卒業研究報告書

題目

ディープラーニングにおけるゲーム AI 開発

指導教員

石水 隆　講師

報告者

13-1-037-0130

石田 祐也

近畿大学理工学部情報学科

平成 30 年 1 月 30 日提出

概要

今日、人工知能の分野ではディープラーニングが注目されている。ディープラーニングとは、システムが人間の力を借りず､データの特徴を学習し事象の認識や分類を行う機械学習の手法の１つである。この手法は画像認識や音声認識など様々な分野で利用されているが､最近では囲碁や将棋などといったテーブルゲームに利用されている。実際､この手法で作成された囲碁プログラム AI はプロの棋士に勝つほどのレベルに達している。そこで、本研究ではテーブルゲームに注目し他のゲームにディープラーニングを応用できるかを検証する。

本研究では、立体三目並べというテーブルゲームにおいて python 言語でディープラーニングのゲーム AI を作成し、他のゲーム AI と対戦させる。そしてある程度の勝率を目指し、このゲームにおいてディープラーニングの有用性を検証するものである。

目次

1　序論

1.1 本研究の背景

今日、人工知能の分野ではディープラーニングが注目されている。ディープラーニングとは、システムが人間の力を借りず、データの特徴を学習し事象の認識や分類を行う機械学習の手法の１つである。この手法は様々な技術革新を起こしている。例えば、画像認識における性能の大幅の向上、車の操縦や制御を支援する運転支援AIなどがある。また最近では、将棋や囲碁などのテーブルゲームのAIにも利用されており、プロの棋士に勝つほどのレベルに達している。このように、ディープラーニングは様々な分野、業界において技術革新を起こしているのである。

本研究では、テーブルゲームに重きを置いて研究を進めていく。

1.2 ディープラーニングとは

ディープラーニングとは、ニューラルネットワークを利用した機械が物事を理解するための学習方法である。

1958 年にアメリカのローゼンブラットによりパーセプトロンと呼ばれるニューラルネットワークが開発された[6]。パーセプトロンの例を図 1 に示す。パーセプトロンは入力層と出力層のみで構成されており出力値は 0 か 1 である。しかし、パーセプトロンを利用した学習では線形分離不可能問題を解けない欠点があった。そこで 1980 年代に多層パーセプトロンが開発された[6]。多層パーセプトロンとはパーセプトロンに隠れ層と呼ばれる層を追加したニューラルネットワークである。層を追加することによって線形分離問題にも対応することが可能になった。このように層を増やすことによって様々な課題を解決できたことから、今日ではディープラーニングが注目されている。

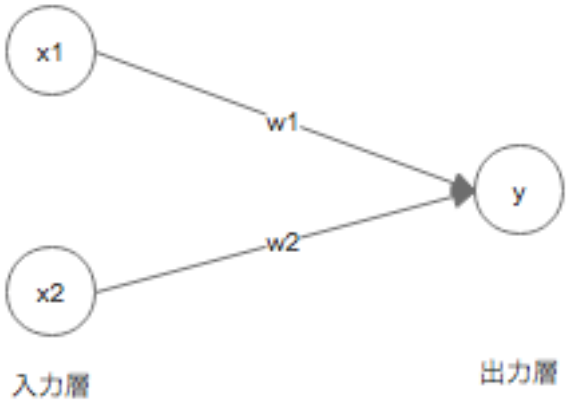

図 1 パーセプトロン

1.3 ディープラーニングに関する既知の結果

ディープラーニングは様々な分野で利用されているが、今日ではα碁[5]と呼ばれるディープラーニングと強化学習を使った囲碁 AI が注目されている。 α碁は Google の Deep Mind 社によって開発された囲碁 AI である。 α碁は囲碁における世界タイトルを獲得したプロの囲碁棋士と 3 度の対戦をしたが、3 戦 3 勝の成績を残している。また、そのほかのプロの囲碁棋士に対しても勝利を収めている。

2017 年 10 月にはα碁を改良したα碁ゼロが同会社によって開発された。このα碁ゼロはα囲碁に対して 100 戦 100 勝の成績を収めている。

1.4 本研究の目的

前節で述べたように将棋や囲碁についてはディープラーニングにより強い AI が作られている。一方、それ以外のゲームについてもディープラーニングを用いることにより強い AI を作れるかはまだまだ不明な点が多い。そこで、どのようなゲームであればディープラーニングを有効に用いられるか、あるゲームに対する有効性を確認し、ディープラーニングが有効となるかを検証する。本研究では、立体三目並べに対してディープラーニングを適用し、有用であるかを検証することを目的とする。

