カスタムSIに使ってみよう

OpenAI Gymを使った強化学習

2020年11月26日

株式会社エクサ
堀　扶
tasuku-hori@exa-corp.co.jp

注意

- 主観を含む記述が含まれています
  - 書籍など他のコンテンツにて確認してください
- 登場する会社名、製品名およびサービスは、各社の商標または登録商標です

- セッションの目的
  - 業務システムなどのSIプロジェクトにて強化学習を適用する際に、最初に知っておくと良い情報を提供
    - とりあえず動作するものを作るために必要な用語、設計の流れを解説

目次

- 用語理解編　自律走行車を例に用語を解説
- 設計試行編　じゃんけんを強化学習で設計・実装
- まとめ

- 補足　※強化学習アルゴリズムの知識が必要

強化学習＞教師あり学習

教師あり学習・強化学習

教師あり学習

- 学習データが必要
- ビジネス事例多い
- 画像処理、自然言語処理
- ほぼ共通のトレーニング手段
- 画像、NLP以外にも適用進む

強化学習

- 環境を模倣する機能で代用
- ビジネス事例意外と少ない
- 囲碁将棋、工場ロボット関係
- 多様なトレーニング手段
- 個別システム適応させて利用

Donkeycarを例に用語解説します

https://youtu.be/KrWM_T5NQuU

Donkeycar(v3)は教師あり学習

教師あり学習

学習
データ

手動操作

自動運転

入力：前方カメラ
出力：スロットル、ステアリング

周回コースをはずれることなくできるだけ早く自立走行する

Donkeycar の問題点

- 自動運転開始までに、準備に時間がかかる
  - ラジコンを上手に運転できる人が必要
  - 学習データに必要なデータ量：2万件以上
  - 1件＝1/20秒なので、16.6分以上ミスなく走行できる人が必要
- 解決策
  - 転移学習→学習に必要なデータ量を1000件以下にできる
  - 強化学習→実コースに相当する環境をプログラムで提供

Donkeycar

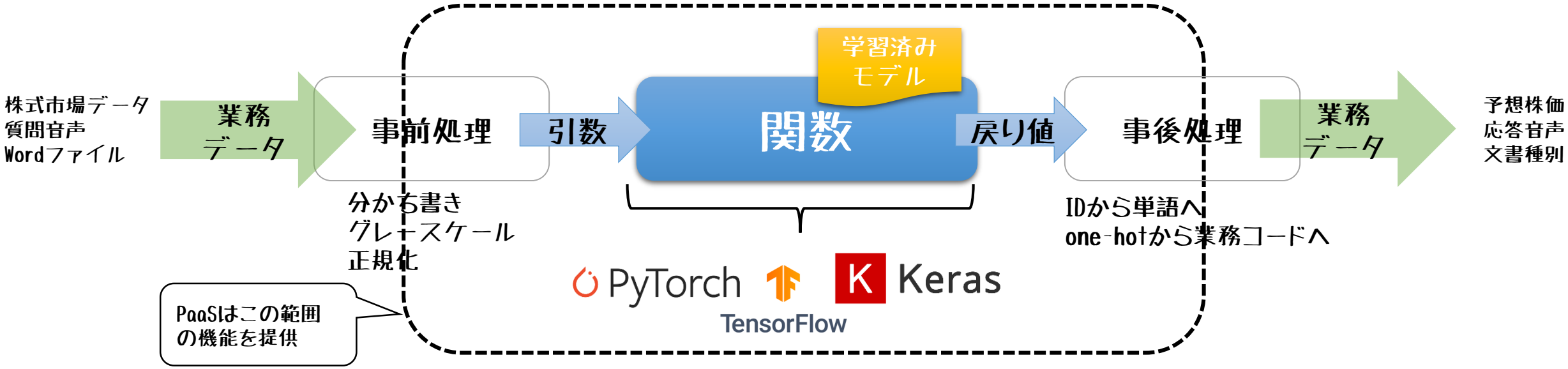
jetcar

Deep Racer

Donkey Simulator

AIはプログラム上では「関数」

- 引数：数値群、戻り値：数値群
- 事前処理・事後処理
  - 業務データ→引数、戻り値→業務データ
- 関数内のロジックは機械学習ライブラリで実装



強化学習におけるAI＝方策

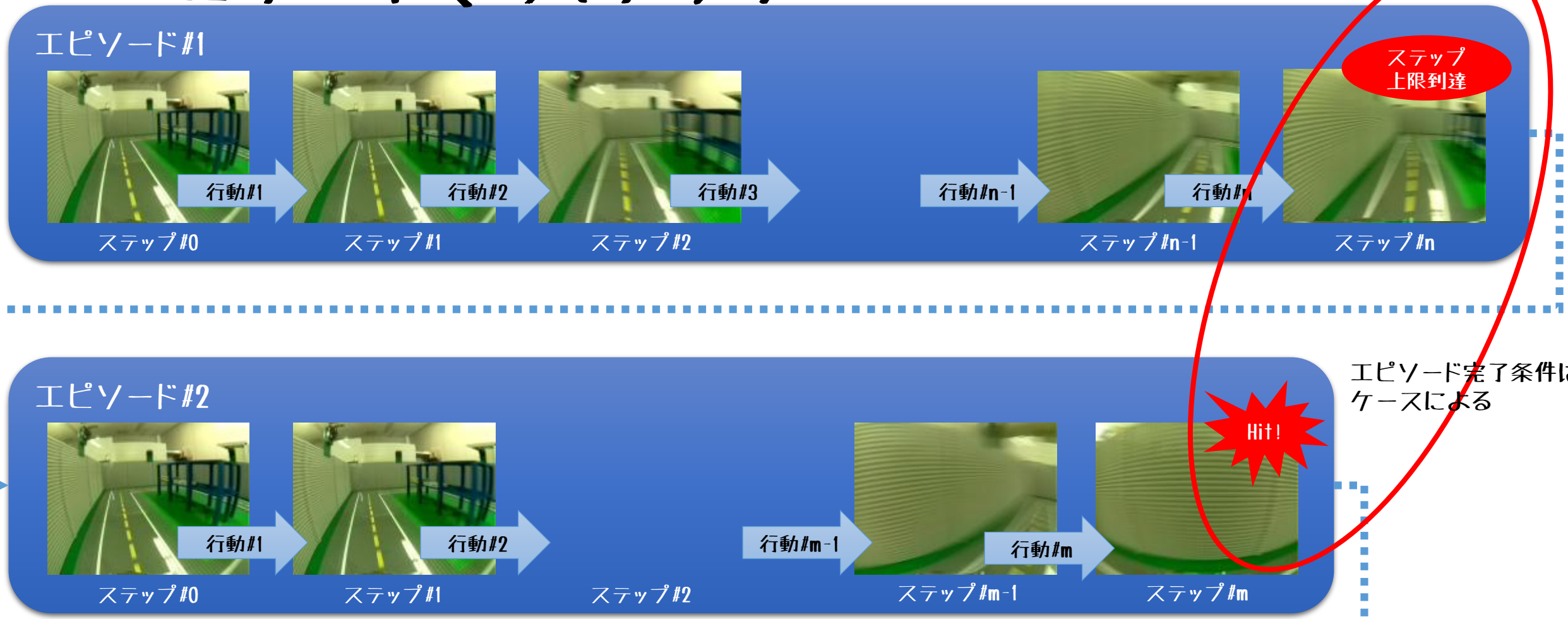
- 強化学習ではAI本体を「方策」と呼ぶ
  - 「方策」の引数：「観測」
  - 「方策」の戻り値：「行動」

方策（Policy）

- 強化学習における用語、AI関数本体
- 機械学習におけるトレーニング対象の関数
- 強化学習では引数を「観測」、戻り値を「行動」と呼ぶ(後述)
- 既存の実装済み関数を活用できる
  - 中身のロジックを知らなくても、SI適用できてしまう(独自実装も可)

