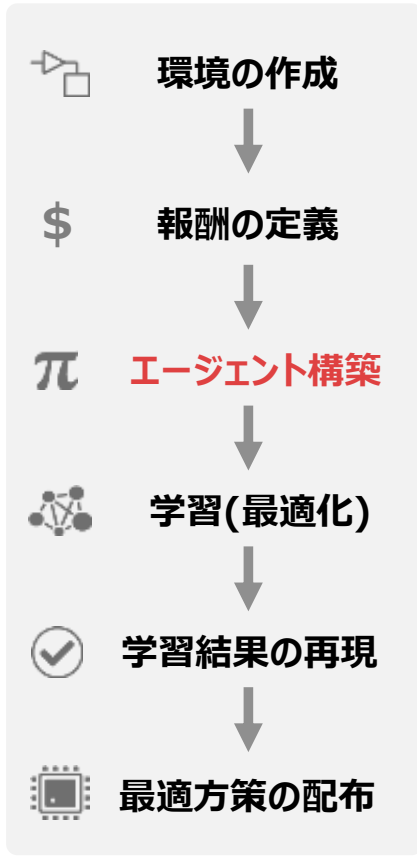
DQN
Double DQN
rlDQNAgent

A2C /A3C
rlACAgent

REINFORCE
(Policy Gradients)
rlPGAgent

DDPG
rlDDPGAgent

SARSA
rlSARSAAgent



R2019a時点では
※1. `rlPGAgent` および `rlACAgent` は行動空間が離散であることを要求します
※2. `rlACAgent` は１つのネットワークで状態価値関数および方策関数を表現することができません