1.5 本報告書の構成

本報告書の構成は以下の通りである。まず第 2 章でディープラーニングの仕組み、DQN[1]、立体三目並べについて述べる。3 章で結果、考察を述べる。4 章で結論、今後の課題を述べる。

2　研究内容

2.1 ディープラーニングの仕組み[1]

ディープラーニングとは、ニューラルネットワークを利用した機械が物事を理解するための学習方法である。ニューラルネットワークとは人の神経を模したネットワーク構造であり、入力層、隠れ層、出力層を持ち、各層は複数のユニットがエッジで結ばれている。基本的なニューラルネットワークの例を図２に示す。図 2 の丸がユニット、各層をつなぐ矢印がエッジ、x が入力値、y が出力値である。また、各層は活性化関数を持ち、エッジは重み(w1~12)を持つ。活性化関数とはユニットがどのような値を出力するかを決定づける関数である。重みとはエッジの接続元が接続先に対する影響力のことである。これらの値、関数を利用し出力値を出している。

では、具体的にどのような計算が行われているかをみるために図 2 の z1 の値に注目する。z1 は次の式で表現できる。

$$z1= f(x1w1+x2w3+x3w5+b)\ldots. 1$$

１の式をみてわかるように入力値と重みの乗算結果を足したものを活性化関数fの引数とした値となっている。このような計算をそれぞれのユニットで行い出力値を出している。

またディープラーニングによる学習を行うためには、ニューラルネットワークの出した出力値が正しいか判断するための関数が必要である。それが損失関数である。損失関数の値は小さい程出力値が正しい。つまり、ディープラーニングは損失関数の値を小さくするために重みを変化していくものである。

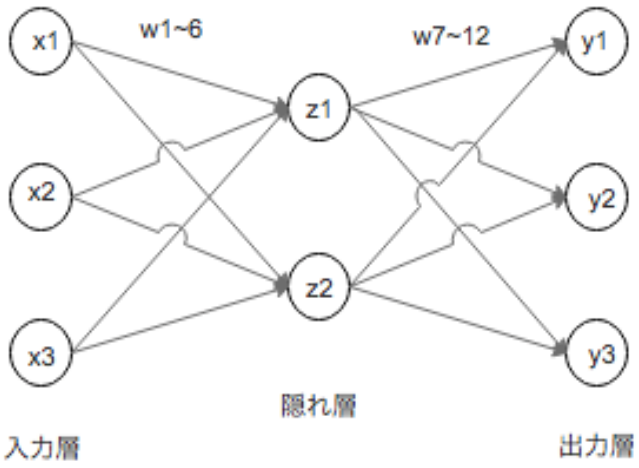

図 2　基本的なニューラルネットワーク

2.2 DQN

DQN とは Q 学習とディープラーニングを利用した強化学習のことである。[1]

Q 学習とは強化学習の 1 つであり、ある状況下でどのような行動をとるべきかを指標にした Q 値をより高い報酬を得られるよう適切な値を試行錯誤する学習方法である。例えば図 3 の X0 の状況において 2 つの行動パターンがあるが、Q 値の高い X2 の状況へ遷移するための行動をする。また、X2 の状況において 2 つの行動パターンがあるが、Q 値の高い X4 の状況へ遷移するための行動をする。そして X4 には報酬があるので今までの行動が正しかったと判断し、これまで行ってきた行動に対しての Q 値を更新する。Q 学習はこのような仕組みになっている。また、図３の場合は偶然報酬を得られる行動に対して高い Q 値が設定されているが、Q 値の初期値はランダムであり正しい Q 値を設定するためには、それぞれの状況下で様々な行動をとる必要がある。