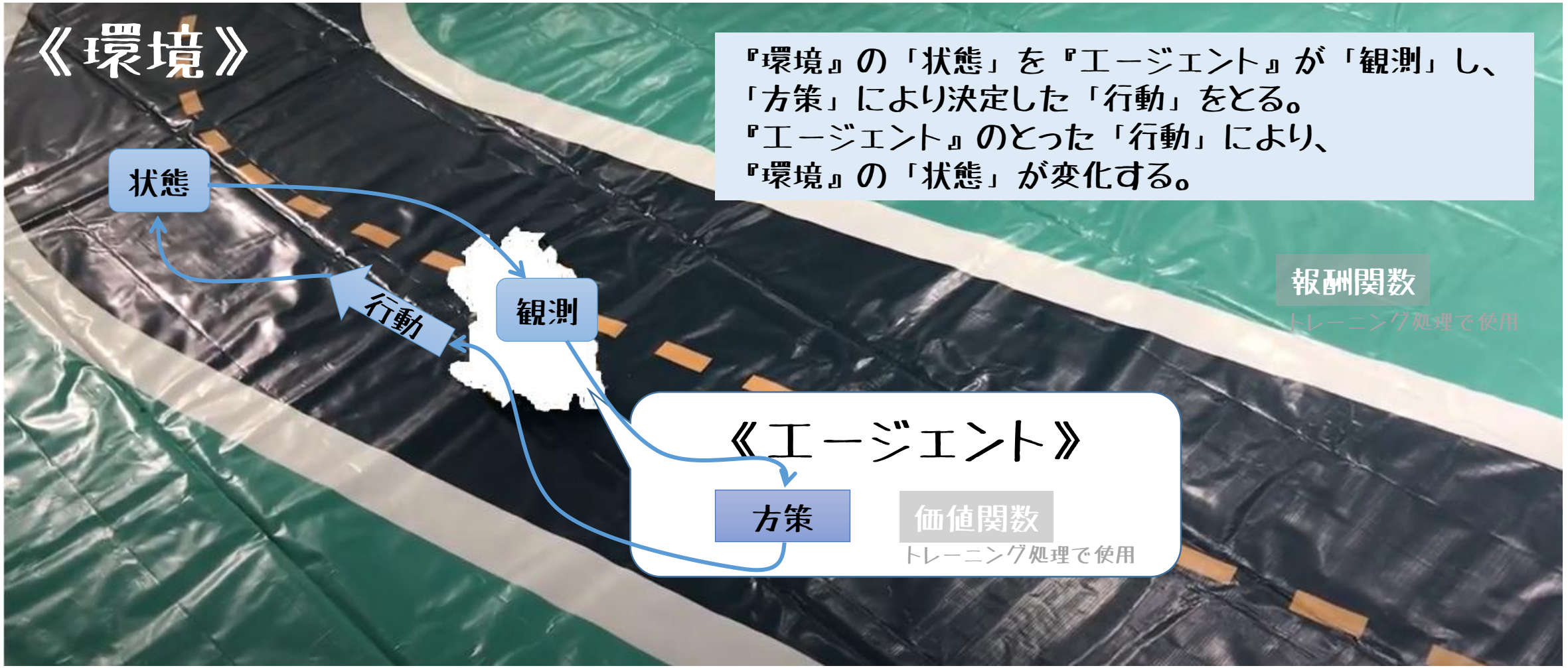
トレーニングにおける単位

エピソード、ステップ



環境、エージェント

※強化学習の用語はロボットを想定すると理解しやすい



観測（observation）

エージェントが次の行動を選択する際に活用するデータ
方策の引数

Donkeycar

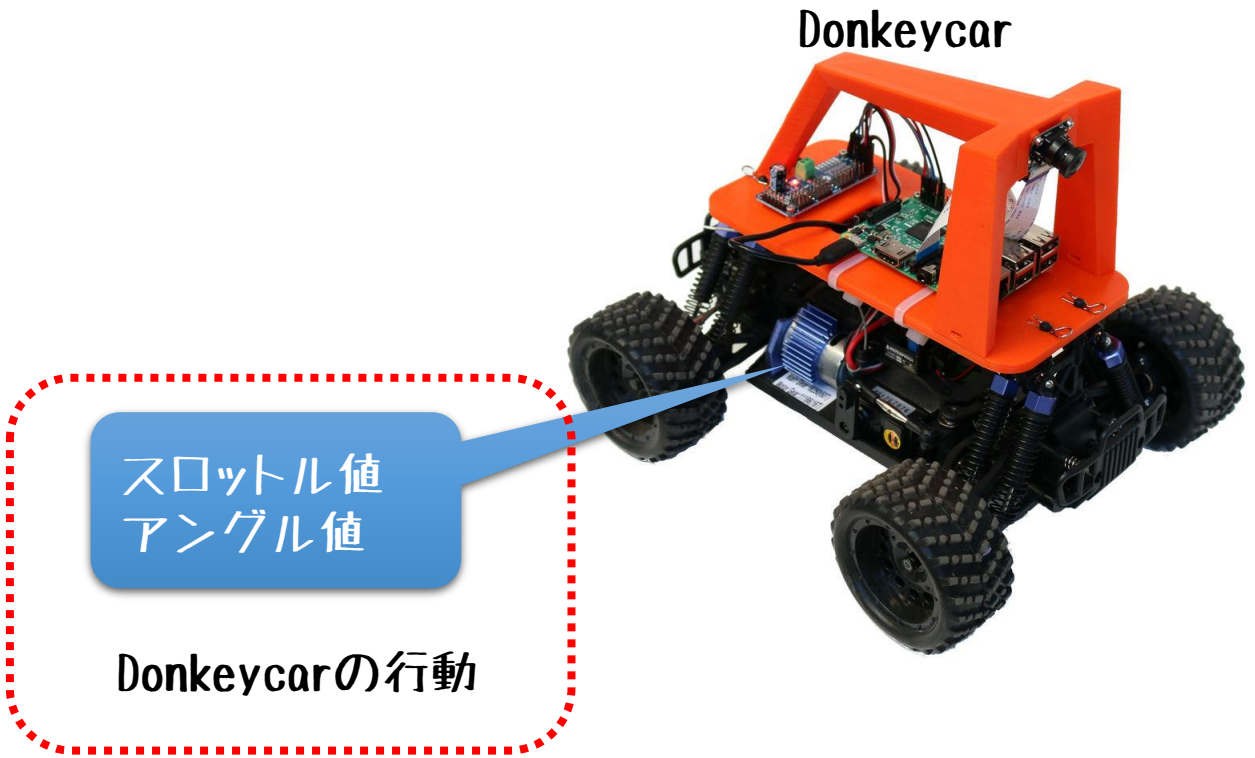
Wide Angle Pi Camera
・前方カメラ画像

Donkeycarの観測

観測空間：0から255までのfloat値で構成された(120, 160,3)形式配列

行動（action）

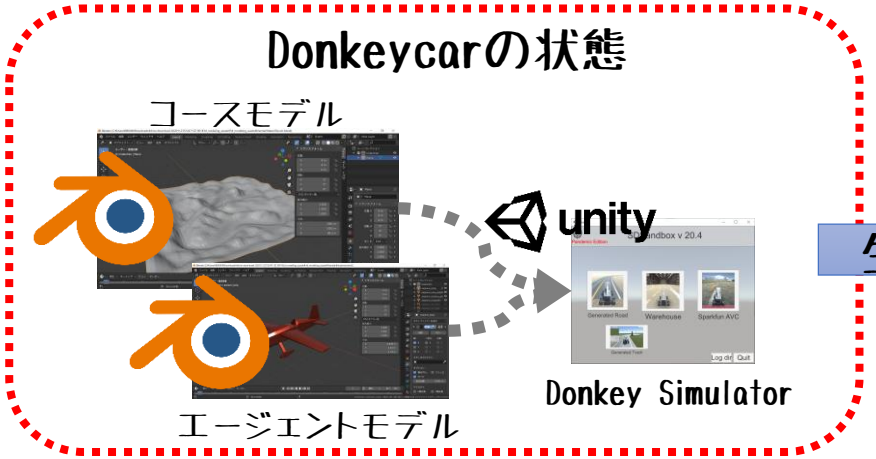
各ステップにおいて、エージェントがとった行動を表すデータ
方策の戻り値

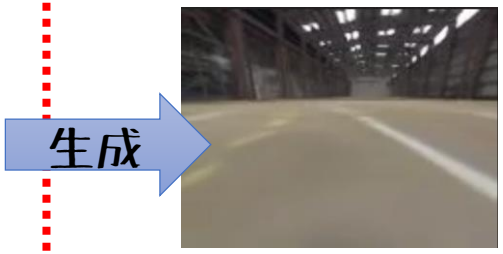


行動空間：-1.0から1.0までのfloat値×2

状態（state）

環境側が保持しているデータ
状態をもとに観測が生成される

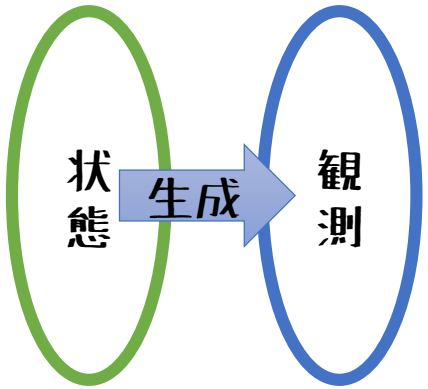


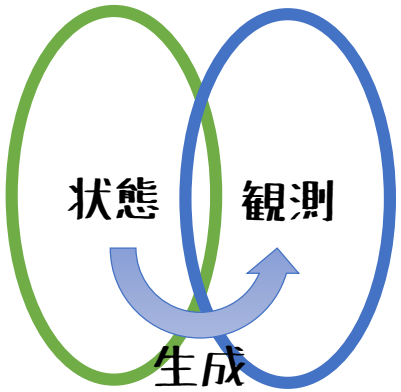


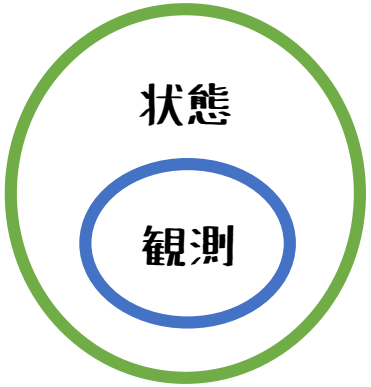
《観測：強化学習》
シミュレータ前方視界

≒

《観測：本物》
Donkeycar前方カメラ







※初心者向け

報酬（reward）

行動の良い・悪いを決める指標値（float値の範囲内で仕様を決める）
各ステップで報酬を計算する
→あるステップにおける”状態”から算出可能な数値として設計する

Donkeycarの報酬
　エージェントの速度/黄破線からエージェントまでの距離

報酬関数（reward function）

- ある”行動”をとった時の報酬値を返す関数
- ステップ単位で実行される
- 報酬仕様
  - 範囲（Open AI Gymデフォルトは-np.infからnp.inf）
  - ステップ毎の報酬
  - エピソード完了時の報酬
  - “状態”から算出するロジック
- 通常は”環境”クラス内に実装される
- 機械学習ライブラリに既存の関数実装がない
  - Open AI Gym提供の既存”環境”クラスには実装済み
  - カスタムSI場合は独自実装が必要

強化学習の目的

トレーニング中の指標

収益（revenue）、価値（value）

1エピソード内で得たすべての報酬合計値

収益

エピソード完了時に評価するための指標、「決算」に相当

Step 1 報酬 | Step 2 報酬 | … | Step n 報酬

選択した行動に対する報酬を表す数値

トレーニング中エピソード内で得たすべての報酬値合計に
エピソード完了まで得られる予測報酬値合計を加算した値

価値

エピソード中に評価するための指標、「推定実績」に相当

Step 1 報酬 | Step 2 報酬 | … | Step k 報酬 | Step k+1 予測報酬 | … | Step n 予測報酬

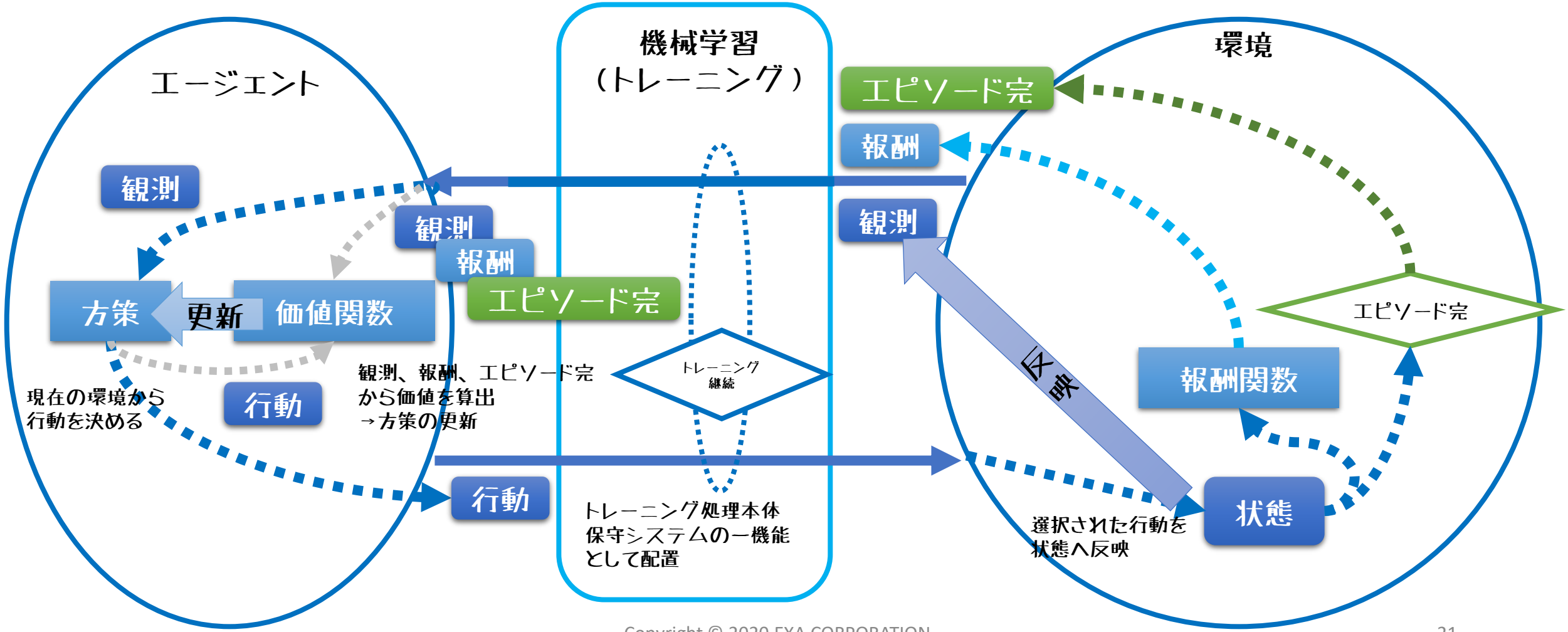
マルコフ性前提の場合、最後に選択された行動をもとに予測

価値関数（value function）

- 現在のエピソードの価値を算出する関数
  - 引数は強化学習アルゴリズムにより異なる
  - 戻り値：価値
- 算出された価値をもとに方策（パラメータ）を更新する
- Stable Baselinesを使用する場合、基本独自実装しない
  - 強化学習アルゴリズムを選択した時点で価値関数も選択したこととなる
  - マルコフ性（現在の状態から将来の報酬の確率分布が決まる性質）

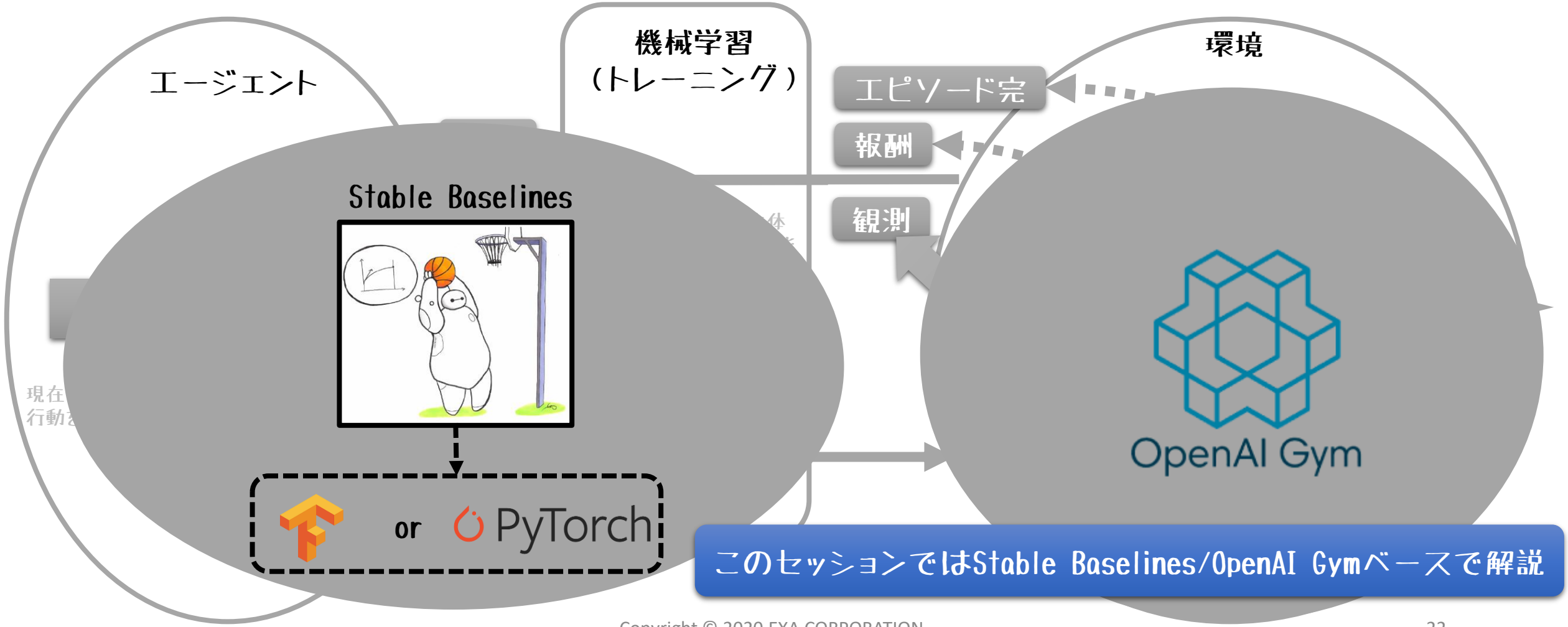
システムコンテキストダイヤグラムっぽい絵

《保守システム（トレーニング）》



使用するソフトウェア例

《保守システム（トレーニング）》



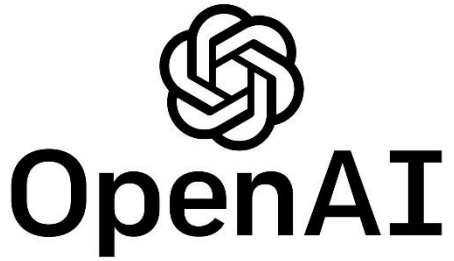
OpenAI Gym

- OpenAI が2016年4月に発表した強化学習アルゴリズムの検証プラットフォーム
  - 機械学習ライブラリ(TensorFlow, PyTorch..)に依存しない
- 強化学習アルゴリズム開発や比較のためのツールキットも提供
  - 検証に活用可能なレトロゲームの環境クラス群
  - カスタム環境を構築するためのテンプレートメソッド
- MITライセンス準拠（一部環境では要MuJoCoライセンス）

- OpenAI
  - 2015年12月設立された人工知能を研究する非営利団体
  - 最近ではGPT-3(Transformer言語モデルベース)で有名
    - OpenAI’s GPT-3 may be the biggest thing since bitcoin
    - Microsoft、汎用言語モデル「GPT-3」のライセンス取得

OpenAI Baselines

- OpenAI が公開しているGitHubリポジトリ
- 高品質な強化学習アルゴリズムの実装群を提供(MITライセンス)
- 新しい強化学習アルゴリズムの研究向け
  - 自分の研究結果と比較する「ベースライン」として
- TensorFlow前提
  - masterブランチ：TensorFlow1.4～1.14
  - tf2ブランチ：TensorFlow2.0



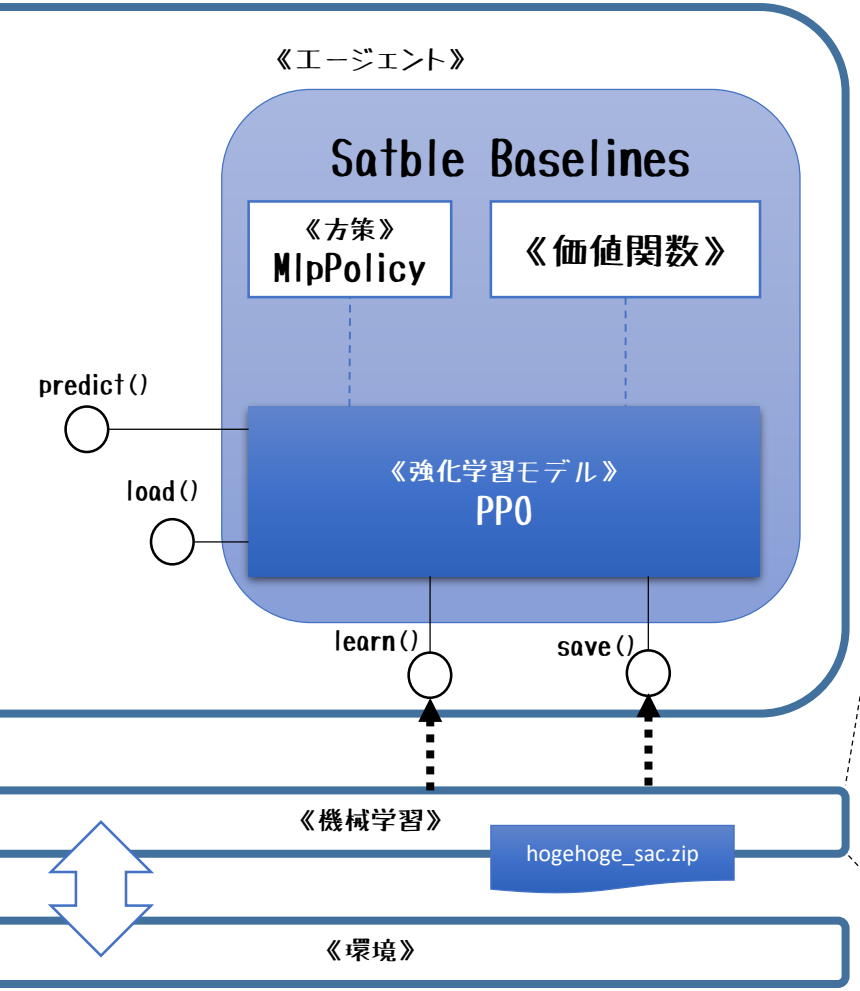
- アルゴリズム間のI/Fが統一されていない
- 公開ドキュメントが少ない

Stable Baselines

- OpenAI Baselinesをフォークしリファクタリングされた強化学習アルゴリズム群を提供（MITライセンス）
- 強化学習アルゴリズムの共通I/F、公開ドキュメント
- カスタム環境、カスタム方策にも対応可能
- OpenAI Baselines より先に最新の強化学習アルゴリズムに対応
- TensorFlow/PyTorchに対応
  - Stable Baselines: Tensorflow1.x(2.0は計画中)
  - Stable Baselines3: PyTorch1.4～
- Stable Baselines Zoo上にて学習済みAgentを公開

カスタムSI開発向き

Stable Baselinesトレーニング処理実装例




```
《トレーニング処理サンプル》
:
# トレーニング用環境の作成
env = HogehogeEnv(length=10, max_steps=10000)
env = Monitor(env, './logs', allow_early_resets=True)
env = DummyVecEnv([lambda: env])

# Stable Baselines PPO モデルを MlpPolicy指定で作成
# 妥当性検査ON、TensorBoardによる可視化ON
model = PPO('MlpPolicy', env, verbose=1, tensorboard_log=
'./logs')

# トレーニング環境を使ってSACをトレーニング
model.learn(total_timesteps=10000)

# モデルファイルを保存
model.save('hogehoge_sac')

# 環境を閉じる
env.close()
```

