```
# -*- coding: utf-8 -*-


import threading
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random
from copy import copy
import time
import csv


from Maze_1 import Maze_1
from Maze_2 import Maze_2
from Maze_2A import Maze_2A
from Maze_2AB import Maze_2AB


tf.compat.v1.disable_eager_execution()
tf.compat.v1.reset_default_graph()
"""
実行方法：291行の実験パラメータを設定してから実行する
"""




#N_S = env_info['state_shape']# observationの次元数4
N_S = 4
print("observationの次元数は{}".format(N_S))
#N_A = env_info['n_actions']# 行動の次元数4
N_A = 4
print("行動の次元数は{}".format(N_A))


class ACNet(object):
    def __init__(self, scope, globalAC=None):

        if scope == GLOBAL_NET_SCOPE_1 or scope == GLOBAL_NET_SCOPE_2:   # get global network
            with tf.compat.v1.variable_scope(scope):# 'Global_Net'
                [redacted]
                self.s = tf.compat.v1.placeholder(tf.float32, [None, N_S], 'S') #  observationのplaceholderを生成する
                [redacted]
                self.a_params, self.c_params = self._build_net(scope)[-2:]# ActorとCriticのパラメータをゲットする
                [redacted]
        else:   # local network, calculate losses
            with tf.compat.v1.variable_scope(scope):# eg.Worker_1

                #print("-----------------Local "+str(scope)+" is created!-----------------------------")
                self.s = tf.compat.v1.placeholder(tf.float32, [None, N_S], 'S')
                self.a_his = tf.compat.v1.placeholder(tf.int32, [None, ], 'A')
                self.v_target = tf.compat.v1.placeholder(tf.float32, [None, 1], 'Vtarget')

                self.a_prob, self.v, self.a_params, self.c_params = self._build_net(scope)

                td = tf.compat.v1.subtract(self.v_target, self.v, name='TD_error')# TD誤差、目標値と予測値の差
                with tf.compat.v1.name_scope('c_loss'):# Criticの損失関数の定義
                    self.c_loss = tf.compat.v1.reduce_mean(tf.square(td)) # Criticの損失関数はTD誤差の二乗として定義する
                    #print("Criticの損失関数c_loss is {}".format(self.c_loss.shape))

                with tf.compat.v1.name_scope('a_loss'):# Actorの損失関数の定義
```

インスタンス
　クラスを　実際に使用するために実体化したもの

コンストラクタ
　クラス内にある__init__の事.

インデント.
　字下げの事.