本研究では、Q 学習とディープラーニングを合わせた DQN を用いて研究を行う。

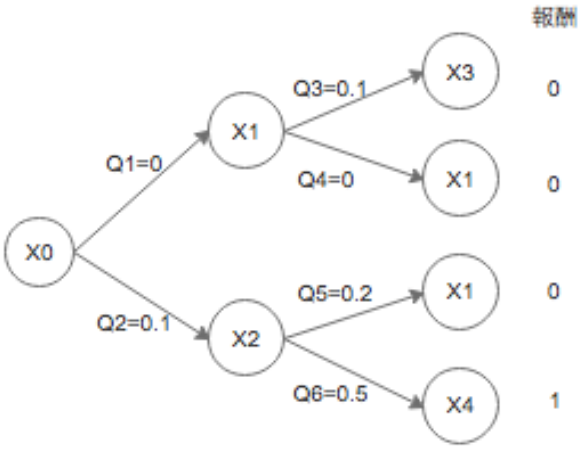

図 3Q 値と報酬

2.3 過学習

過学習とは、モデルの学習の際に利用されるごく 1 部のデータに対して、または学習に利用されるデータに過度に学習してしまう状態のことである。この状態に至ると未知のデータに対して正しい結果が出力されないなどの問題が発生する。

この過学習を防ぐ方法には L1 正則化と L2 正則化がある。L1 正則化とは特定のデータの重みを 0 にすることで不要なデータを削除する方法である。L2 正則化とはデータの大きさに応じて特定のデータの重みを 0 に近づけて滑らかなモデルを作る方法である。

2.4 立体 3 目並べ

本節では、本研究で対象となる立体 3 目並べのルールや前提、などを述べる。

立体3目並べとは 、3×3×3の27マスからなる立体を盤面とし、交互に〇と×を書き込み、〇あるいは×が3個直線上に並べると勝ちになるゲームである。図4に立体三目並べのゲーム盤を示す。また、今回はDQNで作成したゲームAIが必ず先手で〇をうつことを前提とする。

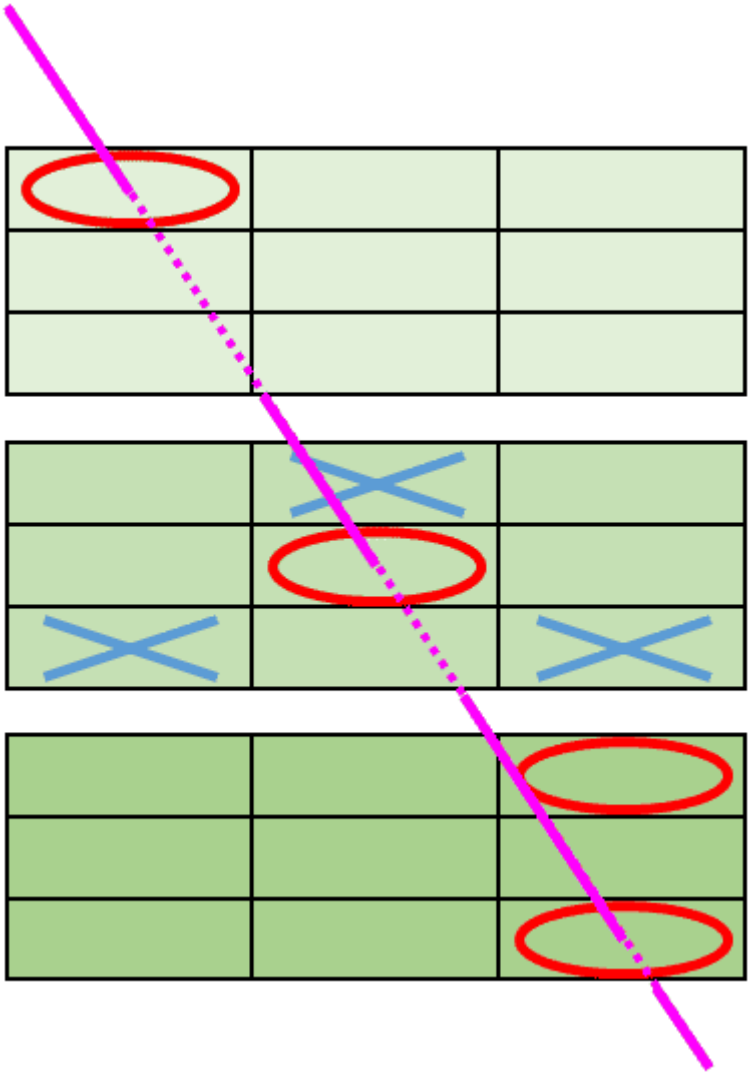

図 4 盤面

2.5 対戦させる AI

本研究では以下の 2 つの戦略に従う立体三目並べ AI を作成し、DQN を用いた AI と対戦し機械学習させる。

・戦略 A:盤面の状況に関係なく×をうつ。

・戦略 B:自分にリーチがかかっている場合は 3 つ並ぶマスに×をうつ。それ以外で相手にリーチがかかっている場合はそれを邪魔するマスに×をうつ。どちらでもない場合は戦略 A に従う。