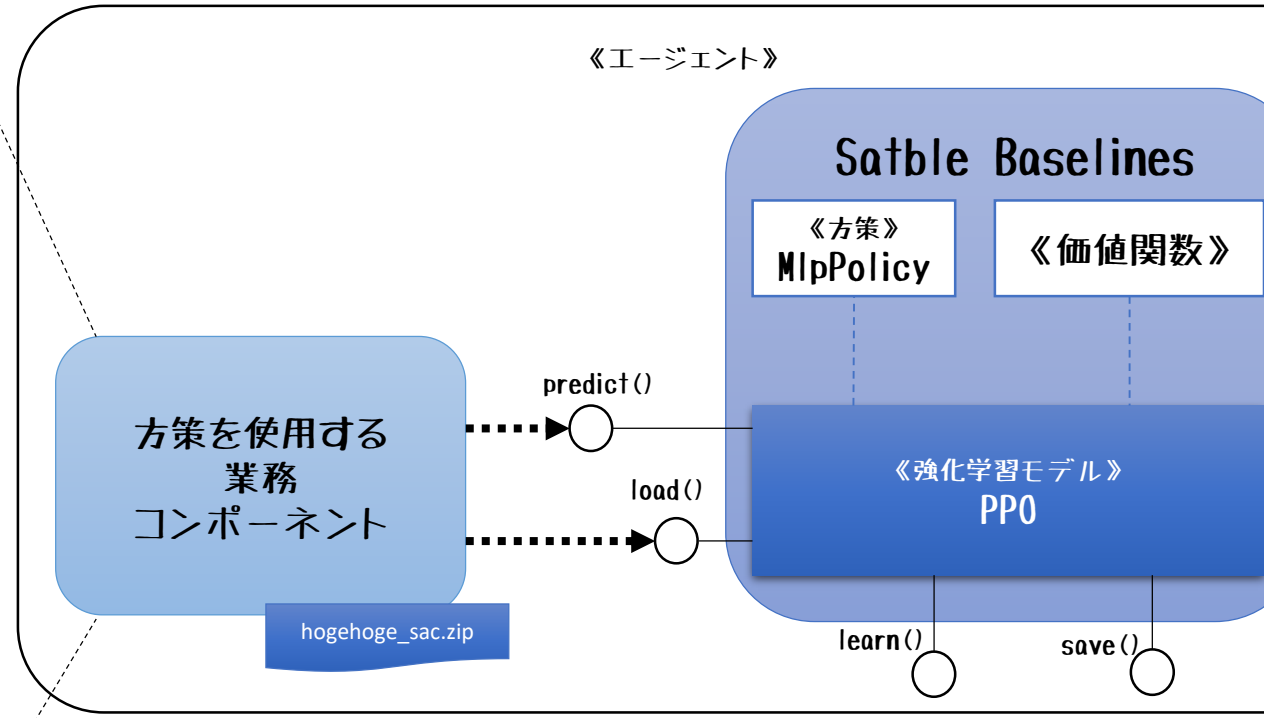
機械学習側で作成したモデルファイルを
エージェント字側実行環境に配置する

Stable Baselines本番側実装例

```
《本番システム上での推論実行サンプル》
:
# 作成済みモデルファイルをロードして学習済みモデルを復元
model = PPO.load('hogehoge_sac')

:
# 実行用AI関数の本体がこれ
# 変数 observation に実システムの観測が格納済みとする
action = model.predict(observation)[0]
```

hogehoge_sac.zip を読み込み
学習済みSACクラスインスタンを復元

predict関数を使って推論結果（行動）を取得



ホットデプロイ機能は
業務コンポーネント側で実装する必要がある

OpenAI Gym カスタム環境クラス

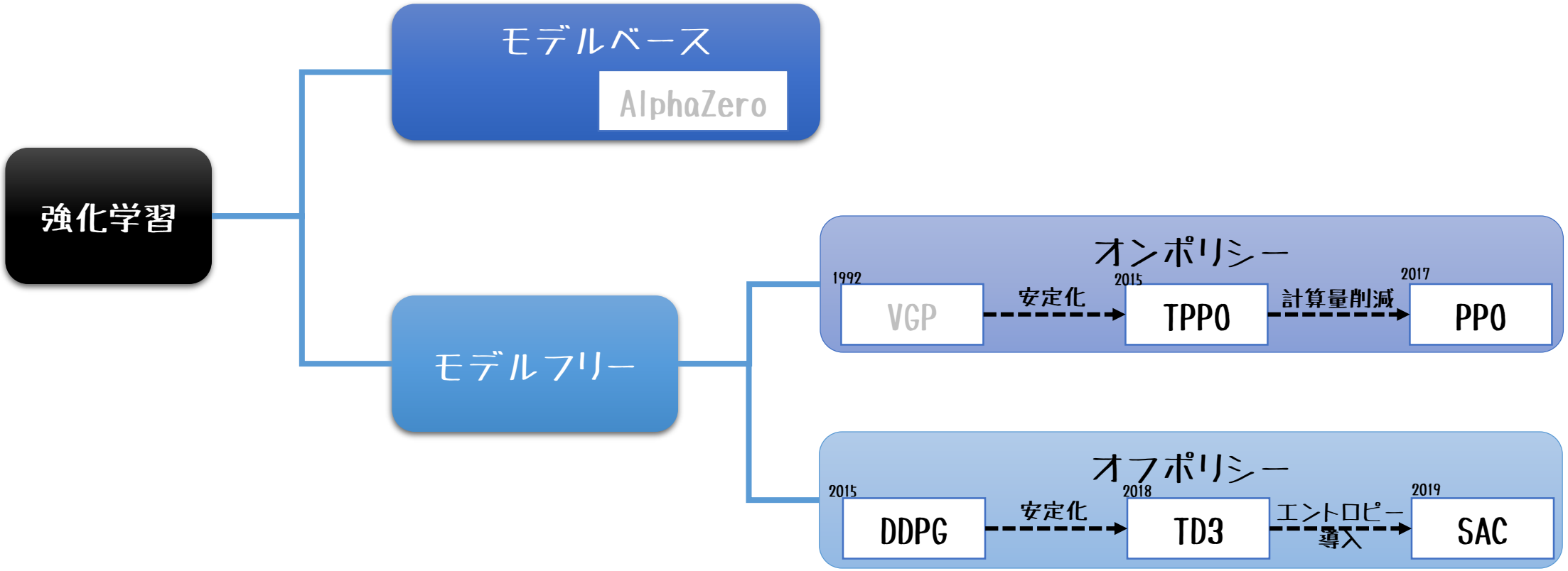
- gym.Env クラスを継承、以下のメソッドをオーバライドする
- オーバライド必須なプロパティ・メソッド

名称	定義・引数	戻り値	説明
《プロパティ》 action_space	gym.space オブジェクト	N/A	行動空間（方策の戻り値）
《プロパティ》 observation_space	gym.space オブジェクト	N/A	観測空間（方策の引数）
《メソッド》 reset()	なし	observation：観測	エピソード開始時の観測データを返却するメソッドを実装
《メソッド》 step()	action:行動（方策が選択した行動）	observation: 観測 reward:報酬値 done:エピソード完 info:その他情報	方策がとった行動により変化した観測、得られた報酬値、エピソード完了したか、その他情報を返却するメソッドを実装