の理解を行う

tfとnp内のメソッドが
わからない

変数を定義する

型の一種

差を出力

Critic

```
        # Q * log π (a|s)
        log_prob = tf.compat.v1.reduce_sum(tf.compat.v1.log(self.a_prob + 1e-5) *
                                           tf.one_hot(self.a_his, N_A, dtype=tf.float32),# eg. a_his = [0,1,3] , N_A = 4 -> [[1,0,0,0],[0,1,0,0],[0,0,0,1]]
                                           axis=1, keepdims=True)
        [redacted] 1)
        exp_v = log_prob * tf.compat.v1.stop_gradient(td)
        [redacted]
        entropy = -tf.compat.v1.reduce_sum(self.a_prob * tf.compat.v1.log(self.a_prob + 1e-5),
                                           axis=1, keepdims=True)  #  エントロピーの計算
        [redacted]
        self.exp_v = ENTROPY_BETA * entropy + exp_v
        [redacted]
        self.a_loss = tf.compat.v1.reduce_mean(-self.exp_v)
        [redacted]

    with tf.compat.v1.name_scope('local_grad'):#勾配を計算
        self.a_grads = tf.compat.v1.gradients(self.a_loss, self.a_params) # Actorの勾配を計算する

        self.c_grads = tf.compat.v1.gradients(self.c_loss, self.c_params) # Criticの勾配を計算する


    with tf.compat.v1.name_scope('sync'):
        with tf.compat.v1.name_scope('pull'): # パラメータサーバーから重みなどをコピーする
            self.pull_a_params_op = [l_p.assign(g_p) for l_p, g_p in zip(self.a_params, globalAC.a_params)]# パラメータサーバーのActorから重みなどをコピーする
            self.pull_c_params_op = [l_p.assign(g_p) for l_p, g_p in zip(self.c_params, globalAC.c_params)]# パラメータサーバーのCriticから重みなどをコピーする
        with tf.compat.v1.name_scope('push'): # 計算した勾配を利用しパラメータサーバーの重みを更新する計算を定義する
            self.update_a_op = OPT_A.apply_gradients(zip(self.a_grads, globalAC.a_params))
            self.update_c_op = OPT_C.apply_gradients(zip(self.c_grads, globalAC.c_params))

def _build_net(self, scope):
    # ネットワークを構成する
    w_init = tf.random_normal_initializer(0., .1)# 重みを初期化する
    with tf.compat.v1.variable_scope('actor'):# Actorのネットワークを構成する
        l_a = tf.compat.v1.layers.dense(self.s, 200, tf.nn.relu6, kernel_initializer=w_init, name='la')
        a_prob = tf.compat.v1.layers.dense(l_a, N_A, tf.nn.softmax, kernel_initializer=w_init, name='ap') # 行動の選択確率のベクトルを出力する
        #print("行動選択確率ベクトルa_prob is {}".format(a_prob.shape))#(None, 4)
    with tf.compat.v1.variable_scope('critic'):# Criticのネットワークを構成する
        l_c = tf.compat.v1.layers.dense(self.s, 100, tf.nn.relu6, kernel_initializer=w_init, name='lc')
        v = tf.compat.v1.layers.dense(l_c, 1, kernel_initializer=w_init, name='v')  # 状態価値を出力する
        #print("状態価値関数c is {}".format(v.shape))#(None, 1)
    a_params = tf.compat.v1.get_collection(tf.compat.v1.GraphKeys.TRAINABLE_VARIABLES, scope=scope + '/actor')# Actorの重みパラメータをゲットする
    c_params = tf.compat.v1.get_collection(tf.compat.v1.GraphKeys.TRAINABLE_VARIABLES, scope=scope + '/critic')# Criticの重みパラメータをゲットする
    return a_prob, v, a_params, c_params

def update_global(self, feed_dict):# パラメータサーバーの重みを更新するメソッド
    SESS.run([self.update_a_op, self.update_c_op], feed_dict)

def pull_global(self):# パラメータサーバーの重みをコピーするメソッド
    SESS.run([self.pull_a_params_op, self.pull_c_params_op])

def choose_action(self, s):
    prob_weights = SESS.run(self.a_prob, feed_dict={self.s: s[np.newaxis, :]})
    #print("prob_weights is {}".format(prob_weights.shape))#(1, 4)
    action = np.random.choice(range(prob_weights.shape[1]),
                              p=prob_weights.ravel())# [0,1,2,3] から  確率p [0.1,0.2,0.5,0.2] によって一個を選ぶ. p は選択確率が異なる場合
    return action


class Worker(object):
    def __init__(self, name, algorithm,learning_data):

        self.name = name

        self.AC = algorithm

        self.learning_data = learning_data
```

```
    self.data_num  = len(self.learning_data)

    #print(self.data_num)


def update_A3C(self):
    #global GLOBAL_RUNNING_R, GLOBAL_EP,T


    #print("¥n"+"=========================="+str(self.name)+"のパラメータサーバーの重みを更新中==============================")
    #share_percent = 1
