report

氏 名： Matts966

2018 年 1 月 30 日

1　作成したプログラムとその目的について

今回作成したプログラムは、三目並べを解くことを目的として、DoubleDQN（Deep Q Network）と呼ばれる学習手法を、ChainerRL というライブラリを用いて実装し、Minimax 法と簡単なコツを利用したアルゴリズムと戦わせる、というものである。添付した Numpy 形式のログにもあるように、30 万回強の学習回数でも、学習は収束せず、簡単なアルゴリズムにもまけることがある、ということがわかった。ただ、そもそも置けない場所を選択肢に含め、それを選んだ場合に Miss として処理するという計測方法は、条件が厳しかったかもしれない。機械学習も、目的や細かい実装によっては、思い通りの結果を出すことができない、ということの例にはなるかもしれない。最後には、実際に自分の手で遊んで、学習の度合を確かめられるようになっている。以下はそのプログラムである。

1.1　プログラム

```
 1 import chainer
 2 import chainer.functions as F
 3 import chainer.links as L
 4 import chainerrl
 5 import numpy as np
 6
 7 #三目並べ用のボードをクラスとして実装
 8 class Board():
 9     def reset(self):
10         self.board = np.array([0] * 9, dtype=np.float32)
11         self.winner = None
12         self.missed = False
13         self.done = False
14
```

```
15      def move(self, action, turn):
16          if self.board[action] == 0:
17              self.board[action] = turn
18              self.check_winner()
19          else:
20              self.winner = turn*-1
21              self.missed = True
22              self.done = True
23
24      def check_winner(self):
25          win_conditions = ((0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8)
                ,(0,4,8),(2,4,6))
26          for cond in win_conditions:
27              if self.board[cond[0]] == self.board[cond[1]] == self.board[
                    cond[2]]:
28                  if self.board[cond[0]] != 0:
29                      self.winner = self.board[cond[0]]
30                      self.done = True
31                      return
32          if np.count_nonzero(self.board) == 9:
33              self.winner = 0
34              self.done = True
35
36      def get_empty_pos(self):
37          empties = np.where(self.board==0)[0]
38          if len(empties) > 0:
39              return np.random.choice(empties)
40          else:
41              return 0
42
43      def show(self):
44          row = " {} | {} | {} "
45          hr = "\n-----------\n"
46          tempboard = []
47          count = 0
48          for i in self.board:
49              count += 1
50              if i == 1:
51                  tempboard.append("○")
52              elif i == -1:
```

```
53                tempboard.append("×")
54            else:
55                tempboard.append(str(count))
56        print((row + hr + row + hr + row).format(*tempboard))
57        print("\n")
58
59 #用のランダム関数オブジェクト explorer
60 class RandomActor:
61     def __init__(self, board):
62         self.board = board
63         self.random_count = 0
64     def random_action_func(self):
65         self.random_count += 1
66         return self.board.get_empty_pos()
67
68 #関数 Q
69 class QFunction(chainer.Chain):
70     def __init__(self, obs_size, n_actions, n_hidden_channels = 81):
71         super().__init__(
72             l0 = L.Linear(obs_size, n_hidden_channels),
73             l1 = L.Linear(n_hidden_channels, n_hidden_channels),
74             l2 = L.Linear(n_hidden_channels, n_hidden_channels),
75             l3 = L.Linear(n_hidden_channels, n_actions))
76     def __call__(self, x, test=False):
77         #を扱うので-1とした leaky_relu
78         h = F.leaky_relu(self.l0(x))
79         h = F.leaky_relu(self.l1(h))
80         h = F.leaky_relu(self.l2(h))
81         return chainerrl.action_value.DiscreteActionValue(self.l3(h))
82
83 # ボードの準備
84 b = Board()
85
86 # 用のランダム関数オブジェクトの準備 explorer
87 ra = RandomActor(b)
88
89 # 環境と行動の次元数
90 obs_size = 9
91 n_actions = 9
92
```

```
93  # Q-とオプティマイザーのセットアップ function
94  q_func = QFunction(obs_size, n_actions)
95  optimizer = chainer.optimizers.Adam(eps=1e-2)
96  optimizer.setup(q_func)
97
98  # 報酬の割引率
99  gamma = 0.95
100
101 # Epsilon-を使ってたまに冒険。ステップでとなる greedy50000end_epsilon
102 explorer = chainerrl.explorers.LinearDecayEpsilonGreedy(
103     start_epsilon = 1.0, end_epsilon = 0.3, decay_steps = 50000
104     , random_action_func = ra.random_action_func)
105
106 # Experience というで用いる学習手法で使うバッファ ReplayDQN
107 replay_buffer = chainerrl.replay_buffer.ReplayBuffer(capacity = 10 ** 6)
108
109 # の生成（等を共有するつ）Agentreplay_buffer2
110 agent_p1 = chainerrl.agents.DoubleDQN(
111     q_func, optimizer, replay_buffer, gamma, explorer,
112     replay_start_size = 500, update_interval = 1,
113     target_update_interval = 100)
114
115 agent_p2 = chainerrl.agents.DoubleDQN(
116     q_func, optimizer, replay_buffer, gamma, explorer,
117     replay_start_size = 500, update_interval = 1,
118     target_update_interval = 100)
119
120 # のロード。最初から学習させたければコメントアウト、保存されたモデルを再利用してス
        タートしたい場合はアンコメントアウト。Model
121 # agent_p1.load('dqn_result')
122 # agent_p2.load('dqn_result')
123
124 #学習ゲーム回数
125 #ほんとはたくさん稼ぎたいが、実際のところメモリと所要時間的に辛い
126 print('学習回数を入力 > ', end='')
127 n_episodes = int(input())
128
129 #カウンタの宣言
130 miss = 0
131 win = 0
```

```
132 draw = 0
133 #エピソードの繰り返し実行
134 for i in range(1, n_episodes + 1):
135     b.reset()
136     reward = 0
137     agents = [agent_p1, agent_p2]
138     turn = np.random.choice([0, 1])