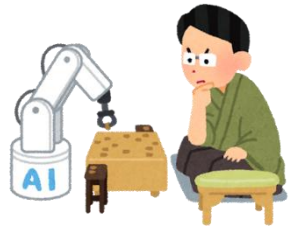
状態・観測をエピソード開始時点に初期化

選択された行動を状態・観測へ反映させ報酬を算出、エピソード完了判定を行う

強化学習アルゴリズムの分類



モデルベース・モデルフリー



《エージェント（モデルベース）》

探索

モデル

疑似経験

Indirect Learning

方策 更新 価値関数

計画更新

実経験

エージェント側にモデルを持ちモデルの生成する疑似経験をもとに方策や価値関数を更新する

《機械学習》

《環境》

《エージェント（モデルフリー）》

環境から得た状態をそのまま方策や価値観数の更新に使用する

Direct Learning

方策 更新 価値関数

計画更新

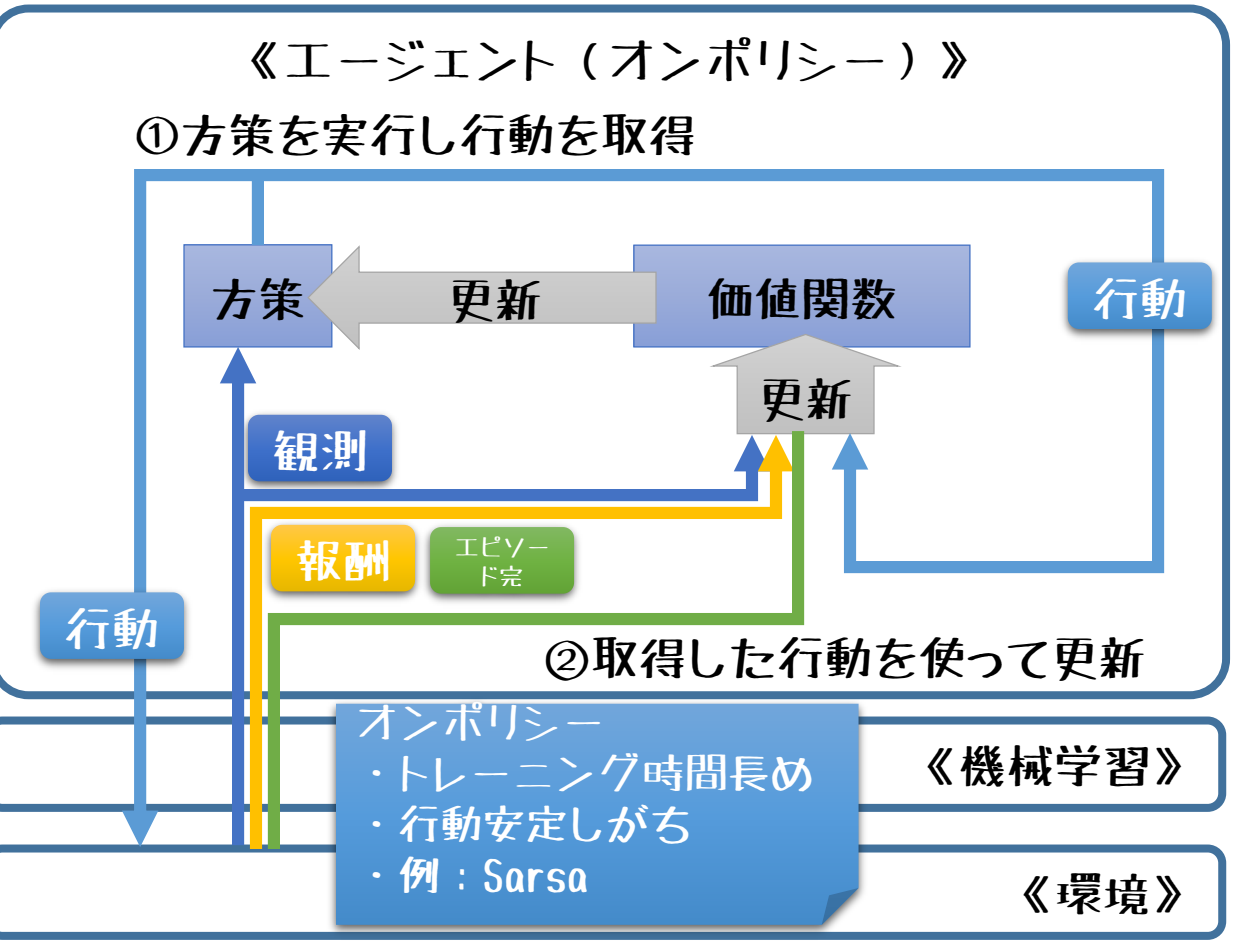
実経験

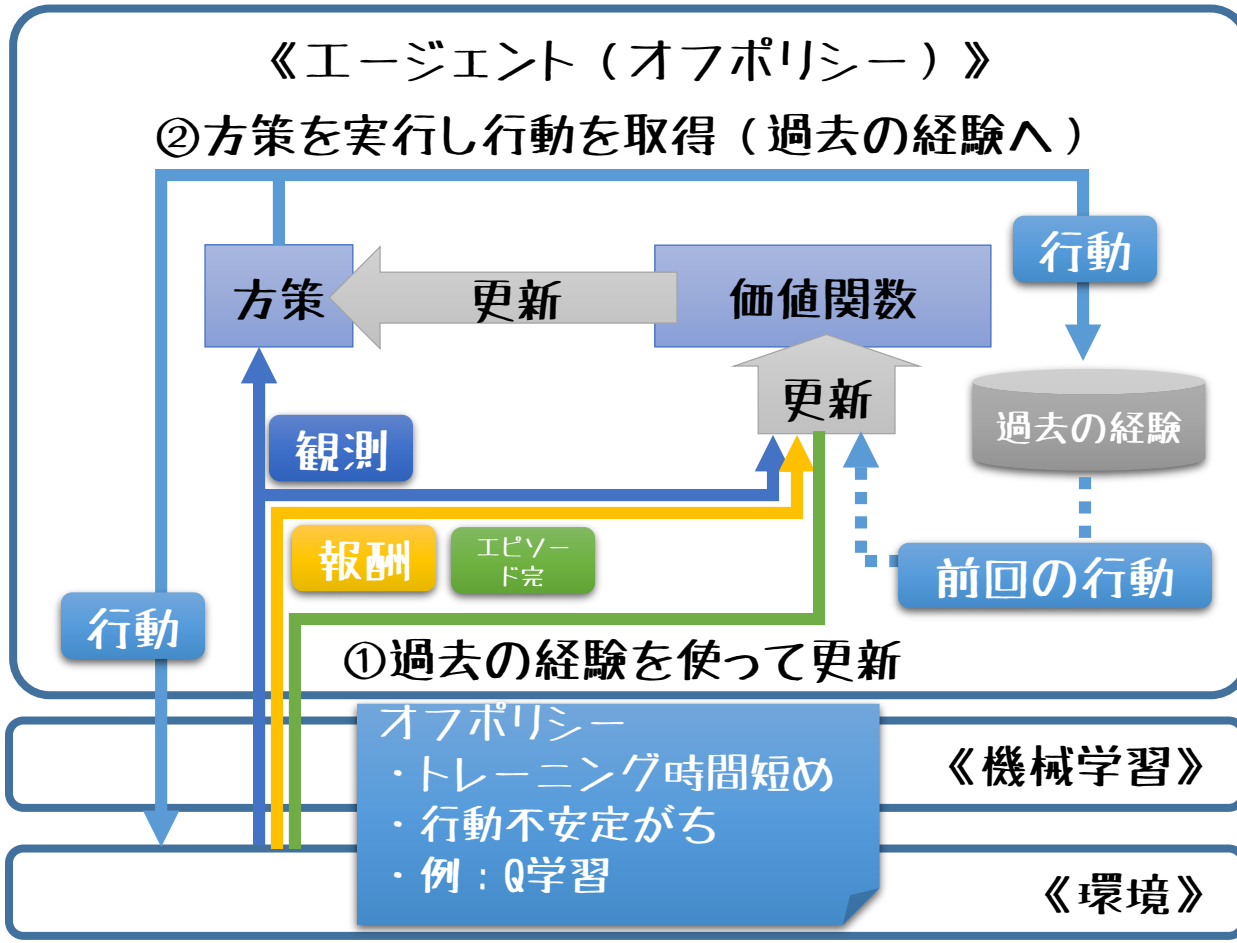
《機械学習》

《環境》

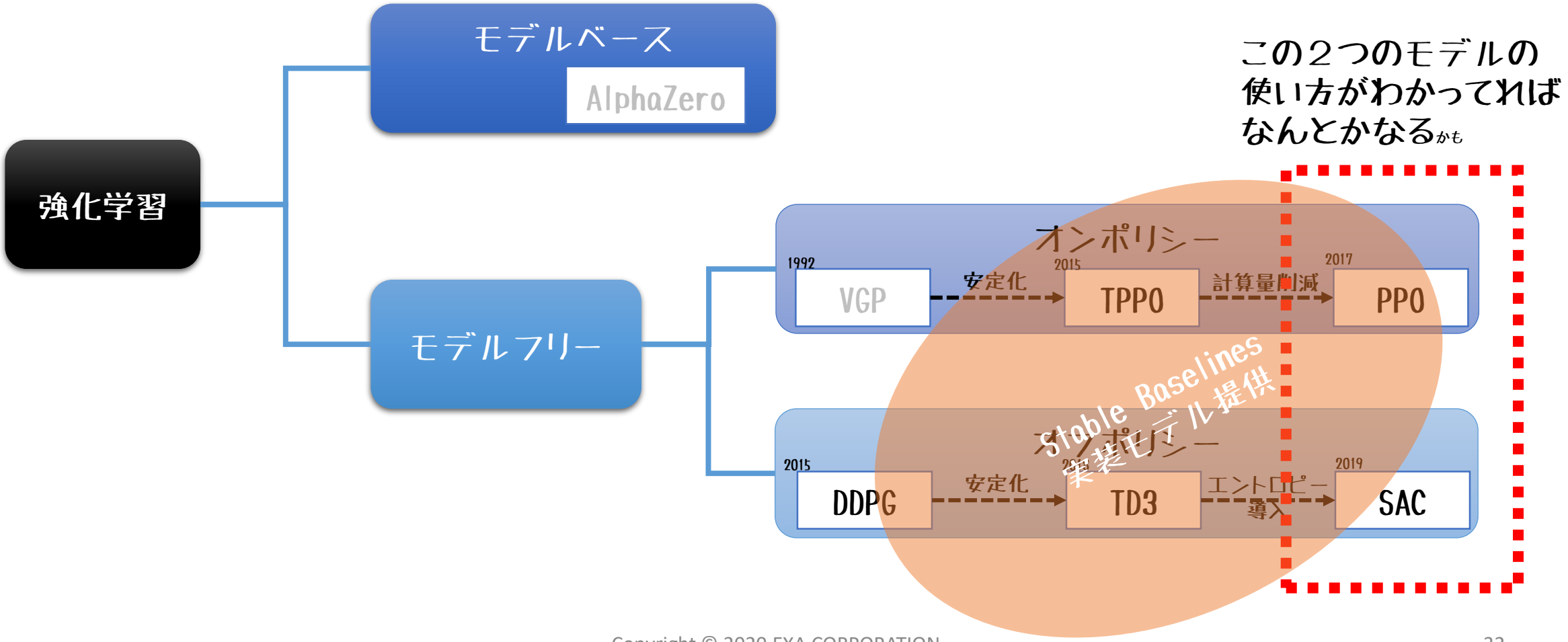
現在の経験のみで方策を更新　　過去の経験も含めて方策を更新

オンポリシー・オフポリシー





Stable Baselines実装モデル



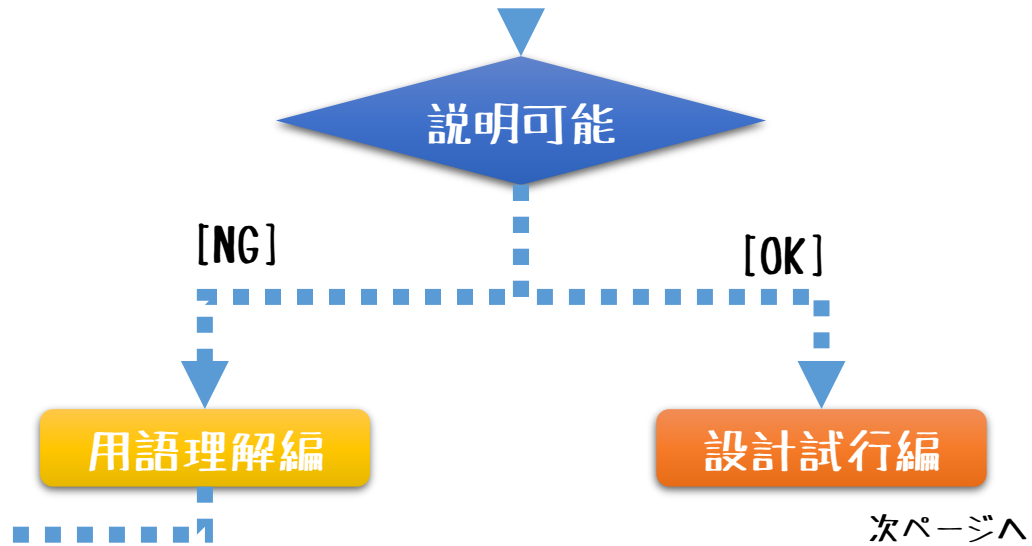
問題：用語を自分の言葉で説明せよ

- エージェント、環境
- エピソード、ステップ
- 方策、行動、観測
- 状態
- 報酬関数

《必須用語》

- OpenAI Gym、Stable Baselines
- 収益、価値、価値関数

設計レビューのために
（丸投げではない）お客様に
用語を理解していただく必要がある！



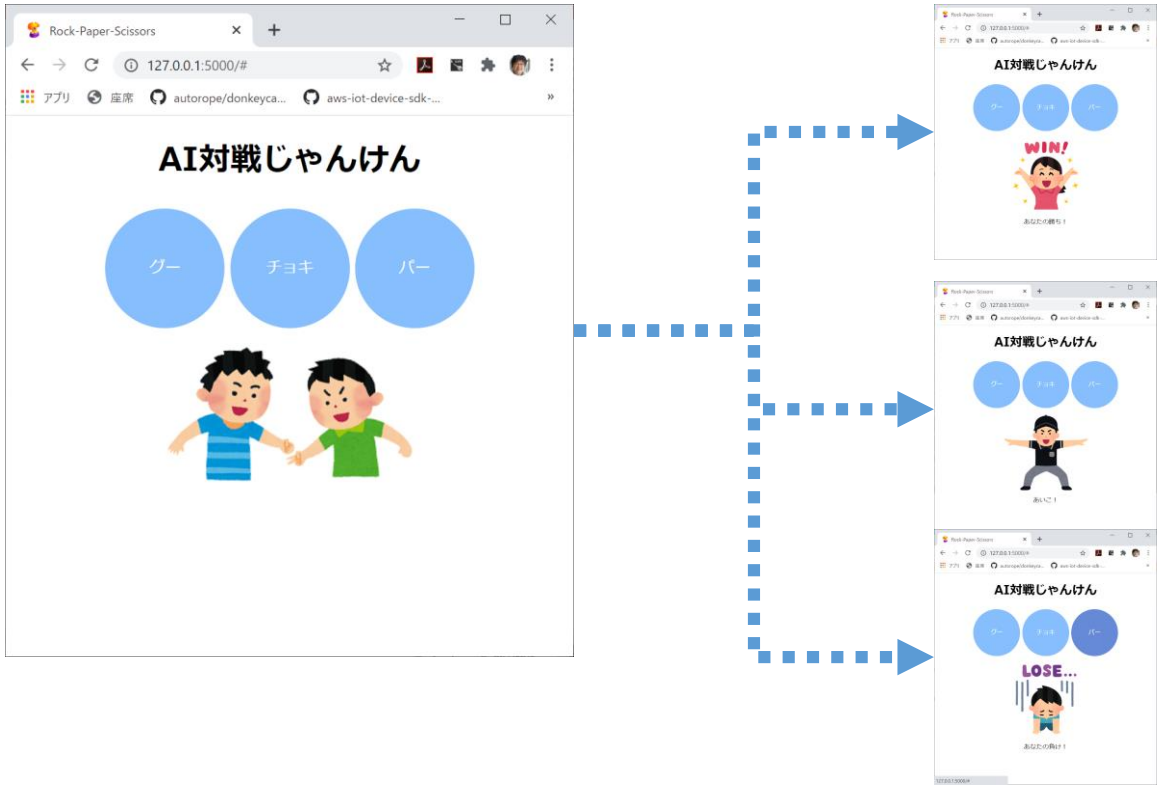
次ページへ

概要

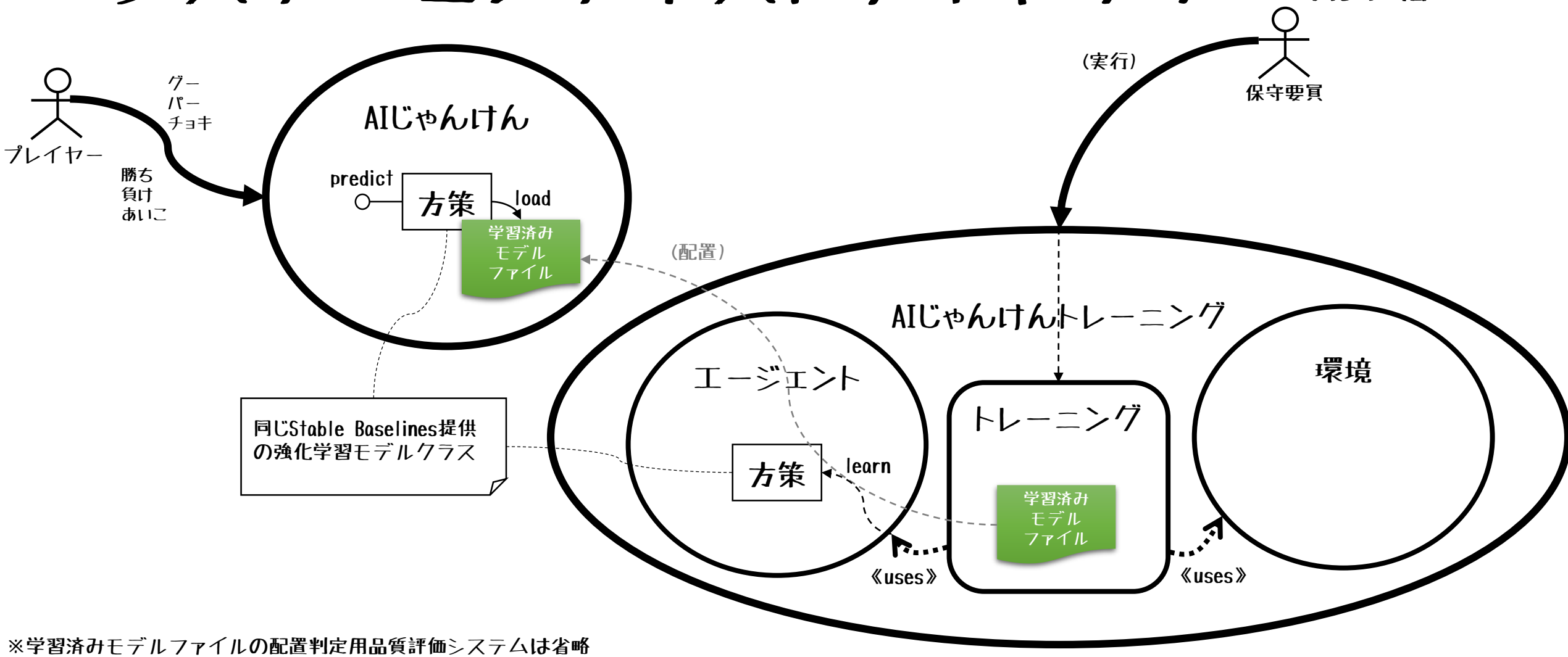
- 設計の流れを解説
  - 3Dシミュレータを使わない事例「じゃんけんAI」
  - 機能要件をどうやって環境に落とし込んでいくか
- 実装についてはサンプルコード参照のこと

サンプルWebアプリ「じゃんけん」

- AIプレイヤー相手にじゃんけんをするだけ



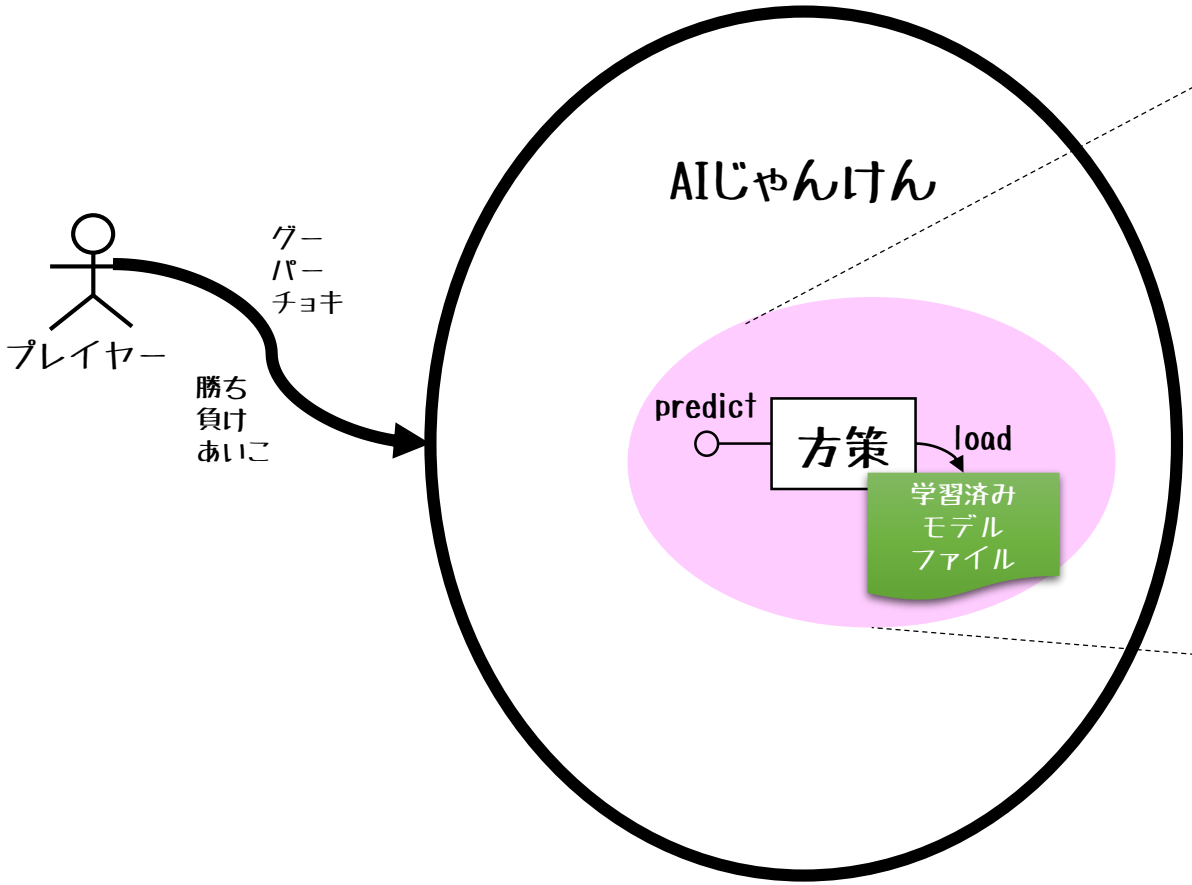
システムコンテキストダイヤグラムっぽい絵



※学習済みモデルファイルの配置判定用品質評価システムは省略

Stable Baselines使用を決めた段階でほぼ決まる処理①

AIじゃんけんの方策呼び出し



強化学習アルゴリズムPPOを使う場合の実装例

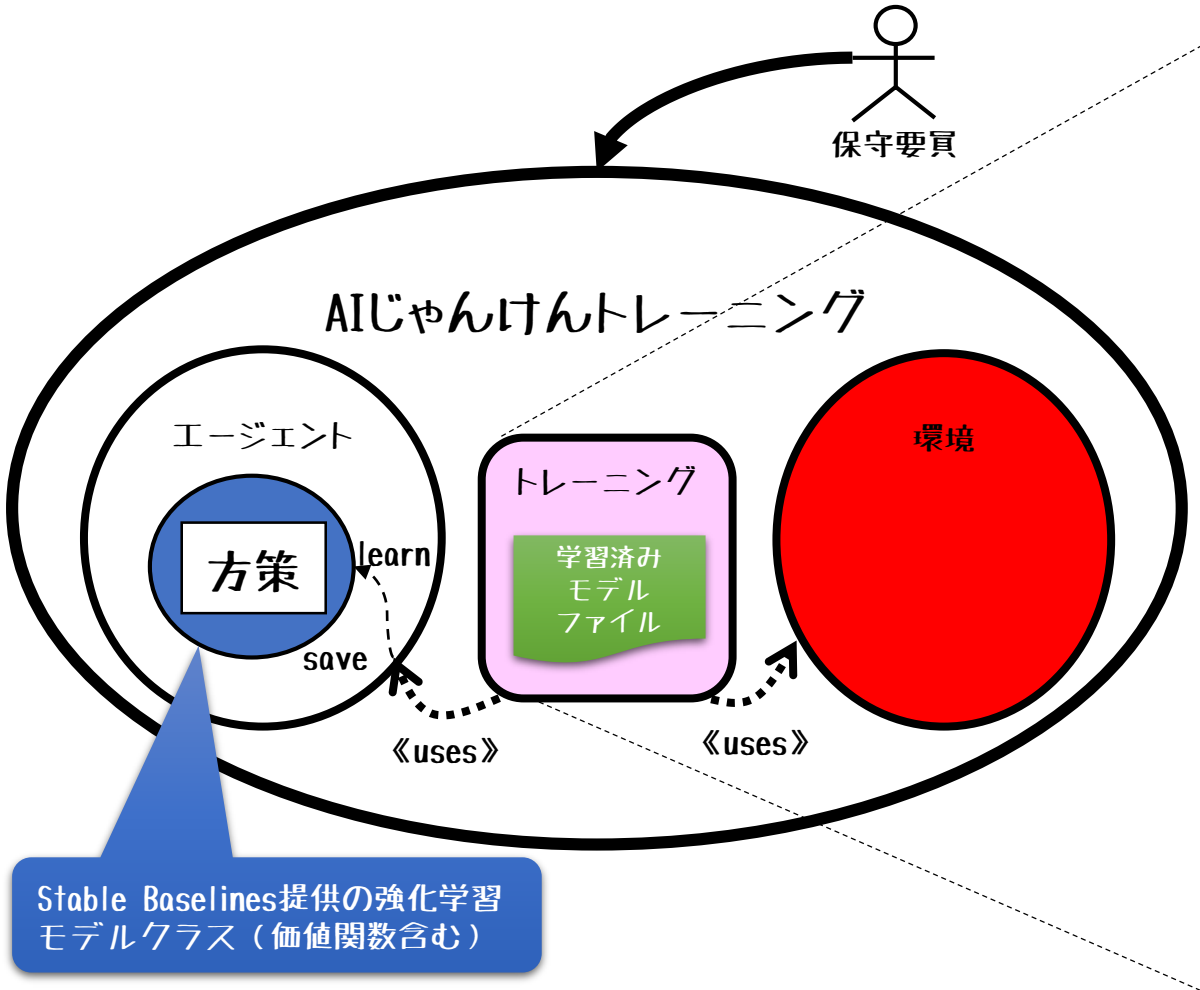
```
《学習済みモデルファイルのロード》
from stable_baselines3 import PPO
:
# 作成済みモデルファイルをロードして学習済みモデルを復元
model = PPO.load('rock_paper_scissors')
:
```

```
《AIじゃんけん側の手を決める》

:
# 変数 observation に実システムの観測が格納済みとする
action = model.predict(observation)[0]
:
```