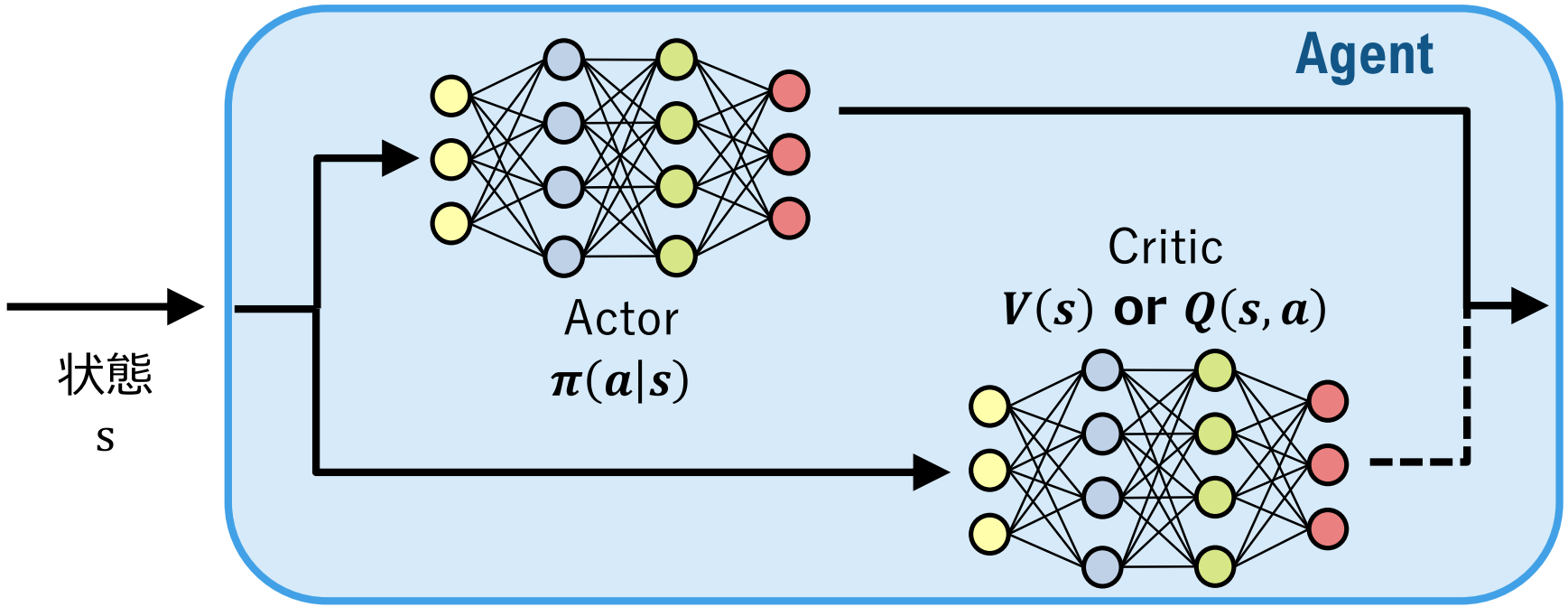
3. エージェント構築

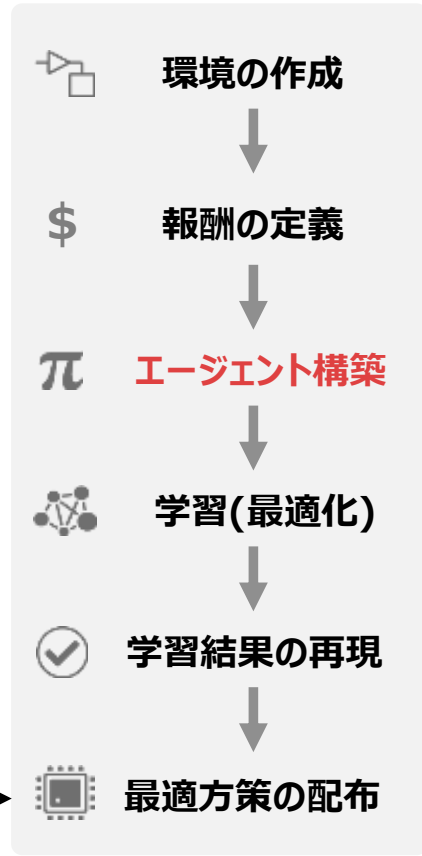
Actor & Critic 用ネットワークの作成

2. ネットワークの構築
   - Q-Learning 系
   - 方策勾配系

```
agent = rlDQNAgent(critic, agentOptions);
```