    #self.data_shuffer(share_percent)

    #if total_step % UPDATE_GLOBAL_ITER == 0 or done:   # 10ステップ一回パラメータサーバーの重みを更新する
    for id in range(self.data_num): # 全てのデータを使って更新

        episode_done = self.learning_data[id][4]# エージェント1のデータのepisode done
        last_obs = self.learning_data[id][2]# エージェント1のデータのlast_obs_r1
        episode_reward = self.learning_data[id][3]# エージェント1のデータのbuffer_r
        episodes_buffer_s = self.learning_data[id][0]# エージェント1のデータのbuffer_s1
        episodes_buffer_a = self.learning_data[id][1]# エージェント1のデータのbuffer_a1




        if episode_done:
            v_s_ = 0   # 終端状態の状態価値は0とする
        else:
            s_ = last_obs# エージェント1のデータのs1_
            #print("s1_ is {}".format(s1_))
            v_s_ = SESS.run(self.AC.v, {self.AC.s: s_[np.newaxis, :]})[0, 0]#?
        #print("2222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222")
        buffer_v_target = [] # robot1

        for r in episode_reward[::-1]:    # reverse buffer r
            v_s_ = r + GAMMA * v_s_ # v(s) = r + GAMMA *  v(s+1)によってtarget_vを計算する
            buffer_v_target.append(v_s_)

        buffer_v_target.reverse()

        #print("buffer_v_target is {}".format(buffer_v_target))



        episodes_buffer_s, episodes_buffer_a, buffer_v_target = np.vstack(episodes_buffer_s), np.array(episodes_buffer_a), np.vstack(buffer_v_target)

        #print("buffer_s1 is {}".format(buffer_s1))
        #print("buffer_a1 is {}".format(buffer_a1))
        #print("buffer_v_target is {}".format(buffer_v_target))
        feed_dict = {
            self.AC.s: episodes_buffer_s,
            self.AC.a_his: episodes_buffer_a,
            self.AC.v_target: buffer_v_target,
            }

        self.AC.update_global(feed_dict)
        self.AC.pull_global()
    #print("=========================="+str(self.name)+"のパラメータサーバーの重み更新完了==============================="+"¥n")

def run(env,algorithm_1,algorithm_2,epsilon = 0.1):
    """

    Parameters
    ----------
    env:
        環境のインスタンス.
```

```
    algorithm_1 : TYPE
        agent 1 の  アルゴリズム.
    algorithm_2 : TYPE
        agent 1 の  アルゴリズム.

    Returns
    -------
    joint_actions.

    """
    s_1 = env.sense_observations('1') # 最初の状態  agent1の状態をゲットする
    s_2 = env.sense_observations('2') # 最初の状態  agent2の状態をゲットする
    joint_actions = []

    actions = [0,1,2,3]
    a = random.uniform(0,1)
    #print(a)
    if a < epsilon :
        a_1_sign = random.choice(actions) # agent1 はランダムに行動を選択する
        a_2_sign = random.choice(actions) # agent2 はランダムに行動を選択する
        #print("ランダム")
    else:
        a_1_sign = algorithm_1.choose_action(s_1) # agent1 はA3Cにより行動を選択する
        #a_1 = actions[a_1_sign]
        a_2_sign = algorithm_2.choose_action(s_2) # agent2 はA3Cにより行動を選択する
        #a_2 = actions[a_2_sign]

    joint_actions.append(a_1_sign)
    joint_actions.append(a_2_sign)

    return joint_actions

def random_joint_actions(actions):
    joint_actions = []
    action_1 = random.choice(actions)
    action_2 = random.choice(actions)
    joint_actions.append(action_1)
    joint_actions.append(action_2)
    return joint_actions

def data_shuffer(share_percent,data_r1,data_r2):# データ共有率
    #print(share_percent)
    #print("Sharing model is {}".format(share_model))
    data_num = len(data_r1)
    sample_num = int(share_percent * data_num)
    #print("sample_num is {}".format(sample_num))


    sample_data_r1 = []
    sample_data_r2 = []

    sample_data_r1 = random.sample(data_r1,sample_num)# エージェント1のデータからサンプリングした共有データ
    sample_data_r2 = random.sample(data_r2,sample_num)# エージェント2のデータからサンプリングした共有データ



    #print("sample_data_r1 is {}".format(sample_data_r1))
    #print("sample_data_r2 is {}".format(sample_data_r2))
    #elif share_model == 3 :


    learning_data_r1 = data_r1 + sample_data_r2 # エージェント2の共有データはエージェント1の学習用
    #print(str(len(self.learning_data_r1)))
    learning_data_r2 = data_r2 + sample_data_r1 # エージェント1の共有データはエージェント2の学習用