Stable Baselinesはモデル共通のI/Fがあるため
基本的に上記のコードで対応可能

Stable Baselines使用を決めた段階でほぼ決まる処理②

AIじゃんけんトレーニング実行



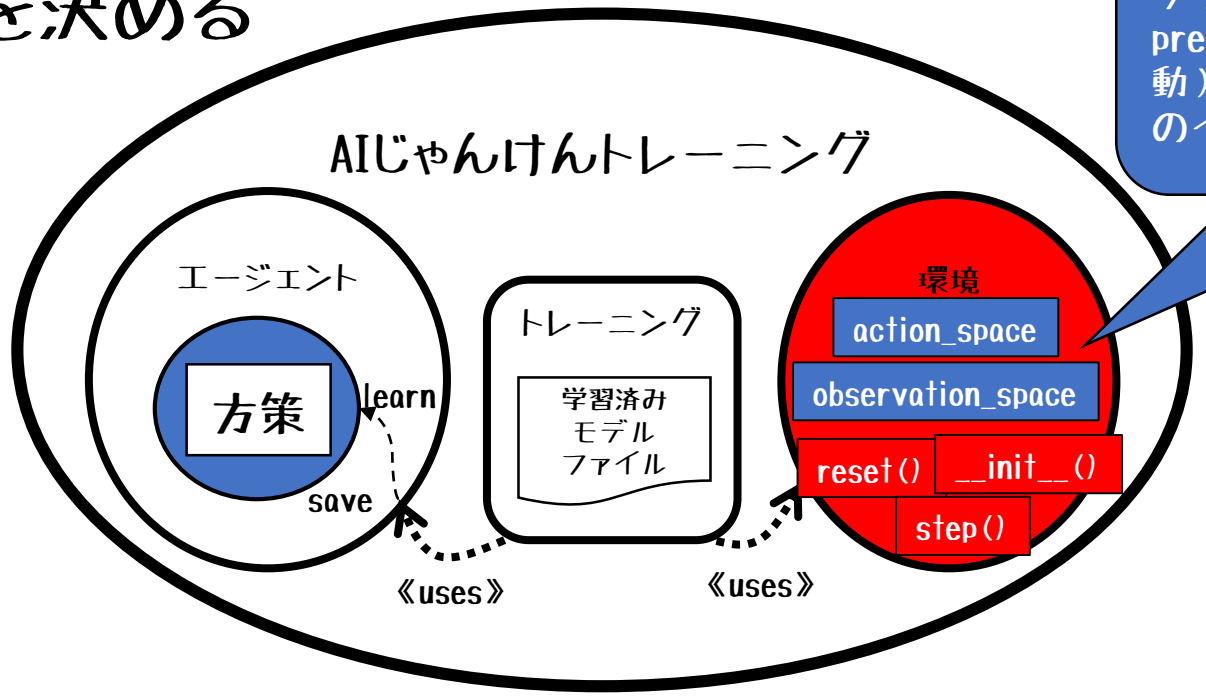
強化学習アルゴリズムPPOを使う場合の実装例

```
《トレーニング処理》
:
# 独自実装したじゃんけん環境の作成
env = RockPaperScissorsEnv(length=10, max_steps=10000)
env = Monitor(env, './logs', allow_early_resets=True)
env = DummyVecEnv([lambda: env])

# Stable Baselines PPO モデルを MlpPolicy指定で作成
# 妥当性検査ON、TensorBoardによる可視化ON
model = PPO('MlpPolicy', env, verbose=1, tensorboard_log=
'./logs')

# トレーニング環境を使ってSACをトレーニング
model.learn(total_timesteps=10000)

# モデルファイルを保存
model.save('rock_paper_scissors')

# 環境を閉じる
env.close()
```

トレーニング処理ステップ数上限。収束しない場合は数値を増やしていく。

設計の流れ

①方策の仕様を決める
②環境の仕様を決める



Stable Baselines提供の強化学習モデルクラスを使う場合、方策側メソッドpredictの引数（観測）・戻り値（行動）の取りうる範囲仕様を環境クラス上のインスタンス変数で行う

方策（の呼び出し方）と環境クラス仕様に絞って解説

①方策の仕様を決める

- ステップ
  - 方策の呼び出しタイミング
- エピソード
  - どの範囲の収益最大化を目標としてトレーニングするか
- 行動
  - 方策の戻り値
- 観測
  - 方策の引数
- observation_space, action_space
  - 観測、行動の取りうる値から gym.spaces クラスを決める
- Stable Baselines3 モデルクラス
  - どのクラスを使うか
  - コンストラクタに指定する方策ネットワークの決定
- 学習済みモデルファイル名

①方策の仕様を決める

ステップ・エピソード

項目	説明	AIじゃんけん仕様
ステップ	方策を１回実行するタイミング	• AIじゃんけん側の手を決定する時 • つまり『ぽん』した時であるため、ステップ間の時間間隔は不定
エピソード （完了条件含む）	• 方策のトレーニングでは収益の最大化をめざす • 収益：エピソードで区切られた範囲の報酬合計 • ユースケースによっては１ステップ＝１エピソードにする場合もある	• 相手と勝負して勝ち負けが決まるまで • ステップ数上限は設けないこととする

ステップの実行間隔は、環境上に時間という概念が必要かどうかに関わってくる
エピソードの終了条件は、どのように方策に学習してもらいたいかを決める項目のひとつ

①方策の仕様を決める

行動・観測

論理仕様は、業務データの項目名で定義する
物理仕様は、intやfloat値のとりうる範囲を含めて定義する

項目	説明	AIじゃんけん仕様
行動	• 方策の戻り値 • 分類の場合0以上の整数 • 回帰の場合1つ以上のfloat値	【論理仕様】 • グー、パー、チョキのいずれかひとつ 【物理仕様】 • int値(0,1,2のいずれか1つ) • 0: グー、1:パー、2:チョキ
観測	• 方策の引数 • 方策を使う側のシステムでどんな情報を方策にあたえられるか • 観測は環境クラス上で選択された行動ごとに遷移する仕様を決める必要があることを踏まえて決定する	【論理仕様】 • 過去100件分のAIじゃんけん側の手とプレーヤーの手の組 【物理仕様】 • (100, 2) 形式のリスト • 1次元：0(最も古い手)～99(前回の手) • 2次元：0=自分、1=敵 • 値：int値(0,1,2のいずれか)

観測の論理→物理が事前処理、行動の物理→論理が事後処理の仕様となる

①方策の仕様を決める

【参考】行動・観測空間定義用クラス

独自環境クラスのプロパティaction_space、observation_space は以下のクラスで定義する

gym.spaces	説明	補足
Discrere	0～n-1までのint値を表現する、分類の場合に使用 【コンストラクタ引数】 n: 分類数 (Discrete(10)の場合0,1,2,3,4,5,6,7,8,9)	行動空間に主に使用される 分類結果を表現する際に使用（Boxを使って確率分布を表現しargmaxで最大値をもとめる方法で代替可能）
Box	要素に数値(型指定可能)の入るリスト、回帰の場合に使用 【コンストラクタ引数】 low: 下限値 high: 上限値 shape: 配列の型(デフォルト:None) dtype: 要素の型(デフォルト:np.float32)	基本Boxで行動空間、観測空間を定義する Boxを行動空間指定すると、Stable Baselines上の実装モデルの選択肢が増える
MultiDiscrete	要素がDiscreteであるリスト 【コンストラクタ引数】 nvec: Discreteのコンストラクタ引数nのリスト (MultiDiscrete([2, 3]の場合[Discrete(2),Discrete(3)])	マルチボタンのジョイスティック入力値などに使用 あまり使用される
MultiBinary	要素がDiscrete(2)であるリスト、多次元配列指定可能 【コンストラクタ引数】 n: リストの型	真偽値の多次元配列を扱う場合に使用

一般的にこの中から決定する

じゃんけんの行動空間

じゃんけんの観測空間

①方策の仕様を決める

observation_space, action_space

プロパティ名	論理仕様	物理仕様
observation_space (観測空間)	【物理仕様】 • (100, 2) 形式のリスト • 1次元：0(最も古い手)～99(前回の手) • 2次元：0=自分、1=敵 • 値：int値(0,1,2のいずれか)	Box(low=0, high=2, shape=(100, 2), dtype=np.int8)
action_space (行動空間)	【物理仕様】 • int値(0,1,2のいずれか1つ) • 0: グー、1:パー、2:チョキ	Discrete(3)

※「行動・観測」より

①方策の仕様を決める

モデルが対応している行動空間

OpenAI Gym マルチプロセスで学習可能

Name	Box	Discrete	MultiDiscrete	MultiBinary	Multi Processing
A2C	✔	✔	✔	✔	✔
DDPG	✔	✖	✖	✖	✖
DQN	✖	✔	✖	✖	✖
PPO	✔	✔	✔	✔	✔
SAC	✔	✖	✖	✖	✖
TD3	✔	✖	✖	✖	✖

ActorCritic ロボティクス

強化学習の基本アルゴリズム

オフポリシ 最新

じゃんけん 行動空間 物理仕様

Stable Baselines3 で使用可能なモデル

※じゃんけんでは、行動空間仕様に合致する最新モデルであるPPOを選択

→ Stable Baselines3 PPOクラスを採用

①方策の仕様を決める

【参考】方策ネットワーク

Stable Baselines3 で使用可能なPolicy

方策名	説明	Stable Baselines3	Stable Baselines
MlpPolicy	MLP（64x2層）を使用して、Actor-Criticを実装した方策	✔	✔
CnnPolicy	CNN（NatureCNN：CNNの後に全結層をつないたもの）を使用して、Actor-Criticを実装した方策	✔	✔
MlpLSTMPolicy	MLP 特徴抽出でLSTMを使用して、Actor-Criticを実装した方策		✔
MlpLnLSTMPolicy	MLP特徴抽出でレイヤ正規化LSTMを使用して、Actor-Criticを実装した方策		✔
CnnLSTMPolicy	CNN特徴抽出でLSTMを使用して、Actor-Criticを実装した方策		✔
CnnLnLSTMPolicy	CNN 特徴抽出でレイヤ正規化LSTMを使用して、Actor-Criticを実装した方策		✔

汎用性が高いのでベースラインとして仕様

主に画像を扱う場合に使用

※じゃんけんの観測では画像を扱わないので、MlpPolicyを選択

→ Stable Baselines3 PPOクラスのコンストラクタには’MlpPolicy’を指定

①方策の仕様を決める

Stable Baselines3 モデルクラス

項目	説明	物理仕様
モデルクラス名	最初に使用するStable Baselines3 モデルクラスについては、行動空間に指定したクラスに対応するStable Baselines3 モデルクラス候補の中から、最新のアルゴリズムを選定する	PPO クラス from stablebaselines3 import PPO
方策ネットワーク	Stable Baselines3 の場合はMlpPolicyかCnnPolicyのどちらかを選択する 観測としてイメージデータを扱う場合はCnnPolicy、それ以外はMlpPolicyを選択する	MlpPolicy
環境	カスタムSIの場合、独自環境クラスを使用するため、「②環境の仕様を決める」にて確定するが、とりあえずクラス名だけはここで決めておく	RockPaperScissorsEnv
妥当性ログ	トレーニング(learn())実行中標準出力に妥当性ログを出す場合1、出さない場合は0	1
TensorBoardログ	Stable Baselines3のトレーニングの可視化にTensorBoardを使う場合はディレクトリパスを指定	‘./logs’

※上記仕様は、Stable Baselines3 モデルクラスのコンストラクタ引数となる

```
model = PPO('MlpPolicy', RockPaperScissorsEnv(), verbose=1, tensorboard_log= './logs’)
```

①方策の仕様を決める

学習済みモデルファイル名

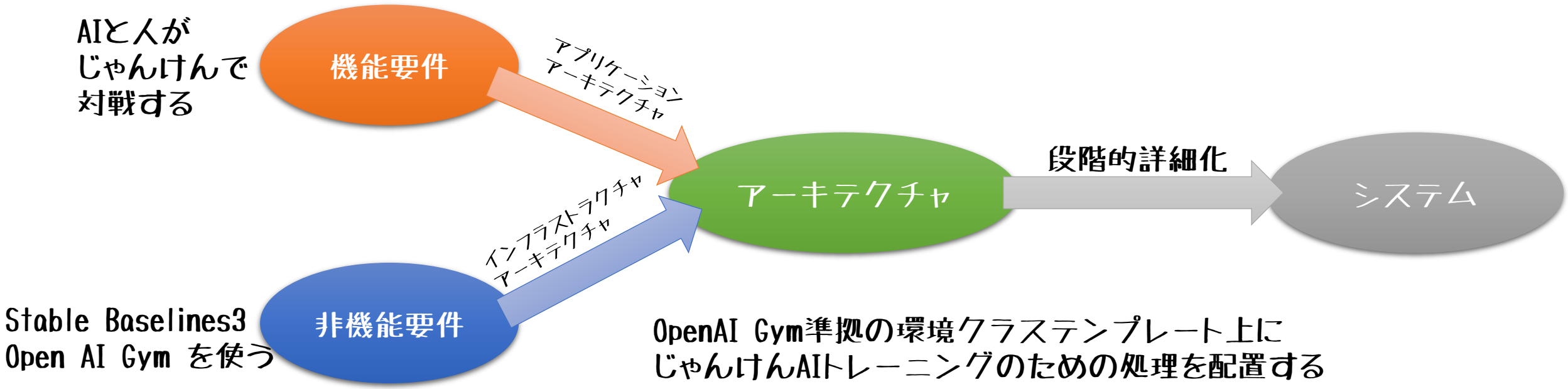
項目	説明	物理仕様
学習済みモデルファイル名	カスタムSIの場合、プロジェクトルールに準拠したファイル名にする	‘rock_paper_scissors’ ※実際のファイル名は拡張子.zipがつく

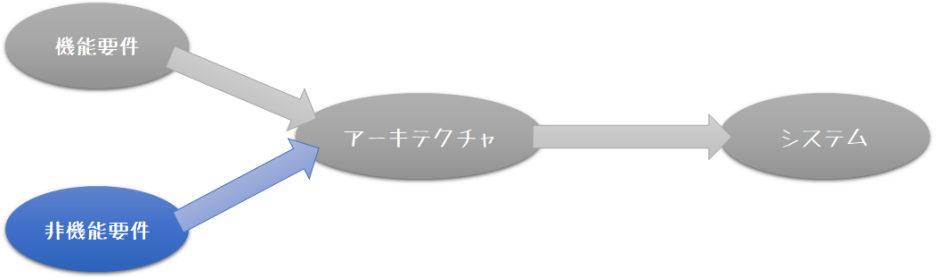
※上記仕様は、Stable Baselines モデルクラスsave()の引数となる