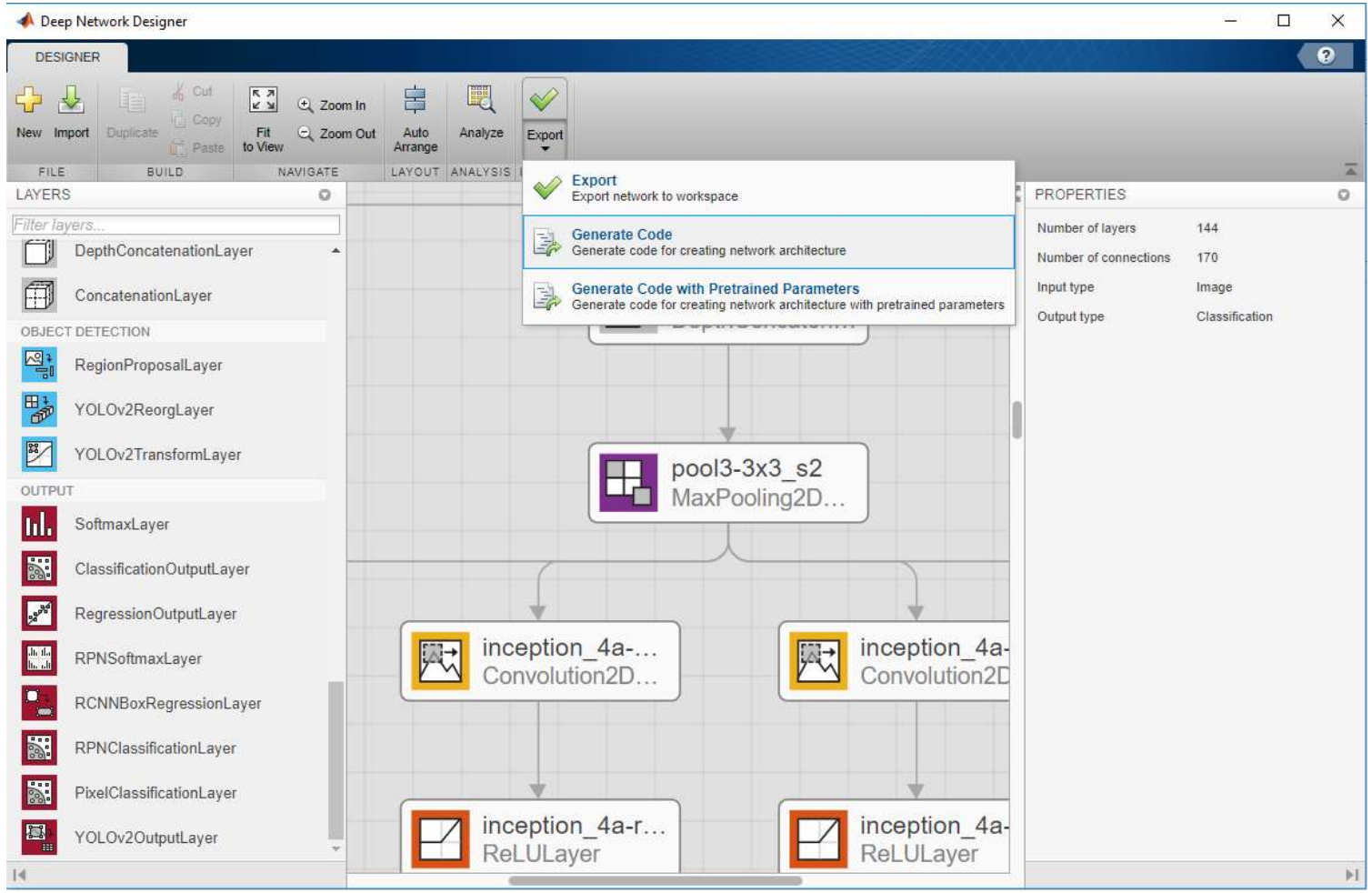
```
agent = rlDDPGAgent(actor, critic, agentOptions);
```