3　結果・考察

ディープラーニングを用いたゲーム AI と戦略 A、B の対戦結果を図 5、図 6 に示す。縦軸は勝率、横軸は試行回数(学習回数)である。また、勝負が一回決まるごとに 1 試行と数える。

図 5 より戦略 A において、初めは負け続けているが試行回数が約 1 万回で勝率が 9 割超えており、その後

も安定して勝ち続けていることがわかる。一方戦略 B において、図 6 より初めは負け続けているが徐々に勝利するようになり、試行回数が 6 万 5 千回で勝率が 9 割超えているが、その後急激に勝利が下がっていることがわかる。

この結果から対戦相手強いほど、より試行回数が必要となることがわかる。また、勝率の減少については、過学習が原因と考える。

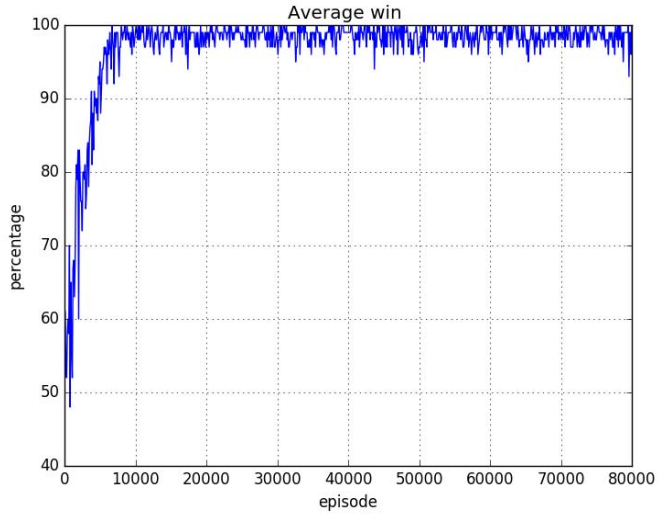

図 5 戦略 A との対戦結果

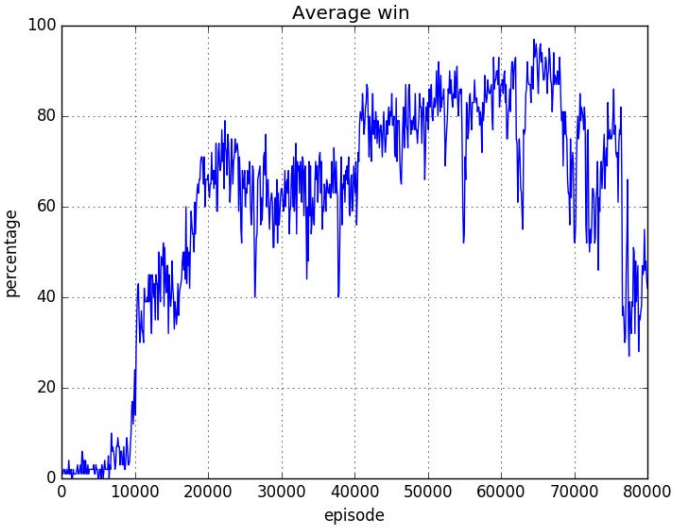

図 6 戦略 B との対戦結果

4　結論・今後の課題

本研究では立体三目並べにおいてディープラーニングによるゲーム AI を作成した。本研究で作成したゲーム AI は戦略 A、B に対して 9 割を超える勝率を実現することに成功した。しかし、結果からある程度強い戦略に対してはかなりの試行回数が必要と考えられる。また、戦略 B において一定の試行回数から勝率が急激に下がっており過学習の状態になっていると考えられるので、過学習が起きる前に学習を止める、過学習を起きないよう対策が必要である。