※トレーニング処理のtotal_timestepsは、10000をデフォルトにしてトレーニングを試行して調整していく

```
# トレーニング環境を使ってSACをトレーニング
model.learn(total_timesteps=10000)

# モデルファイルを保存
model.save(‘rock_paper_scissors')
```

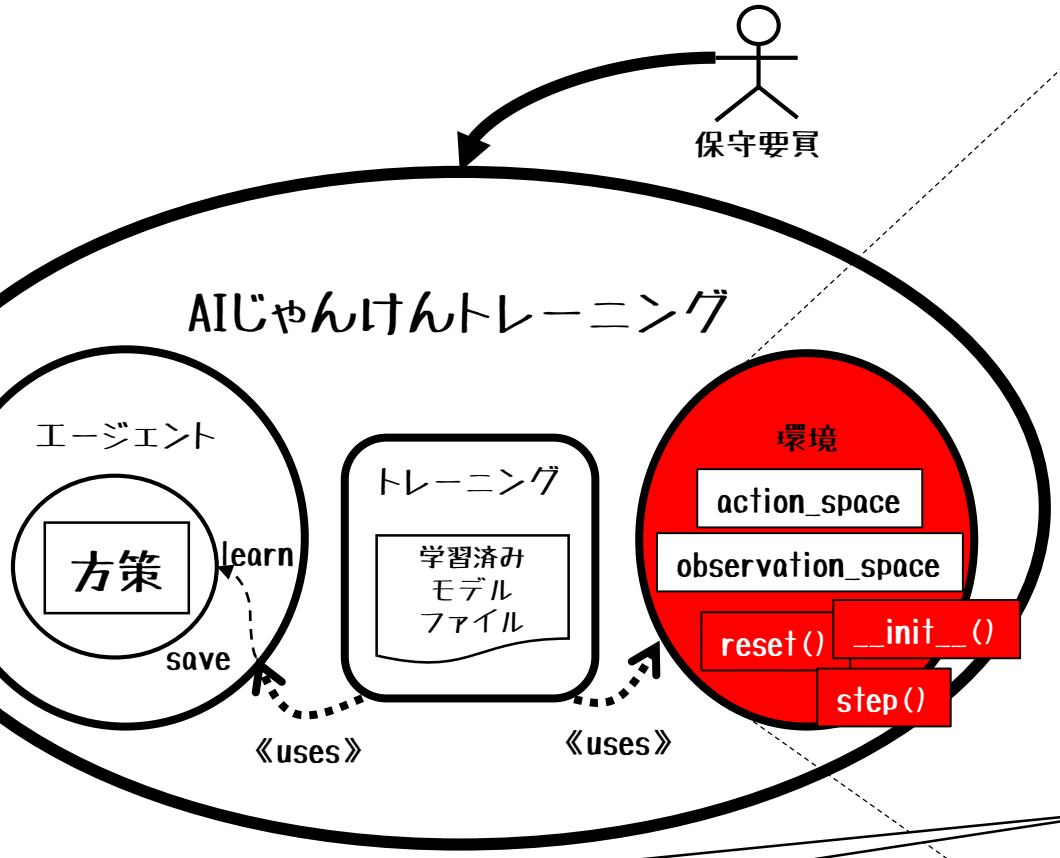
②環境の仕様を決める





②環境の仕様を決める

環境クラス テンプレート

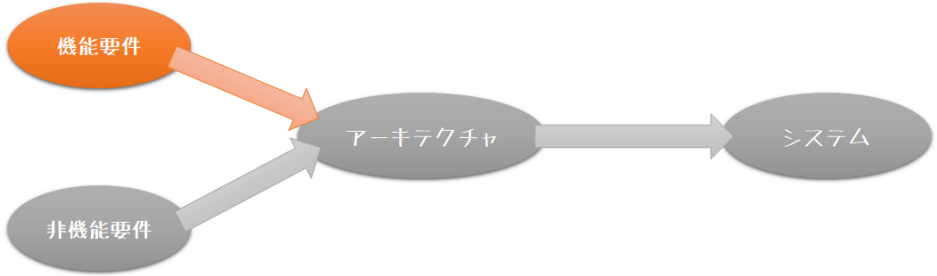


決定した環境仕様をもとに
テンプレート上のコメント部を埋める

```
import gym
from gym import spaces
import numpy as np

class RockPaperScissorsEnv(gym.Env):

  def __init__(self):
    super().__init__()
    self.observation_space = spaces.Box(
          0, 2, (2, 100,), dtype=np.int8)
    self.action_space = spaces.Discrete(3)
    # 状態の初期化
    # 戻り値なし

  def reset():
    # エピソード開始時の観測を返却する
    return None

  def step(action):
    # 方策が選択した行動を引数として受け取る
    # 観測、報酬、エピソード完、その他情報を返却
    return None, None, None, {}
```

②環境の仕様を決める

じゃんけんAIトレーニング



【ミニバッチ毎】
更新
方策上の
パラメータを更新する

【エピソード開始】観測
過去100見聞の両者の手
（ランダムに作成）

《観測》
過去100件分の
両者の手

じゃんけん審判

《方策》
AIプレイヤー
観測をもとに
次の手を決定

敵プレイヤー
確率1/3で
ランダムに手を決定する

対戦

《観測》
過去100件分の
両者の手

《報酬》
勝:+10、負:-10、引分:-1

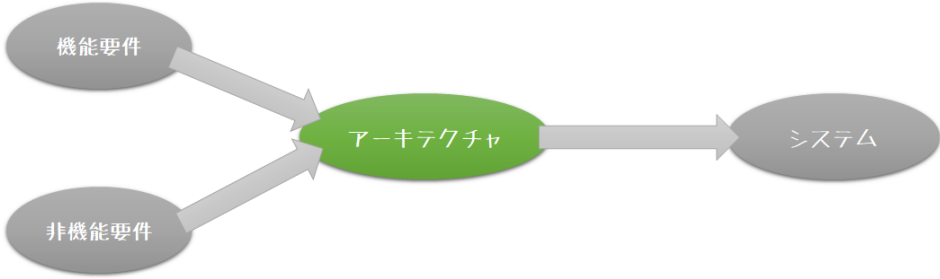
《エピソード完》
勝:True、負:False、引分:False

WIN!

敵プレイヤーの手をどう
決定するかの決めるの

②環境の仕様を決める

コンポーネント設計



《stable_baselines3》
PPO

①で確定済

【エピソード開始】観測
過去100見聞の両者の手
（ランダムに作成）

《観測》
過去100件分の
両者の手

じゃんけん審判

《gym.Env》
RockPaperScissorsEnv

《uses》

《uses》

EnemyPlayer

敵プレイヤー

対戦

《観測》
過去100件分の
両者の手

《報酬》
勝:+10、負:-10、引分:-1

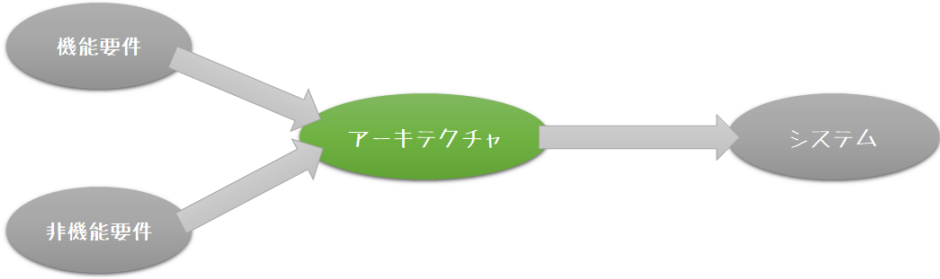
《エピソード完》
勝:True、負:False、引分:False

predict()
(PPOクラスと同じシグニチャ)

定する

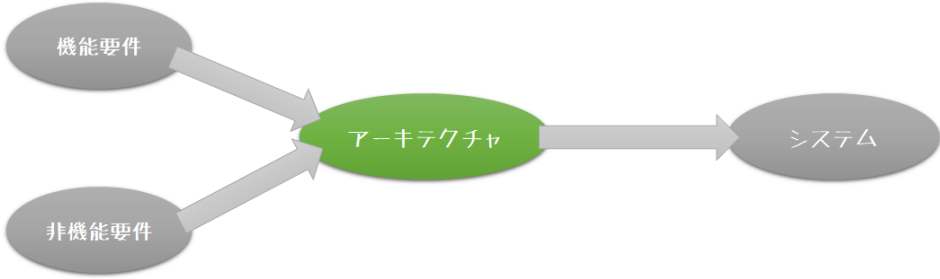
②環境の仕様を決める

EnemyPlayer



Stable Baselines3 PPOクラスのじゃんけんの敵として手を生成するクラス
OpenAI Gym 準拠ではない通常のクラスとして設計
環境クラスRockPaperScissorsのコンストラクタの引数にインスタンスを指定して使う

メソッド名	引数	戻り値	説明	処理仕様
__init__()	prob_list=[0.33, 0.33, 0.34]	N/A	グー・パー・チョキを出す確率をインスタンス変数に格納するリストは長さ3で、グーの確率、パーの確率、チョキの確率がfloat値で格納されている状態とする	prob_listの比率を変えずに、合計値が1となるように変換インスタンス変数に変換後の値を格納する
predict()	observation	action	敵プレイヤーの次の手を返す引数、戻り値はStable Baselinesモデルクラスと同じ	0から1までのfloat値乱数を1つ取得する 【0以上prob_list[0]未満の値】 0を返却 【prob_list[0]以上(prob_list[0]+prob_list[1]未満の値】 1を返却 【上記以外】 2を返却

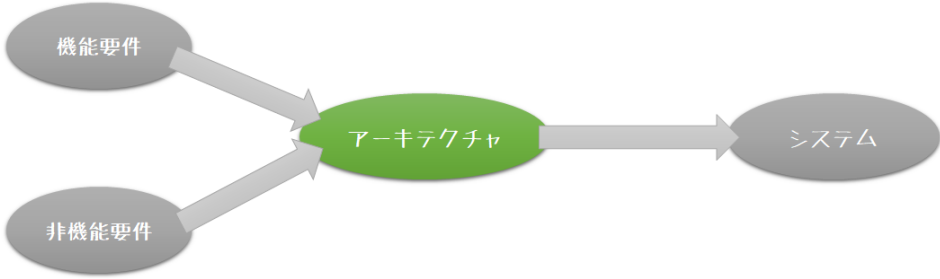


②環境の仕様を決める

RockPaperScissorsEnv 1/2

テンプレートメソッドの仕様を確定させる
じゃんけん環境では「状態＝観測」として設計
エピソード完了判定関数is_done()、報酬関数calc_reward()は非テンプレートメソッド

メソッド	引数	戻り値	説明	処理仕様
__init__()	player	N/A	インスタンス変数action_spaceを定義する インスタンス変数observation_spaceを定義する 状態を初期化する	引数player：EnemyPlayerインスタンス ※action_space，observation_space定義は省略 引数playerをインスタンス変数へ格納する 要素としてランダムに0から2までの値を格納した(100，2)形式のリストをインスタンス変数observationへ格納する
reset()	N/A	observation	インスタンス変数上の状態をエピソード開始時まで初期化する 状態をもとに環境を生成する	インスタンス変数observationをそのまま返却する ※過去の手は継続して使用する仕様とし、初期化しない
step()	action	observation reward done info	ステップごとに呼び出される 行動を受け取り観測、報酬、エピソード完、その他情報を返却する 観測：選択された行動によって変化した観測 報酬：報酬をあらわすfloat値 done：エピソード完了したかどうかを表すBoolean info:その他情報、型は各自で決めて良い(NoneでもOK)	インスタンス変数playerのpredictを呼び出し、敵の手を取得する（引数はインスタンス変数observationを使用） エピソード完了判定関数is_done()を行い結果を真偽値doneへ格納する 報酬関数calc_reward()を実行し、報酬をrewardへ格納する 引数action値と敵の手をインスタンス変数observation末尾に追加する インスタンス変数observation、reward、done、Noneを返却する



②環境の仕様を決める

RockPaperScissorsEnv 2/2

エピソード完了判定と報酬関数は別メソッドに切り出し

メソッド	引数	戻り値	説明	処理仕様
id_done()	my_action enemy_action	done	エピソード完了判定処理を実装する ※env.Gymテンプレートメソッドではない	以下のロジックでエピソード完了判定結果を算出し返却する 【my_action==enemy_action】done: True 【else】 done: False
calc_reward()	my_action enemy_action	reward	報酬関数を実装する ※env.Gymテンプレートメソッドではない	以下のロジックで報酬を算出し返却する 【my_action===enemy_action】報酬：-1 【my_action==0】 　【enemy_action==1】報酬：-10 　【else】報酬：+10 【my_action==1】 　【enemy_action==2】報酬：-10 　【else】報酬：+10 【else】 　【enemy_action==0】報酬：-10 　【else】報酬：+10

②環境の仕様を決める

【参考】gym.Env テンプレートメソッド

	名称	定義・引数	戻り値	説明
必須	《プロパティ》 action_space	gym.space オブジェクト	N/A	行動空間（方策の戻り値）
必須	《プロパティ》 observation_space	gym.space オブジェクト	N/A	観測空間（方策の引数）
	《プロパティ》 reward_range	(float, float)型タプル	N/A	報酬値の定義域、基底クラスで(-inf, +inf)が定義されている
必須	《メソッド》 reset()	なし	observation：観測	エピソード開始時の観測データを返却するメソッドを実装
必須	《メソッド》 step()	action:行動（方策が選択した行動）	observation: 観測 reward:報酬値 done:エピソード完 info:その他情報	方策がとった行動により変化した観測、得られた報酬値、エピソード完了したか、その他情報を返却するメソッドを実装
	《メソッド》 render()	mode:対象をさす文字列(プロパティmetadataの要素)	任意：たとえばmode=‘rgb’の場合、イメージデータ(np.array())を指定する	現在の環境を可視化する場合に実装
	《メソッド》 seed()	seed:型定義なし	int値のリスト 乱数ジェネレータが使用するシード値リスト	環境内で使用する乱数のシードを固定したい場合などに実装
	《メソッド》 close()	なし	なし	環境の仕様終了時処理

問題：「じゃんけん」を実装せよ

- Python3.x、OpenAI Gym、Stable Baselines3(PyTorch)を使用すること
- 各自のPC上で実装する（Python開発環境を整える）
- わからない場合は自分で調べること

自分で設計・実装してみよう

※自分で実装することで、成り行きベースの「見積」ができるようになります！

《実装サンプル》
https://github.com/coolerking/rock-paper-scissors

問題を解くためのヒント

- 市販書籍「OpenAIGym/Baselines深層学習強化学習人工知能プログラミング実践入門」（税抜3,200円）
  - 本セッションの内容を復習できる
- Pythonがわからない/書けない場合
  - JavaやJavaScriptが書けるなら、参考書片手で実装可能
  - リスト、辞書、関数、クラスがわかっていればなんとかなる
  - Python開発環境に関する知識(pip, virtualenv, anaconda, pyenv, Jupiter notebook/Google Colab)
  - 開発言語が初めての人は市販書籍やオンライン講座でまず自習

その他の参考文献(有料)

- Udemy講座「現役シリコンバレーエンジニアが教えるPython3入門+応用+アメリカのシリコンバレー流コードスタイル」
  - Pyhon環境の作り方から何をシていいかわからない人向け
  - 全セクション学習しなくても良い、セクション1~7までは受講推奨
- Udemy講座「【4日で体験しよう！】 TensorFlow, Keras, Python 3 で学ぶディープラーニング体験講座」
  - 強化学習の基本「Q学習」をTensorflowで実装できる
  - p149まで読めばStable Baselinesによる実装まで可能
  - OpenAI Gym/Stable Baselinesでの実装を体験してから受講を推奨
- 書籍「Unityではじめる機械学習強化学習Unity ML-Agents実践ゲームプログラミングv1.1対応版」
  - Unityゲーム上のAIプレイヤーを作ることができる
  - シミュレータ≠環境 であることが身を持って体験できる
- Udemy講座「Reinforcement Learning: AI Flight with Unity ML-Agents」（英語）
  - 上記書籍では扱わなかったBlenderを使ったモデリングから学べる
  - フライトシミュレータゲームのAIプレイヤーを作成できる