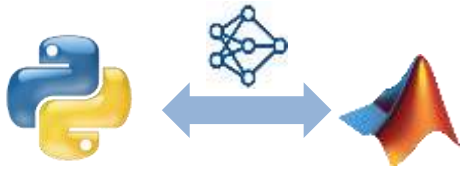


Deep Network Designer
対話的なネットワーク作成アプリ

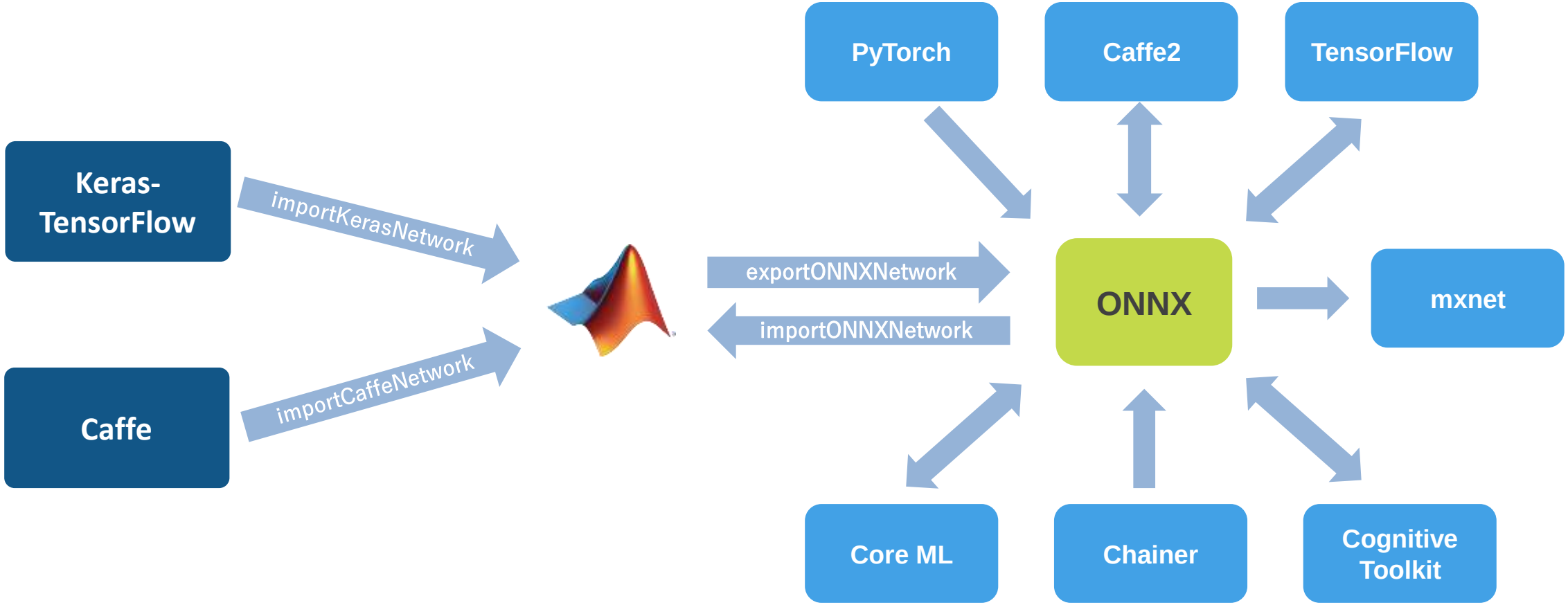
- Simulink のようにマウス操作でネットワークを作成
- ネットワークの整合性をチェックする「Deep Network Analyzer」
- 作成したネットワークに対するコード生成



Model Importers



- Keras-TensorFlow をはじめとする他のフレームワークからのネットワークの移行



ONNX = Open Neural Network Exchange Format

3. エージェント構築

Hyper-Parameters の設定

- Actor の定義

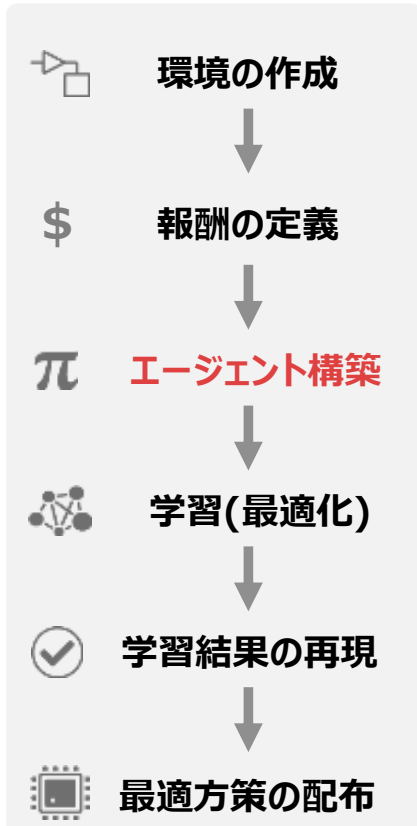
```
actorOptions = rlRepresentationOptions('Optimizer','adam','LearnRate',1e-4, ...
'GradientThreshold',1,'L2RegularizationFactor',1e-5);
actor  = rlRepresentation(actorNetwork, … ,actorOptions);
```

- Critic の定義

```
criticOptions = …
critic  = rlRepresentation(criticNetwork, … ,criticOptions);
```