今後の課題としては、戦略 A、B に対しては高い勝率を実現できたが、戦略 A、B で学習させたモデルは他の戦略に対して高い勝率を実現できるとは限らないので、それを確かめる必要があるだろう。また、人間と対戦させて学習させること、過学習など学習させたモデルの崩壊が起きないよう対策を施すことが挙げられる。

謝辞

本研究を行うにあたって石水隆講師から些細な悩みから始まり、レジュメや卒論の添削など様々なご指導を受け賜わりました。ここに感謝の意を表します。

参考文献

[1] 藤田一弥、高原歩夢　：実装ディープラーニング、オーム社(2016)

[2] 斎藤康毅　：ゼロから作る Deep Learning-Python で学ぶディープラーニングの理論と実装、オライリージャパン（2016）

[3] 深層学習(ディープラーニング)を素人向けに解説-基礎となるニューラルネットワークについて、Stone Washer’s Jqurnal,2015 年 3 月 5 日、http://stonewashersjournal.com/2015/03/05/deeplearning1/

[4] Kazuto Seki:TECHNOTE 囲碁の最強人工知能 AlphaGo の仕組みとは？　2017 年 9 月 17 日,株式会社 div(2017) https://tech-camp.in/note/technology/32855/#AlphaGo

[5] 伊藤毅志，村松正和:ディープラーニングを用いたコンピュータ囲碁～ Alpha Go の技術と展望～，情報処理，Vol.57，No.4，pp.335-337，情報処理学会，(2016)．http://id.nii.ac.jp/1001/00158059/

[6] 日経 BigData　2015 年 4 月 21 日、ニューラルネットの歩んだ道、ディープラーニングの登場で全てが変わった、http://business.nikkeibp.co.jp/article/bigdata/20150419/280107/?P=1