じゃんけんAIを実装した人へ

あれ？

- 自分で実装した人は、おそらく「あれ？」と思うと思います
- ここに書かれた仕様のまま実装すると、簡単に勝つことのできるAIになっているでしょう
- では、どのように変更すれば、より人に勝てるAIになるのか？

- ヒント
  - 強化学習の目的は「収益の最大化」にある
  - 人にAI側の戦術がばれないようにするには、どこをまず変更したほうが良いのか

動作するだけなら作るのは難しくないが、精度を上げるのは大変

トラブルシューティング 勝手私案

※環境にバグがないという前提とする

問題	分析	解決案
意図した行動が得られない	・平均報酬値が低い ・収益値が低い ・システム使用感が良くない ・うごくけど、動作が..	定量評価指標の確定、目標値の決定後、以下の順番でトライアンドエラー ①報酬、報酬関数の見直し ②状態・観測の見直し ③行動の見直し ④ハイパーパラメータの見直し ⑤他のアルゴリズムに変更
学習時間が長い	・学習が収束しない、終わらない ・もっと短くしたい	定量評価指標の確定、目標値の決定後、以下の順番でトライアンドエラー ①他の（オンポリシー系）アルゴリズムに変更 ②ハイパーパラメータを調整 ③報酬、報酬関数の見直し ④状態・観測の見直し ⑤行動の見直し

データベースのパフォーマンスチューニングと同じで、結局トライアンドエラーによる解決策模索となる

まとめ

カスタムSIに適用する際の注意

- 状態（観測）の定義と報酬関数の発明が鍵
  - 教師あり学習以上の機能要件把握が必要
  - 画像を扱う場合はUnityなど画像生成系の実装手段が必要
- 強化学習向きの機能の切り分け
  - 業務データ（状態・観測）にノイズが含まれている
  - 人間は処理できる（行動の決定）が、アクティビティ図化できない
  - 学習データの準備ができない
  - 良い行動・悪い行動を人間は判定でき、アクティビティ図化できそう
- アルゴリズムの勉強は、先に１つ動くものを作ってから
  - 「どうしてうまく行かないのか」が学習動機の場合、理解が早くなる
  - ハイパーパラメータの意味→具体的なアルゴリズムの構造

まとめ

ロール別まとめ

《プロジェクトマネージャ》
メンバ育成/データ準備をPMPに組み込む
育成コスト（書籍代や研修費用含む）の確保

《アーキテクト》
強化学習前提のアーキテクチャ設計の確立
強化学習を前提とした開発手順の組み立てる

《アプリケーション設計エンジニア》
いち早く見積もり可能に
実装まで完了させて勘所をいち早く掴む

《運用設計エンジニア》
モデル更新前提運用システム設計
学習済みモデルファイルの安全な更新手段の設計

《コンサルタント/営業》
実現可能なケースの模索
どのケースなら自組織で設計・実装できそうか

《なりたてエンジニア》
方向性を決めていないなら、強化学習スペシャリストを目指すのも良いかも
メンバシップ型からジョブ型へ、ジェネラリストからスペシャリストへ

まとめ

with コロナ時代のスキルアップには

自習力

がものをいう..かも

おわり

補足

誤った報酬関数を使った場合

https://openai.com/blog/faulty-reward-functions/ より引用

補足

報酬の仕様決めは、意外と難しい

例題：ゴルゴ13の”報酬”を考えよう

- 用件受諾(エピソード開始)時に前金
- 行動中は報酬なし
- 完遂(エピソード完了)時、残金を入金
- 未完遂の場合、返金は俺ルールに従う
  - ターゲットを先に殺害された場合、前金を返すこともある
- 用件説明時虚偽があった場合、返金なし(依頼者は殺害される)

そもそも誰得の報酬なのか？
ゴルゴにとって？依頼者にとって？

報酬を「金額」と仕様を決めるのは誤り

※回答例は用意してません

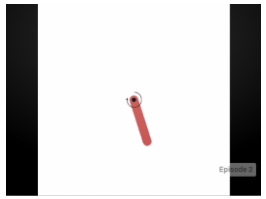
補足

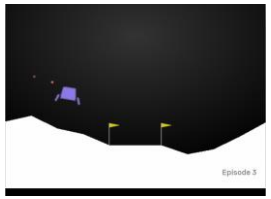
OpenAI Gym 環境クラス

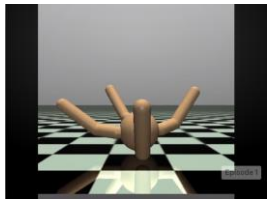
- トレーニングに使用する本番環境に代替するクラス
  - 状態・観測の漸化式
  - 報酬関数、エピソード完了判定
  - 可視化
- 実装済み環境クラスを提供（以下、一部）

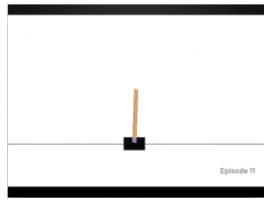
※https://gym.openai.com/ より引用


SpaceInvaders-v0


Pendulum-v0


LunarLander-v0

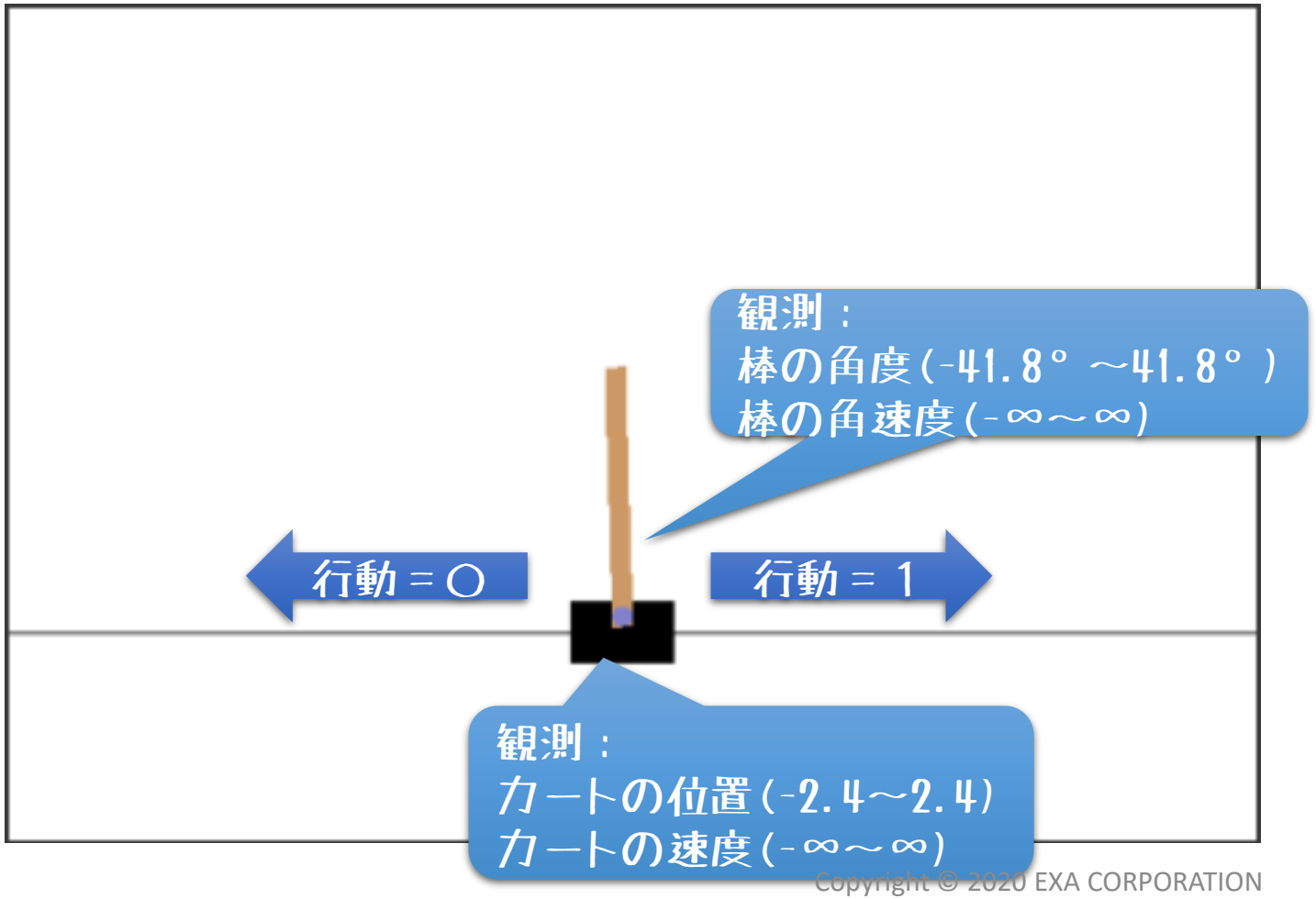

Ant-v2


CartPole-v1

  - 画面のない環境クラスあり
- カスタム環境クラスも実装可能

OpenAI Gymが提供する環境クラスは独自モデルの研究目的であるため
業務システム用の環境クラスは個別に実装する必要あり

【参考】OpenAI Gym 環境 CartPole-v0



エピソード：棒が倒れるまで
ステップ：一定時刻間隔

エピソード完了判定：
棒の角度が範囲(-41.8° ～41.8°)を
超えた場合

※状態＝観測

補足

独自の方策ネットワークを使う

※Stable Baselines利用前提

- Stable Baselines にはカスタム方策を採用するI/Fが提供されている
  - 詳細は公開ドキュメント参照のこと
- policy_kwargs を使う
  - 方策ネットワークの内部構造を変更したい場合
- カスタムFeaturesExtractor
  - レイヤ構成自体を独自に実装したい場合
  - 基底クラスBaseFeaturesExtractorを継承したクラスをpolicy_kwargsに指定
- カスタムPolicy
  - 独自の方策・価値関数を定義したい場合
  - 基底クラスActorCriticPolicyを継承したクラスを実装
  - Stable Baselinesモデルクラスコンストラクタに方策ネットワークとして指定

強化学習アルゴリズムの知識が必要

補足

用語集 1

- 状態価値関数 V(s)
  - 引数：状態、戻り値：価値
  - ある状態の価値を計算する関数、価値観数の一種
  - ある状態でとった行動が有効であった場合価値を上げるように更新していく
- 行動価値関数 Q(s, a)
  - 引数：状態、行動、戻り値：価値
  - ある状態にて、ある行動をとったときの価値を計算する関数、価値観数の一種
  - 単純な実装例では、Qテーブル（行：行動、列：状態、値：価値）で表現することが多い
  - Qテーブルの多変量化したものを扱うために深層学習モデルを使うようになった

用語集 2

- Actor-Critic
  - 学習器の構造を指す用語
  - Actor
    - 環境に対してactする→方策
  - Critic
    - Actorのとった行動をTD誤差などでcriticize(評価)する→価値関数
- TD誤差
  - Temporal Difference 誤差
  - 行動前の評価値と行動後の評価値の誤差を指す
  - SarsaやQ学習では行動価値関数の更新に使用されている

用語集 3

- Curiosity
  - エージェントに好奇心をもたせ、エージェントが未知の状態への探索を選択するように促す学習方法
  - BCと併用して効果を発揮した事例あり
- セルフプレイ
  - テニスやサッカーなどの対照的で敵対的なゲームにおいて、エージェントの現在および過去の「自分」を対戦相手とすることで、より優れたエージェントを育てる学習方法

用語集 4

- カリキュラム学習
  - 学習環境の難易度を徐々にもたせ、より効率的な学習を実現する学習方法
- 環境パラメータのランダム化
  - 環境にバリエーションをもたせることで、エージェントが将来の環境の変化に適応できるように訓練する学習方法
- マルコフ性
  - 将来の状態の条件付き確率分布が、現在の状態のみに依存
  - 過去の状態に依存しない
  - 現在の状態から将来の予測報酬値→価値を算出する関数を実装

用語集 5

- 探索
  - 二人ゼロサム有限確定完全情報ゲームで使用される手法
    - わかりやすく言うと、囲碁やオセロ
  - “先読み”
  - モデルベースの強化学習
    - モデル内で最適な行動パスを探す際に使用する探索アルゴリズム
  - 探索アルゴリズムにも種類がある
    - ミニマックス法、アルファベータ法、原始モンテカルロ木探索、モンテカルロ木探索など
      - AlphaZeroではモンテカルロ木探索を採用

補足

方策を更新する手法

《方策反復法》

方策に従って行動させ、その結果成功した行動を重要視
できるだけその行動を選択させるように取り入れる

《方策勾配法》
Qテーブルをそのまま使用
ε-greedyを併用する事が多い

状態価値関数を主に使用することが多い

《価値反復法》

現在の状態価値と次の状態価値の差分を算出
その差分だけ現在の状態の価値を増やすように更新する

《Salsa》
TD誤差にQテーブルの値をそのまま使用

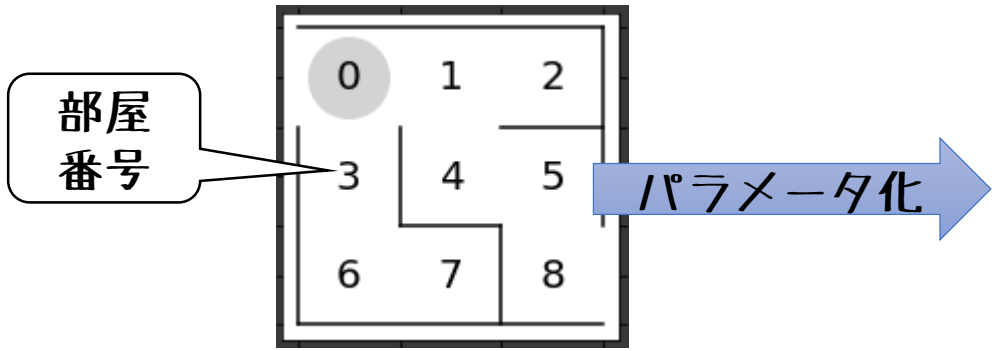
《Q学習》
TD誤差にQテーブルの
該当状態行の最大値を使用

行動価値関数を主に使用することが多い

補足

Qテーブル

- 行：状態、列：行動　である行列
- 要素は用途によって異なる



《状態》

《行動》	上	右	下	左
部屋番号0	nan	1	1	nan
部屋番号1	nan	1	1	1
部屋番号2	nan	nan	nan	1
部屋番号3	1	nan	1	nan
部屋番号4	1	1	nan	nan
部屋番号5	nan	nan	1	1
部屋番号6	1	1	nan	nan
部屋番号7	nan	nan	nan	1

移動可能

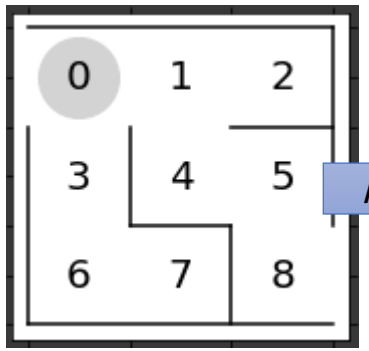
移動不可

３×３迷路脱出ゲームにおけるQテーブルの例

補足

方策パラメータ θ

- 方策が使用するパラメータ、構造はアルゴリズムにより様々
- 状態から行動を選択する際に使用する
- 機械学習強により更新される対象



パラメータ化

《行動》

《状態》	上	右	下	左
部屋番号0	nan	1	1	nan
部屋番号1	nan	1	1	1
部屋番号2	nan	nan	nan	1
部屋番号3	1	nan	1	nan
部屋番号4	1	1	nan	nan
部屋番号5	nan	nan	1	1
部屋番号6	1	1	nan	nan
部屋番号7	nan	nan	nan	1

パラメータ化

現在の状態

方策

《行動》

《状態》	上	右	下	左
部屋番号0	0	0.9	0.1	0
部屋番号1	0	0.3	0.6	0.1
部屋番号2	0	0	0	1
部屋番号3	0.8	0	0.2	0
部屋番号4	0.2	0.8	0	0
部屋番号5	0	0	0.9	0.1
部屋番号6	0.9	0.1	0	0
部屋番号7	0	0	0	1