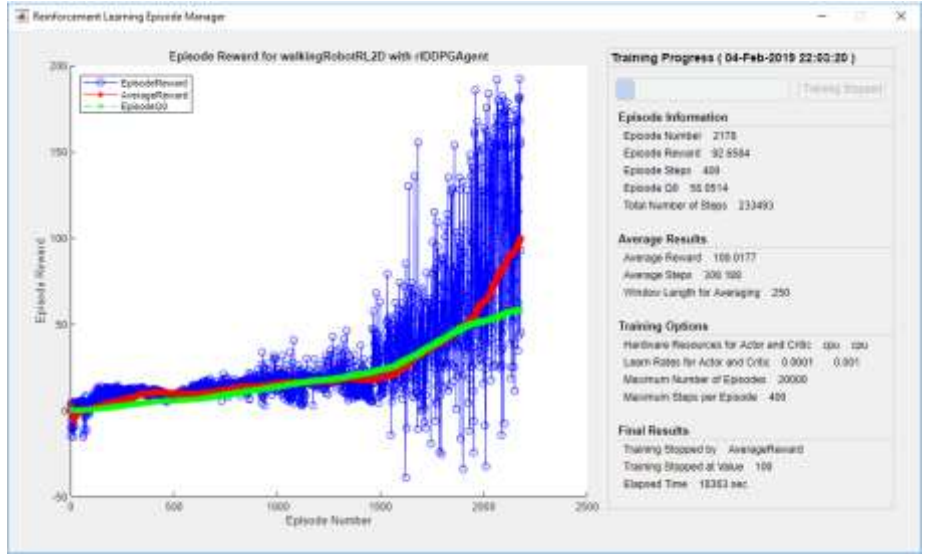
- Agent Option の定義
  - DDPG の例

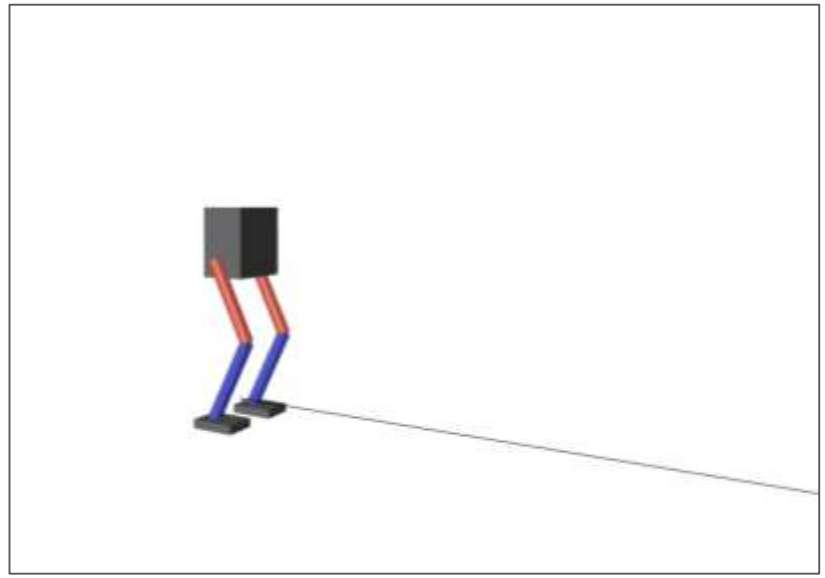
```
dT = 0.05; Tf = 20;
agentOptions = rlDDPGAgentOptions('SampleTime', dT, 'TargetSmoothFactor',1e-3, ...
'ExperienceBufferLength', 1e6, 'DiscountFactor', 0.99, 'MiniBatchSize', 128, ...
'UseDevice','GPU');
```