ソースプログラム

本研究で作成したプログラムのソースファイルを以下に示す。

- Agent.py

```
class QNet(chainer.Chain):

    def __init__(self, n_in, n_units, n_out):
        super(QNet, self).__init__(
            l1=L.Linear(n_in, n_units),
            l2=L.Linear(n_units, n_units),
            l3=L.Linear(n_units, n_out),
        )

    def value(self, x):
        h = F.relu(self.l1(x))
        h = F.relu(self.l2(h))
        return self.l3(h)

    def __call__(self, s_data, a_data, y_data):
        self.loss = None

        s = chainer.Variable(self.xp.asarray(s_data))
        Q = self.value(s)

        Q_data = copy.deepcopy(Q.data)

        if type(Q_data).__module__ != np.__name__:
            Q_data = self.xp.asnumpy(Q_data)

        t_data = copy.deepcopy(Q_data)
        for i in range(len(y_data)):
            t_data[i, a_data[i]] = y_data[i]

        t = chainer.Variable(self.xp.asarray(t_data))
        self.loss = F.mean_squared_error(Q, t)

        print('Loss:', self.loss.data)
```

```
        return self.loss


# エージェントクラス
class MarubatsuAgent(Agent):

    # エージェントの初期化
    # 学習の内容を定義する
    def __init__(self, gpu):
        # 盤の情報
        self.n_rows = 3
        self.n_cols = self.n_rows
        self.n_z = self.n_rows

        # 学習の Input サイズ
        self.dim = self.n_rows * self.n_cols * self.n_z
        self.bdim = self.dim * 2

        self.gpu = gpu

        # 学習を開始させるステップ数
        self.learn_start = 5 * 10**3

        # 保持するデータ数
        self.capacity = 1 * 10**4

        # eps =  ランダムに○を決定する確率
        self.eps_start = 1.0
        self.eps_end = 0.001
        self.eps = self.eps_start

        # 学習時にさかのぼる Action 数
        self.n_frames = 3

        # 一度の学習で使用するデータサイズ
        self.batch_size = 32

        self.replay_mem = []
        self.last_state = None
```

```
        self.last_action = None
        self.reward = None
        self.state = np.zeros((1, self.n_frames, self.bdim)).astype(np.float32)

        self.step_counter = 0

        self.update_freq = 1 * 10**4

        self.r_win = 1.0
        self.r_draw = -0.5
        self.r_lose = -1.0

        self.frozen = False

        self.win_or_draw = 0
        self.stop_learning = 200

    # ゲーム情報の初期化
    def agent_init(self, task_spec_str):
        task_spec = TaskSpecVRLGLUE3.TaskSpecParser(task_spec_str)

        if not task_spec.valid:
            raise ValueError(
                'Task spec could not be parsed: {}'.format(task_spec_str))

        self.gamma = task_spec.getDiscountFactor()
        self.Q = QNet(self.bdim*self.n_frames,200, self.dim)

        if self.gpu >= 0:
            cuda.get_device(self.gpu).use()
            self.Q.to_gpu()
        self.xp = np if self.gpu < 0 else cuda.cupy

        self.targetQ = copy.deepcopy(self.Q)

        self.optimizer = optimizers.RMSpropGraves(lr=0.00025, alpha=0.95,
                                                  momentum=0.0)
        self.optimizer.setup(self.Q)
```

```
# environment.py env_start の次に呼び出される。
# 1 手目の○を決定し、返す
def agent_start(self, observation):
    # step を 1 増やす
    self.step_counter += 1

    # observation を[0-2]の 9 ユニットから[0-1]の 18 ユニットに変換する
    self.update_state(observation)

    self.update_targetQ()

    # ○の場所を決定する
    int_action = self.select_int_action()
    action = Action()
    action.intArray = [int_action]

    # eps を更新する。eps はランダムに○を打つ確率
    self.update_eps()

    # state = 盤の状態 と action = ○を打つ場所 を退避する
    self.last_state = copy.deepcopy(self.state)
    self.last_action = copy.deepcopy(int_action)

    return action

# エージェントの二手目以降、ゲームが終わるまで
def agent_step(self, reward, observation):
    # ステップを 1 増加
    self.step_counter += 1

    self.update_state(observation)
    self.update_targetQ()

    # ○の場所を決定
    int_action = self.select_int_action()
    action = Action()
    action.intArray = [int_action]
    self.reward = reward
```

```
        # eps を更新
        self.update_eps()

        # データを保存 (状態、アクション、報酬、結果)
        self.store_transition(terminal=False)

        if not self.frozen:
            # 学習実行
            if self.step_counter > self.learn_start:
                self.replay_experience()

        self.last_state = copy.deepcopy(self.state)
        self.last_action = copy.deepcopy(int_action)

        # ○の位置をエージェントへ渡す
        return action

    # ゲームが終了した時点で呼ばれる
    def agent_end(self, reward):
        # 環境から受け取った報酬
        self.reward = reward

        if not self.frozen:
            if self.reward >= self.r_draw:
                self.win_or_draw += 1
            else:
                self.win_or_draw = 0

            if self.win_or_draw == self.stop_learning:
                self.frozen = True
                f = open('result.txt', 'a')
                f.writelines('Agent frozen¥n')
                f.close()

        # データを保存 (状態、アクション、報酬、結果)
        self.store_transition(terminal=True)

        if not self.frozen:
            # 学習実行
```

```
        if self.step_counter > self.learn_start:
            self.replay_experience()

    def agent_cleanup(self):
        pass

    def agent_message(self, message):
        pass

    def update_state(self, observation=None):
        if observation is None:
            frame = np.zeros(1, 1, self.bdim).astype(np.float32)
        else:
            observation_binArray = []

            for int_observation in observation.intArray:
                bin_observation = '{0:02b}'.format(int_observation)
                observation_binArray.append(int(bin_observation[0]))
                observation_binArray.append(int(bin_observation[1]))

            frame = (np.asarray(observation_binArray).astype(np.float32)
                                                     .reshape(1, 1, -1))
        self.state = np.hstack((self.state[:, 1:], frame))

    def update_eps(self):
        if self.step_counter > self.learn_start:
            if len(self.replay_mem) < self.capacity:
                self.eps -= ((self.eps_start - self.eps_end) /
                             (self.capacity - self.learn_start + 1))

    def update_targetQ(self):
        if self.step_counter % self.update_freq == 0:
            self.targetQ = copy.deepcopy(self.Q)

    def select_int_action(self):
        free = []
        bits = self.state[0, -1]

        for i in range(0, len(bits), 2):
```

```
            if bits[i] == 0 and bits[i+1] == 0:
                free.append(int(i / 2))

        s = chainer.Variable(self.xp.asarray(self.state))
        Q = self.Q.value(s)

        # Follow the epsilon greedy strategy
        if np.random.rand() < self.eps:
            int_action = free[np.random.randint(len(free))]
        else:
            Qdata = Q.data[0]

            if type(Qdata).__module__ != np.__name__:
                Qdata = self.xp.asnumpy(Qdata)

            for i in np.argsort(-Qdata):
                if i in free:
                    int_action = i
                    break

        return int_action

    def store_transition(self, terminal=False):
        if len(self.replay_mem) < self.capacity:
            self.replay_mem.append(
                (self.last_state, self.last_action, self.reward,
                 self.state, terminal))
        else:
            self.replay_mem = (self.replay_mem[1:] +
                [(self.last_state, self.last_action, self.reward, self.state,
                  terminal)])

    def replay_experience(self):
        indices = np.random.randint(0, len(self.replay_mem), self.batch_size)
        samples = np.asarray(self.replay_mem)[indices]

        s, a, r, s2, t = [], [], [], [], []

        for sample in samples:
```

```
            s.append(sample[0])
            a.append(sample[1])
            r.append(sample[2])
            s2.append(sample[3])
            t.append(sample[4])

        s = np.asarray(s).astype(np.float32)
        a = np.asarray(a).astype(np.int32)
        r = np.asarray(r).astype(np.float32)
        s2 = np.asarray(s2).astype(np.float32)
        t = np.asarray(t).astype(np.float32)

        s2 = chainer.Variable(self.xp.asarray(s2))
        Q = self.targetQ.value(s2)
        Q_data = Q.data

        if type(Q_data).__module__ == np.__name__:
            max_Q_data = np.max(Q_data, axis=1)
        else:
            max_Q_data = np.max(self.xp.asnumpy(Q_data).astype(np.float32), axis=1)

        t = np.sign(r) + (1 - t)*self.gamma*max_Q_data

        self.optimizer.update(self.Q, s, a, t)
```