確率

次の行動

ステップ1	部屋番号0	右
ステップ2	部屋番号1	下
ステップ3	部屋番号4	右
ステップ4	部屋番号5	nan

エピソード内の行動履歴

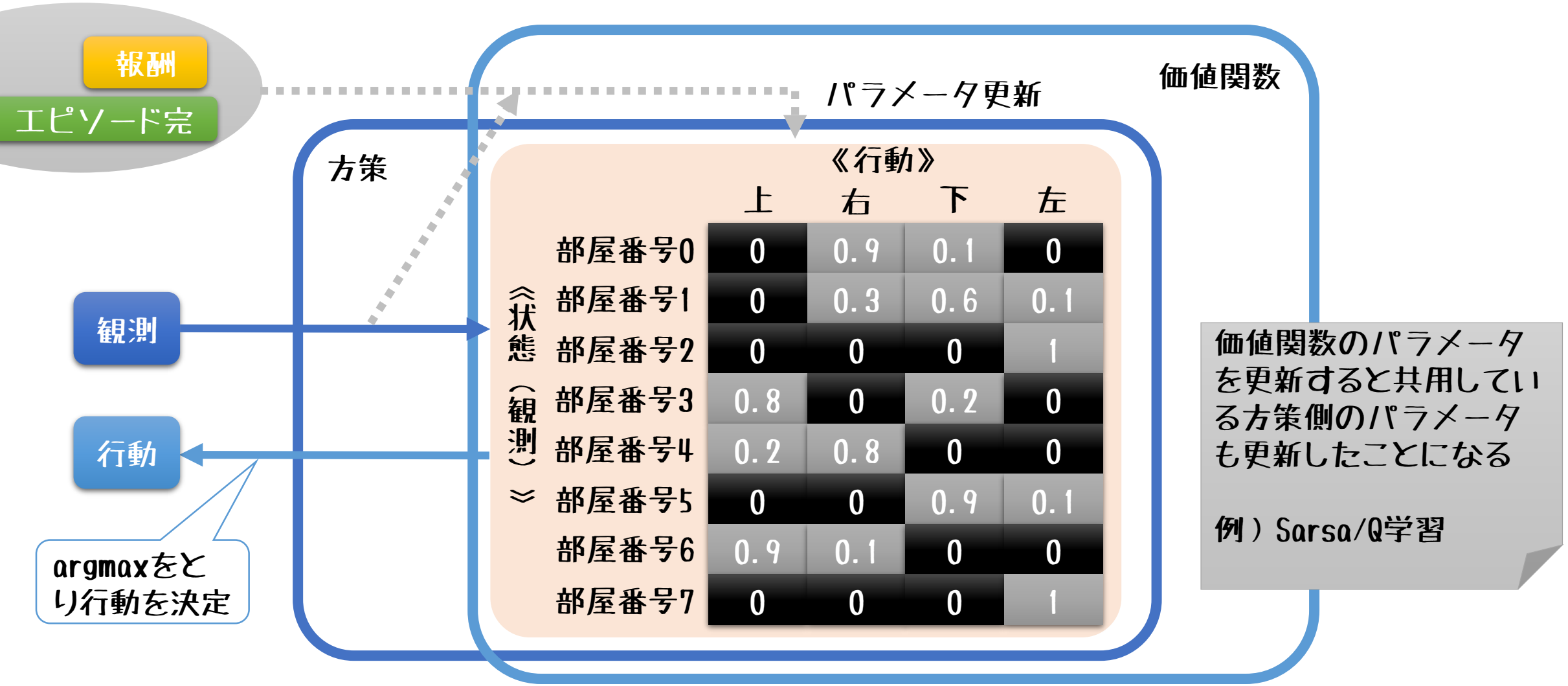
履歴に従い更新

《方策勾配法》
履歴にしたがって更新する際にソフトマックス関数を使う

3×3目迷路ゲームにおける方策パラメータ例

補足

方策・価値関数で同じパラメータを使う



補足

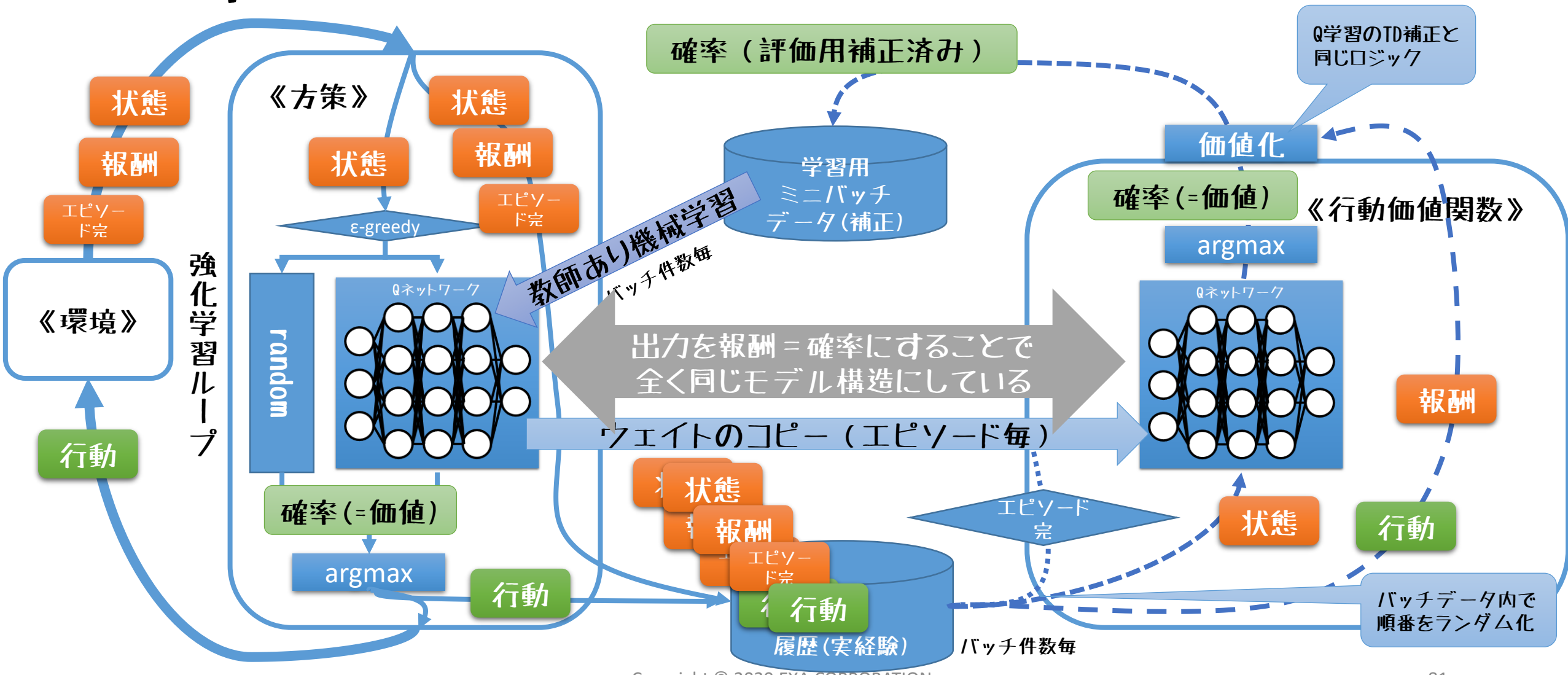
Deep Q Network

- Q学習におけるQテーブルを深層学習に変更
- 入力：状態、出力：行動
- 深層学習モデル
  - 教師あり学習モデルと同じ構造(多層パーセプトロン、CNN、LSTM、…)
- Experience Replayed
  - ミニバッチの各行をランダムに学習させる
- Fixed Target Q-Network
  - 方策側と全く同じ構造で行動価値関数側のモデルを用意
  - ミニバッチ単位で方策側に学習済みパラメータを反映
- Reward Clipping
  - 報酬スケールを-1、0、1に固定
- Huber Loss
  - 誤差関数に平均二乗法のかわりにHuber関数を使用

Stable Baselinesでは
DDPG、TD3、SACがDQNの子孫にあたる

補足

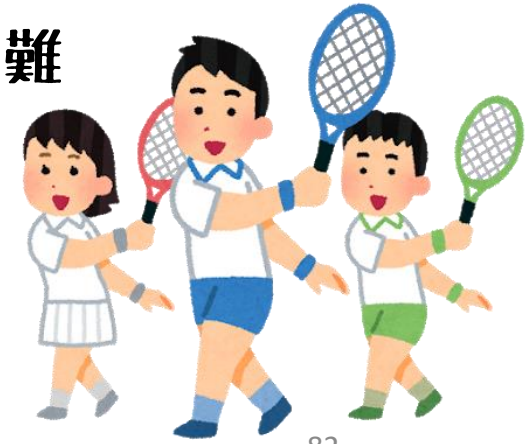
Deep Q Network のトレーニング



補足

模倣学習 Imitation Learning

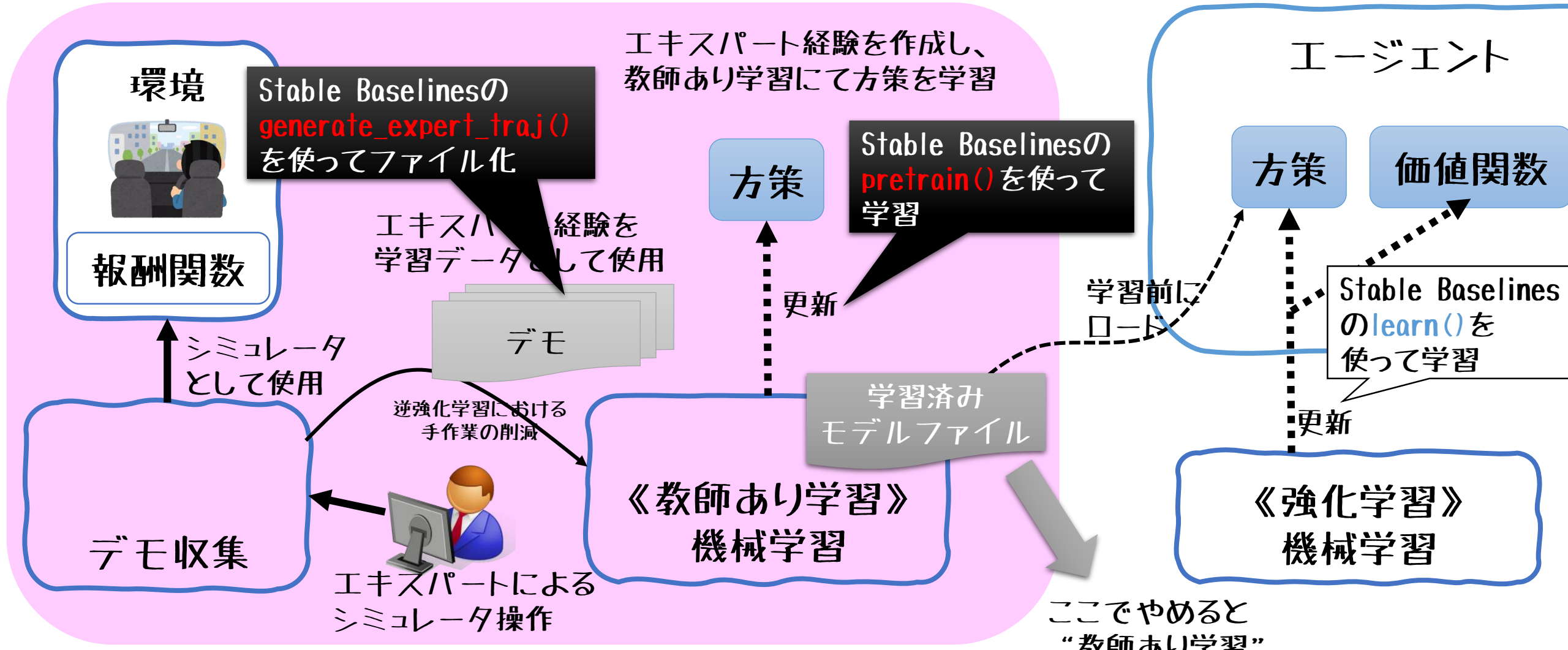
- デモ
  - 熟練者などの教師となる他者（エキスパート）による一連の行動
- 教師となる他者の行動を真似るように学習すること
- 適用前提が限られる
  - 教師となる他者の存在が必須
  - 教師となる他者の行動を可能な限り正確に数値化が難しい
  - デモの要素の定義が困難→観測空間・行動空間の定義が困難



補足

模倣学習を取り入れた強化学習アルゴリズム

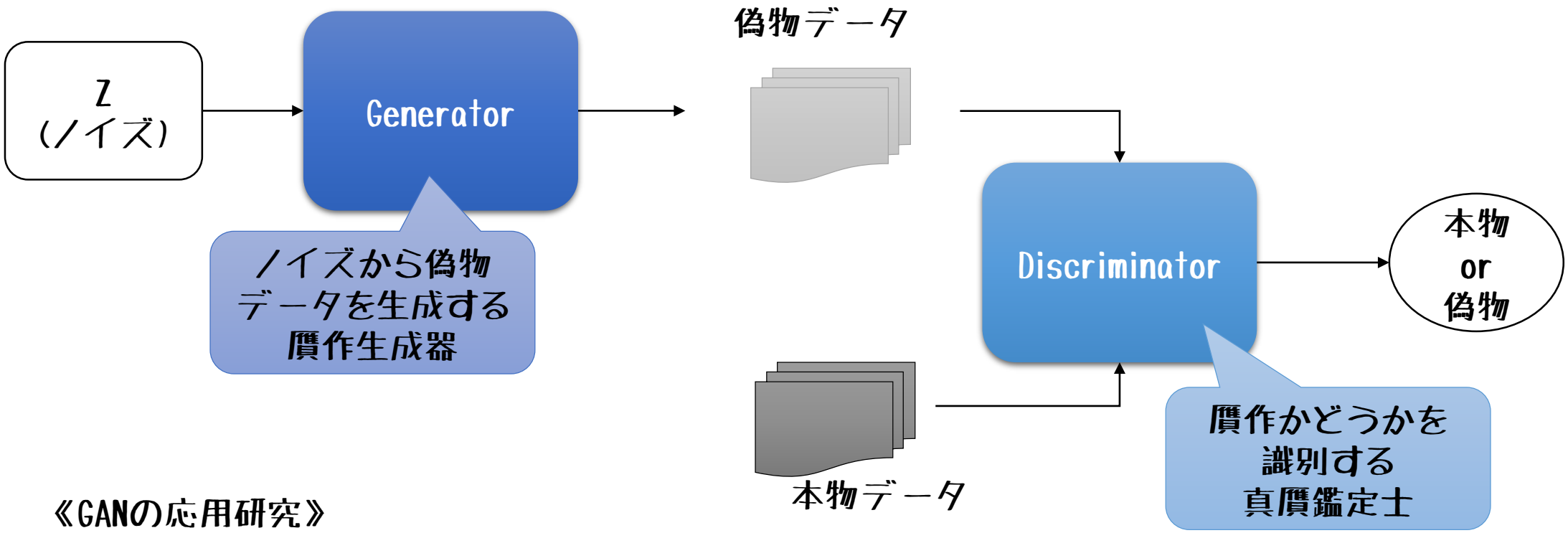
BC: Behavior Cloning



補足

贋作画像生成で有名になった教師あり学習アルゴリズム

敵対的生成ネットワーク(GAN)



《GANの応用研究》

DCGAN:画像認識で有名なCNN技術を応用

CycleGAN：無関係な2枚の写真を使い互いに変換し合うようにサイクル上のネットワークを構築

StyleGAN:入力されるノイズを一旦別の空間でマッピングしそこで得られた情報をGeneratorへ入力

補足

逆強化学習

- 報酬関数をエキスパートデータを用いた教師あり学習で推定
- 報酬関数を教師あり学習アルゴリズムのいずれかで実装
- 教師あり学習のスキームで機械学習を実行
- 学習データとしてエキスパートデータを使用
  - 熟練者の操作データに報酬値を個別に付加
- 学習済みモデルファイルをロードした報酬関数を使って強化学習のスキームで機械学習を実行し、方策を学習させる
- エキスパートデータ作成には手作業が必要となる

活用例：論文「深層強化学習による自動運転の安心走行実現」

補足

模倣学習・逆強化学習・GANを取り入れた強化学習アルゴリズム

GAIL:Generative Adversarial Imitation Learning