    return learning_data_r1,learning_data_r2
```

```
if __name__ == "__main__":

    start_time = time.time() #実行開始時間


    GAMMA = 0.9
    ENTROPY_BETA = 0.001
    LR_A = 0.001    # Actorの学習率
    LR_C = 0.001    # Criticの学習率




    #--------------------実験パラメータ-----------------------

    trials = 10 # 試行回数
    episode_num = 100 #  エピソードの数
    max_steps_num = 40#毎エピソードのステップの数の上限     迷路環境1：40  迷路環境2, 2(A),2(AB):100
    is_share_data = True#データ共有：True 共有しない：False
    N_WORKERS = 4# マルチスレッドのスレッド数

    share_percent = 0.25#経験データ共有率X_A


    env = Maze_1()#迷路環境1
    #env = Maze_2()#迷路環境2
    #env = Maze_2A()#迷路環境2A
    #env = Maze_2AB()#迷路環境2AB

    #--------------------実験パラメータ-----------------------
```

```
    actions = env.actions # 行動集合['up','down','left','right']

    episodes_buffer_s1,episodes_buffer_a1 = [], []# 各episodeのrobot1 の状態、行動を記録する
    episodes_buffer_s2,episodes_buffer_a2 = [], []# 各episodeのrobot2 の状態、行動を記録する
    episodes_buffer_r = []# 共通の報酬を記録する
    episodes_dones = []
    last_obs_r1 = []# 各episodeのrobot1 の最終状態を記録する
    last_obs_r2 = []# 各episodeのrobot2 の最終状態を記録する



    trials_episodes_steps = []
    trials_episodes_rewards = []
    GOAL_EPISODES = []
    one_goal_episodes = []
    trap_episodes = []

    trial_one_goal_episodes_in_episodes = []#全試行の毎エピソード目の中１つだけゴールエピソードを記録する.eg.123エピソード目の時、ゴールエピソードの数を記録する
    trial_two_goal_episodes_in_episodes = []#全試行の毎エピソード目の中2つだけゴールエピソードを記録する.eg.123エピソード目の時、ゴールエピソードの数を記録する
    trial_trap_episodes_in_episodes = []#全試行の毎エピソード目の中罠にはまったゴールエピソードを記録する.



    str_time = time.strftime("%Y%m%d_%H_%M_%S", time.localtime(start_time))
    path = 'learning_data/log'+str(str_time)+'.csv'

    with open(path,'w') as f:
        csv_write = csv.writer(f)
        csv_head = ['trial','Episode','reward','steps','two','one_more','trap']
        csv_write.writerow(csv_head)
```

```
for trial in range(trials):


    tf.compat.v1.reset_default_graph()
    SESS = tf.compat.v1.Session() # tensorflowのsessionを作る
    #SESS = tf.keras.backend.get_session()



    OPT_A = tf.compat.v1.train.RMSPropOptimizer(LR_A, name='RMSPropA') # Actorのoptimizerを生成する (RMSProp)
    OPT_C = tf.compat.v1.train.RMSPropOptimizer(LR_C, name='RMSPropC') # Criticのoptimizerを生成する (RMSProp)

    GOAL_EPISODE_R = [] # 目標達成のエピソードの報酬和を保存する
    GOAL_EP = 0 # 目標達成のエピソードの数
    one_goal_ep = 0 #一つゴールに到達したエピソードの数
    trap_ep = 0#罠にはまったエピソード数

    one_goal_episodes_in_episodes = [] #毎エピソード目の中１つゴールエピソードを記録する
    two_goal_episodes_in_episodes = [] #毎エピソード目の中2つゴールエピソードを記録する
    trap_episodes_in_episodes = [] #毎エピソード目の中罠にはまったエピソードを記録する


    robot1_name = 'robot_1'
    robot2_name = 'robot_2'

    GLOBAL_NET_SCOPE_1 = 'Global_Net_1'# パラメータサーバーのNNの名前
    GLOBAL_NET_SCOPE_2 = 'Global_Net_2'# パラメータサーバーのNNの名前
    GLOBAL_AC_1 = ACNet(GLOBAL_NET_SCOPE_1)  # robot1のパラメータサーバーを生成する
    GLOBAL_AC_2 = ACNet(GLOBAL_NET_SCOPE_2)  # robot2のパラメータサーバーを生成する

    AC_1 = ACNet(robot1_name, GLOBAL_AC_1)# robot1のACネットワークを生成する
    AC_2 = ACNet(robot2_name, GLOBAL_AC_2)# robot2のACネットワークを生成する

    # Coordinatorにより、Sessionの中にある多数のThreadを管理する
    # tf.train.Coordinator()によりインスタンスを生成する
    inif = tf.compat.v1.global_variables_initializer()
    COORD = tf.train.Coordinator()
    SESS.run(inif) # 全ての変数を初期化する


    print("試行 "+str(trial+1)+" 開始!")

    episodes_steps = [] #  全てのepisodeの歩数を記録する
    episodes_rewards = [] #  全てのepisodeの累積報酬を記録する

    episodes_buffer_data_r1 = []
    episodes_buffer_data_r2 = []



    for episode in range(episode_num):


        #print("Episode "+str(episode+1)+" starts!")

        s_1 = env.sense_observations('1') # 最初の状態  robot1の状態をゲットする
        s_2 = env.sense_observations('2') # 最初の状態  robot2の状態をゲットする


        buffer_s1, buffer_a1 = [], []# robot1 の状態、行動を記録する
        buffer_s2, buffer_a2 = [], []# robot2 の状態、行動を記録する
        buffer_r = []# 共通の報酬を記録する

        data_buffer = [] #全てのデータを記録する
```