139     last_state = None
140     while not b.done:
141         #配置マス取得
142         action = agents[turn].act_and_train(b.board.copy(), reward)
143         #配置を実行
144         b.move(action, 1)
145         #配置の結果、終了時には報酬とカウンタに値をセットして学習
146         if b.done == True:
147             if b.winner == 1:
148                 reward = 1
149                 win += 1
150             elif b.winner == 0:
151                 draw += 1
152             else:
153                 reward = -1
154             if b.missed is True:
155                 miss += 1
156             #エピソードを終了して学習
157             agents[turn].stop_episode_and_train(b.board.copy(), reward,
                    True)
158             #相手もエピソードを終了して学習。相手のミスは勝利として学習しないように
159             if agents[1 if turn == 0 else 0].last_state is not None and b.
                    missed is False:
160                 #前のターンでとっておいたを実行後の状態として渡す last_stateaction
161                 agents[1 if turn == 0 else 0].stop_episode_and_train(
                        last_state, reward * -1, True)
162         else:
163             #学習用にターン最後の状態を退避
164             last_state = b.board.copy()
165             #継続のときは盤面の値を反転
166             b.board = b.board * -1
167             #ターンを切り替え
168             turn = 1 if turn == 0 else 0
```

```
169
170     #コンソールに進捗表示
171     if i % 100 == 0:
172         print("episode:", i, " / rnd:", ra.random_count, " / miss:", miss
173         , " / win:", win, " / draw:", draw, " / statistics:", agent_p1.
                get_statistics()
174         , " / epsilon:", agent_p1.explorer.epsilon)
175         #カウンタの初期化
176         miss = 0
177         win = 0
178         draw = 0
179         ra.random_count = 0
180     if i % 10000 == 0:
181         # エピソードごとにモデルを保存 10000
182         agent_p1.save("dqn_result")
183
184 print("Training finished.")
185
186
187 #人間のプレーヤー
188 class HumanPlayer:
189     def act(self, board):
190         valid = False
191         while not valid:
192             try:
193                 act = input("Please enter 1-9: ")
194                 act = int(act)
195                 if act >= 1 and act <= 9 and board[act-1] == 0:
196                     valid = True
197                     return act-1
198                 else:
199                     print("Invalid move")
200             except Exception as e:
201                 print(act + " is invalid")
202
203
204
205 # 法 MiniMax
206 class MiniMax:
207     def minimax(self, copiedBoard, mini_first):
```

```
208         win_conditions = ((0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8)
                ,(0,4,8),(2,4,6))
209         for cond in win_conditions:
210             if copiedBoard[cond[0]] == copiedBoard[cond[1]] == copiedBoard[
                    cond[2]]:
211                 if copiedBoard[cond[0]] == 1:
212                     return -10
213                 elif copiedBoard[cond[0]] == -1:
214                     return 10
215
216         empties = np.where(copiedBoard == 0)[0]
217
218         if len(empties) == 0:
219             return 0
220
221         children = []
222         minimaxedChildren = []
223
224         # 相手の番
225         if np.sum(copiedBoard) < 0 or (not mini_first and np.sum(
                copiedBoard) == 0):
226             for e in empties:
227                 cb = copiedBoard.copy()
228                 cb[e] = 1
229                 children.append(cb)
230             for child in children:
231                 minimaxedChildren.append(self.minimax(child, mini_first))
232             return min(minimaxedChildren)
233
234         # の番 Minimax
235         else:
236             for e in empties:
237                 cb = copiedBoard.copy()
238                 cb[e] = -1
239                 children.append(cb)
240             for child in children:
241                 minimaxedChildren.append(self.minimax(child, mini_first))
242             return max(minimaxedChildren)
243
244
```

```
245     def act(self, board, mini_first):
246         empties = np.where(board==0)[0]
247         if len(empties) == 9:
248             return np.random.choice(empties)
249         if len(empties) == 0:
250             print(board)
251             print('Bug')
252             return 0
253         if len(empties) == 1:
254             return empties[0]
255         children = []
256         for e in empties:
257             b = board.copy()
258             b[e] = -1
259             m = self.minimax(b, mini_first)
260             children.append(m)
261         return empties[children.index(max(children))]
262 
263 # 負けなし戦略ではないがそこそこ強い戦略
264 class intelligentMethod:
265     # 今の状態から次にどこに置くかを決定する
266     def act(self, board):
267 
268         empties = np.where(board==0)[0]
269         win_conditions = ((0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8)
                ,(0,4,8),(2,4,6))
270 
271         # もし勝てるのであれば勝つ
272         for e in empties:
273             b = board.copy()
274             b[e] = -1
275             for cond in win_conditions:
276                 if b[cond[0]] == b[cond[1]] == b[cond[2]]:
277                     if b[cond[0]] == -1:
278                         return e
279 
280         # もし負けるのであれば妨害を入れる
281         for e in empties:
282             b = board.copy()
283             b[e] = 1
```

```
284             for cond in win_conditions:
285                 if b[cond[0]] == b[cond[1]] == b[cond[2]]:
286                     if b[cond[0]] == 1:
287                         return e
288 
289         # もし中央が空いていればそこに置く.
290         if board[4] == 0:
291             return 4
292 
293         # もし隅が空いていればランダムに隅の場所に置く.
294         corner = (0, 2, 6, 8)
295         for c in corner:
296             if board[c] == 0:
297                 return c
298 
299         # それ以外の場合は適当に置く.
300         for c in range(9):
301             if board[c] == 0:
302                 return c
303 
304 
305 # との勝負で検証 MiniMaxDQN
306 mini_max = MiniMax()
307 mini = 0
308 dqn = 0
309 draw = 0
310 for i in range(10):
311     b.reset()
312     dqn_first = np.random.choice([True, False])
313     while not b.done:
314         #DQN
315         if dqn_first or np.count_nonzero(b.board) > 0:
316             b.show()
317             action = agent_p1.act(b.board.copy())
318             b.move(action, 1)
319             if b.done == True:
320                 b.show()
321                 if b.winner == 1:
322                     print("DQN Win")
323                     dqn += 1
```

```
324                 elif b.winner == 0:
325                     print("Draw")
326                     draw += 1
327                 else:
328                     print("DQN Missed")
329                     print('action : ' + str(action))
330                     mini += 1
331                 agent_p1.stop_episode()
332                 continue
333         #Minimax
334         b.show()
335         action = mini_max.act(b.board.copy(), not dqn_first)
336         b.move(action, -1)
337         if b.done == True:
338             b.show()
339             if b.winner == -1:
340                 print("MiniMax Win")
341                 mini += 1
342             elif b.winner == 0:
343                 print("Draw")
344                 draw += 1
345             agent_p1.stop_episode()
346 print ('DQN WIN : ' + str(dqn) + ' MiniMax WIN : ' + str(mini) + ' Draw :
        ' + str(draw))
347
348 # ちょっとした原則に基づいたプレイと比較
349 IntM = intelligentMethod()
350 Int = 0
351 dqn = 0
352 draw = 0
353 for i in range(10):
354     b.reset()
355     dqn_first = np.random.choice([True, False])
356     while not b.done:
357         #DQN
358         if dqn_first or np.count_nonzero(b.board) > 0:
359             b.show()
360             action = agent_p1.act(b.board.copy())
361             b.move(action, 1)
362             if b.done == True:
```

```
363                 b.show()
364                 if b.winner == 1:
365                     print("DQN Win")
366                     dqn += 1
367                 elif b.winner == 0:
368                     print("Draw")
369                     draw += 1
370                 else:
371                     print("DQN Missed")
372                     print('action : ' + str(action))
373                     Int += 1
374                 agent_p1.stop_episode()
375                 continue
376         #Minimax
377         b.show()
378         action = IntM.act(b.board.copy())
379         b.move(action, -1)
380         if b.done == True:
381             b.show()
382             if b.winner == -1:
383                 print("Int Win")
384                 Int += 1
385             elif b.winner == 0:
386                 print("Draw")
387                 draw += 1
388             agent_p1.stop_episode()
389 print ('DQN WIN : ' + str(dqn) + ' Int WIN : ' + str(Int) + ' Draw : ' +
        str(draw))
390
391 #人間との勝負で検証
392 human_player = HumanPlayer()
393 for i in range(10):
394     b.reset()
395     dqn_first = np.random.choice([True, False])
396     while not b.done:
397         #DQN
398         if dqn_first or np.count_nonzero(b.board) > 0:
399             b.show()
400             action = agent_p1.act(b.board.copy())
401             b.move(action, 1)
```

```
402             if b.done == True:
403                 b.show()
404                 if b.winner == 1:
405                     print("DQN Win")
406                 elif b.winner == 0:
407                     print("Draw")
408                 else:
409                     print("DQN Missed")
410                     print('action:' + str(action))
411                 agent_p1.stop_episode()
412                 continue
413         #人間
414         b.show()
415         action = human_player.act(b.board.copy())
416         b.move(action, -1)
417         if b.done == True:
418             b.show()
419             if b.winner == -1:
420                 print("HUMAN Win")
421             elif b.winner == 0:
422                 print("Draw")
423             agent_p1.stop_episode()
424
425 #三十五万回程度では収束せず。。
426 print("Test finished.")
```