- Environment,py

```
import numpy as np
np.random.seed(0)
from rlglue.environment.Environment import Environment
from rlglue.environment import EnvironmentLoader as EnvironmentLoader
from rlglue.types import Observation
from rlglue.types import Reward_observation_terminal


class MarubatsuEnvironment(Environment):

    # 盤の状態 [空白, ○, ×]
```

```
flg_free = 0
flg_agent = 1
flg_env = 2

# 報酬
r_win = 1.0
r_draw = -0.5
r_lose = -1.0

# 敵プレイヤーが正常に打つ確率
opp = 0.75

def __init__(self):
    self.n_rows = 3
    self.n_cols = self.n_rows
    self.n_z = self.n_rows

    # 3 つ並んだら勝敗が決まるライン
    # 3×3 の盤なので、[0,1,2],[3,4,5].....
    lines = []
    #平面
    for i in range(self.n_rows*self.n_cols):
        horizontal = []
        for j in range(self.n_cols):
            horizontal.append(i*self.n_cols + j)
        lines.append(horizontal)

    for i in range(self.n_cols):
        vertical1 = []
        for j in range(self.n_rows):
            vertical1.append(i + j*self.n_cols)
        lines.append(vertical1)

    for i in range(9,12):
        vertical2=[]
        for j in range(self.n_cols):
            vertical2.append(i + j*self.n_cols)
        lines.append(vertical2)
```

```
for i in range(18,21):
    vertical3=[]
    for j in range(self.n_cols):
        vertical3.append(i + j*self.n_cols)
    lines.append(vertical3)

if self.n_rows == self.n_cols:
    n = self.n_rows
    tl_br, tr_bl = [], []
    for i in range(n):
        tl_br.append(i*n + i)
        tr_bl.append(i*n + (n - 1) - i)
    lines.append(tl_br)
    lines.append(tr_bl)

lines.append([9,13,17])
lines.append([11,13,15])
lines.append([18,22,26])
lines.append([20,22,24])

lines.append([0,10,20])
lines.append([2,10,18])

lines.append([0,12,24])
lines.append([6,12,18])

lines.append([6,16,26])
lines.append([8,16,24])

lines.append([8,14,20])
lines.append([2,14,26])



#立体
for i in range(9):
    vertical4=[]
    for j in range(self.n_rows):
        vertical4.append(i+9*j)
```

```
    lines.append(vertical4)

lines.append([0,13,26])
lines.append([8,13,18])
lines.append([6,13,20])
lines.append([2,13,24])



self.lines = lines

self.history = []
```