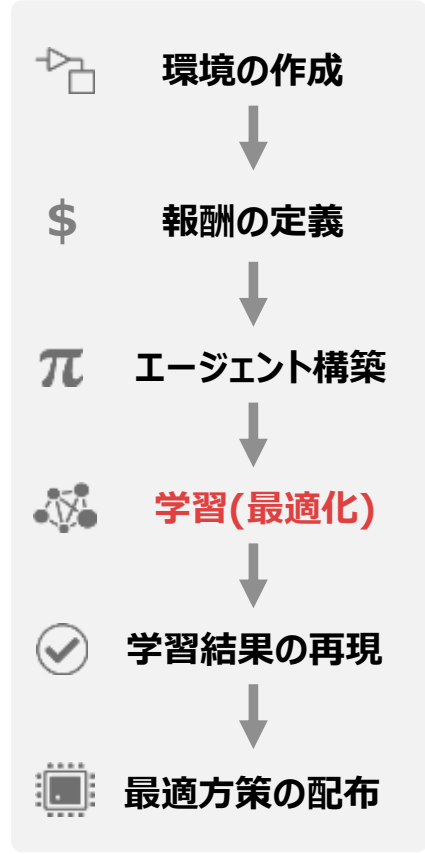


4. 学習（最適化）

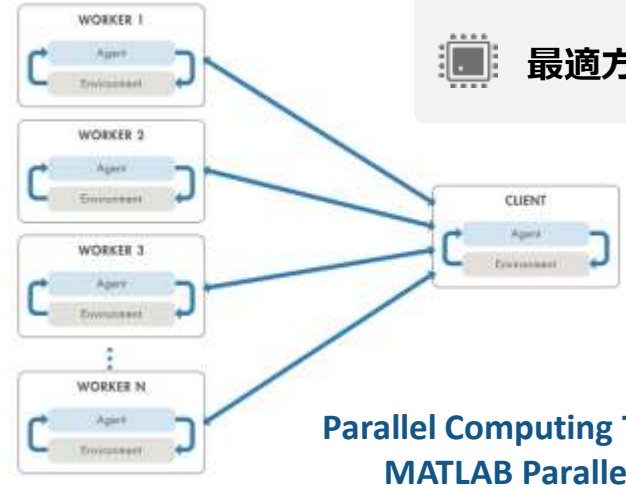
```
trainingStats = train(agent,env,trainOptions);
```







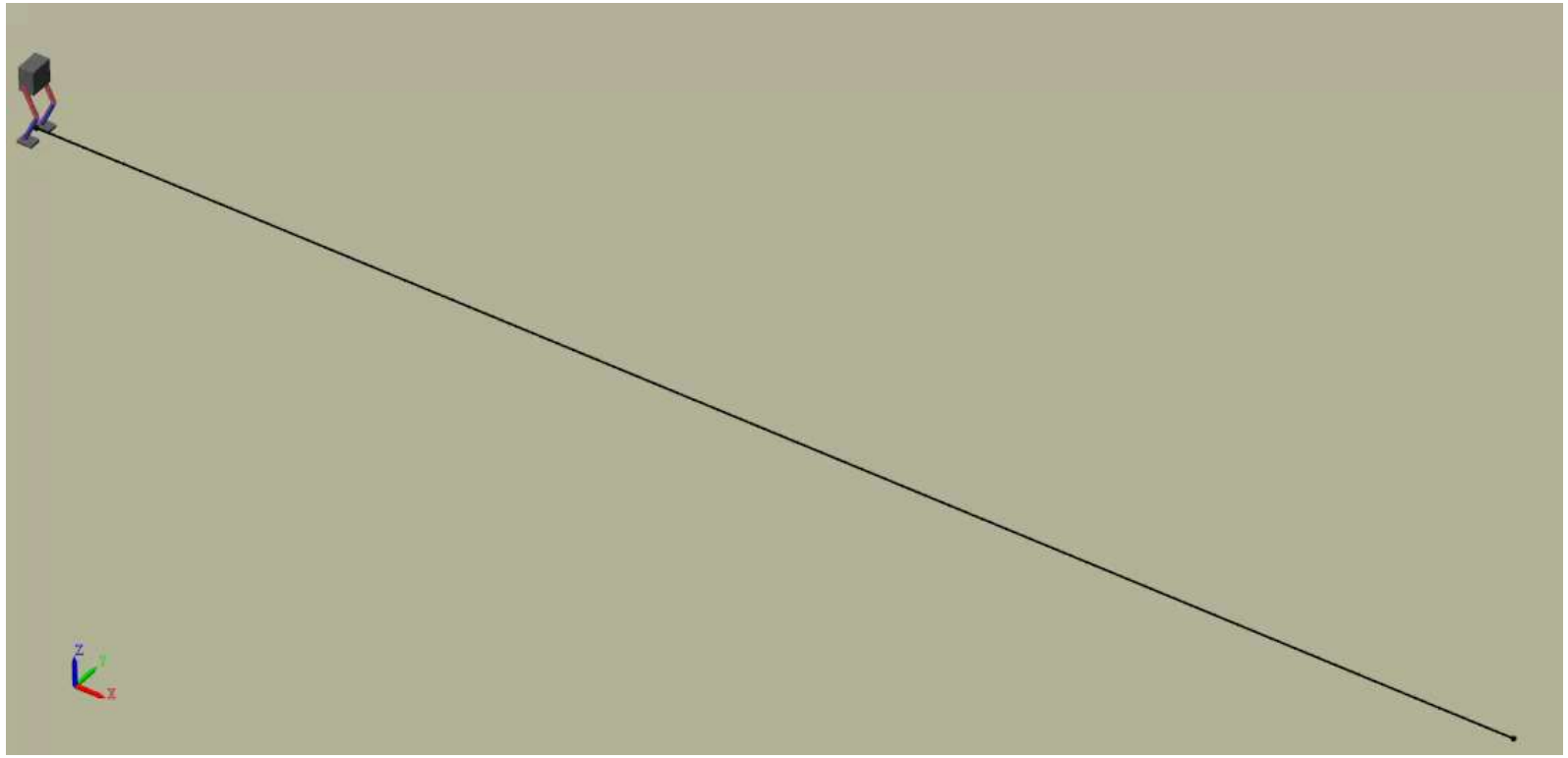
- 学習の高速化
  - マルチコア/マルチCPUs、クラスタ環境を利用したシミュレーションの並列化
  - 高性能なNVIDIA® GPUを使ったディープネットワークの学習の高速化



Parallel Computing Toolbox™
MATLAB Parallel Server™

5. 学習結果の再現

```
simOptions = rlSimulationOptions('MaxSteps', 400);
experience = sim(env, agent, simOptions);
```

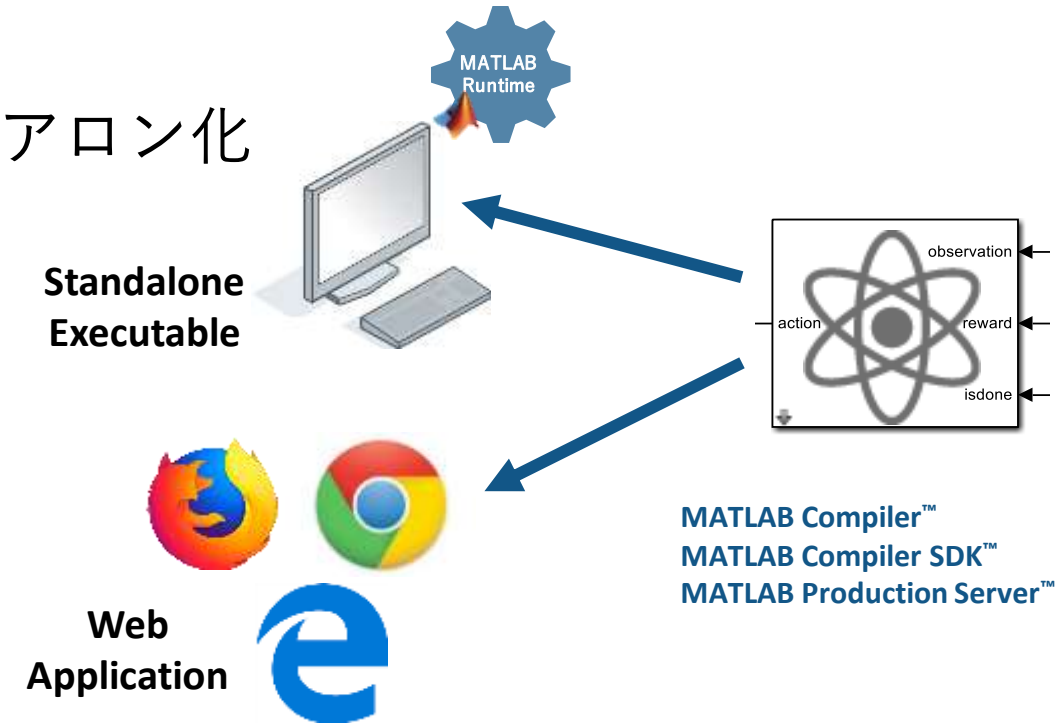


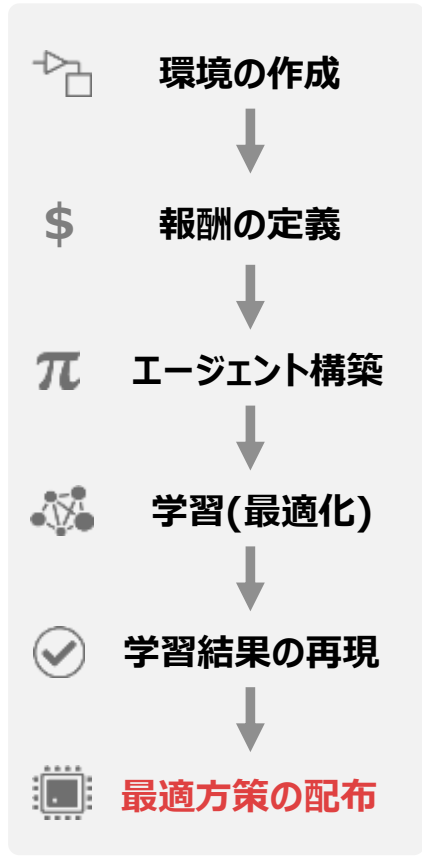
環境の作成
↓
報酬の定義
↓
エージェント構築
↓
学習(最適化)
↓
学習結果の再現
↓
最適方策の配布

6. 最適方策の配布

- エージェントが習得した最適方策の配布準備
  - 方策をファイル(MAT形式)として保存
  - 方策のMATLAB関数化
    - `generatePolicyFunction`

- 最適方策のスタンドアロン化





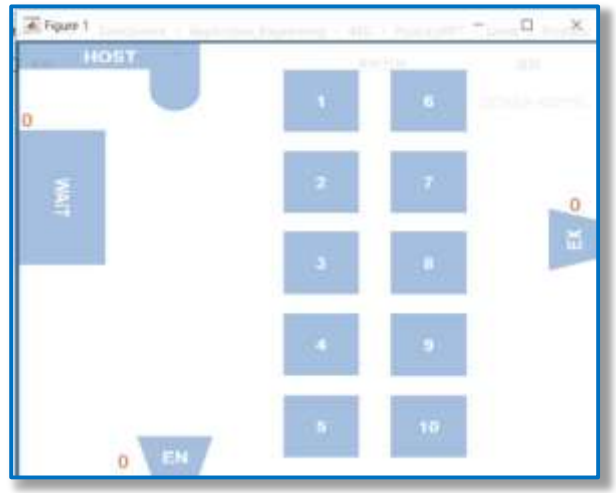
Agenda
強化学習 ~ 最適制御のためのディープラーニングの応用 ~

- 強化学習とは
- MATLAB による強化学習
- まとめ

まとめ
強化学習 ~ 最適制御のためのディープラーニングの応用 ~

- 強化学習専用製品 Reinforcement Learning Toolbox の登場

- MATLAB / Simulink Product Family が提供する「環境」の構築
  - Simscape™ をはじめとするプラントモデリング製品との連携
  - SimEvents™ による待ち行列などの離散的事象問題への取り組み



- レファレンス・アプリケーションを多数提供
  - https://www.mathworks.com/help/reinforcement-learning/examples.html