TensorFlowを実行可能にする

最適化ツール

？ACNetで作成したパラメータサーバーをもう一度ACNetに代入している

何となくわかる

インスタンス化？

AC_1にはNNが入っている？

```
done_sign = False

total_steps = 0 # episodeの歩数を記録する
ep_r = 0 # episodeのrewardを累積する(共通の報酬和)

buffer_data_r1 = [] # robot1 のデータを記録 s,a,s_,r,done
buffer_data_r2 = [] # robot1 のデータを記録



result_log = []

for step in range(max_steps_num):


    data = []# excel書き込み内容
    data.append(str(trial+1))#trial
    data.append(str(episode+1))#episode
    #data.append(str(step+1))#step

    joint_actions = []

    if episode/episode_num > 0.75 :#半分エピソードの後
        joint_actions_sign = run(env, AC_1, AC_2,0.01) # epsilon = 0.01
    elif episode/episode_num > 0.5 :
        joint_actions_sign = run(env, AC_1, AC_2,0.05) # epsilon = 0.05
    else:
        joint_actions_sign = run(env, AC_1, AC_2) # joint actions のインデックスを獲得する  eg.[0,1]


    joint_actions.append(actions[joint_actions_sign[0]])
    joint_actions.append(actions[joint_actions_sign[1]])

    #joint_actions = random_joint_actions(actions)

    #data.append(str(joint_actions[0]))#robot1 action
    #data.append(str(joint_actions[1]))#robot2 action

    #print("Episode "+str(episode+1)+" : step "+str(step+1)+" robot 1 action: "+str(joint_actions[0])+" robot 2 action: "+str(joint_actions[1]))
    s_, r, episode_done,goals_num = env.step(joint_actions)# 次の状態、報酬、目標に到達標識をゲットする
    状態 報酬 判定1 判定2

    #  通路の報酬を変える。0～1/4ステップ数：0.04, 1/4~1/2ステップ数：-0.01,1/2ステップ数以後:-0.04

    if (step+1) >= 0 and (step+1) < (max_steps_num*(1/4)) :
        #print("今は0～1/4ステップ数：0.04")
        if r == -1.04 :
            r = -0.96
        elif r == -0.14 :
            r = -0.06
        elif r == -0.08 :
            r = 0.08
        elif r == 9.96 :
            r = 10.04
        elif r == 19.96 :
            r = 20.04
        else :
            r = r
    elif (step+1) >= (max_steps_num*(1/4)) and (step+1) < (max_steps_num*(1/2)) :
        #print("今は1/4~1/2ステップ数：-0.01")
        if r == -1.04 :
            r = -1.01
        elif r == -0.14 :
            r = -0.11
        elif r == -0.08 :
            r = -0.02
        elif r == 9.96 :
```

行動選択

これと行う
理由不明

```
            r = 9.99
        elif r == 19.96 :
            r = 19.99
        else :
            r = r
    else:
        #print("今は1/2ステップ数以後:-0.04")
        r = r



    #print("次状態:{}".format(s_))
    #print("報酬:{}".format(r))
    #print("エピソード終了:{}".format(episode_done))
    #print("到達した宝物数:{}".format(goals_num))

    s1_ = s_[0] # robot1 の次状態
    s2_ = s_[1] # robot2 の次状態

    ep_r += r # 報酬を累積する
    total_steps += 1 # 歩数を累積する

    buffer_s1.append(s_1)
    buffer_a1.append(joint_actions_sign[0])
    #buffer_a1.append(joint_actions[0])

    buffer_s2.append(s_2)
    buffer_a2.append(joint_actions_sign[1])
    #buffer_a2.append(joint_actions[1])

    buffer_r.append(r)

    s_1 = s1_
    s_2 = s2_

    env.render()


    if episode_done:


        env.reset() # 環境をリセットする

        done_sign = True


        break
if goals_num == 2:# 目標達成する
    GOAL_EPISODE_R.append(ep_r)
    #print('¥n')
    #print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%全部の目標に到達した%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
    #print("Episode:", str(episode+1),"| Ep_r: %i" % GOAL_EPISODE_R[-1],)
    GOAL_EP += 1
elif goals_num == 1:
    one_goal_ep += 1
elif goals_num == -3:
    trap_ep += 1
    #print("trap !")
elif goals_num == -2:
    trap_ep += 1
    #print("trap !")
    one_goal_ep += 1
elif goals_num == -1:
    trap_ep += 1
    #print("trap !")
    GOAL_EPISODE_R.append(ep_r)
```

```
    #print('¥n')
    #print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%全部の目標に到達した%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
    #print("Episode:", str(episode+1),"| Ep_r: %i" % GOAL_EPISODE_R[-1],)
    GOAL_EP += 1

one_goal_episodes_in_episodes.append(one_goal_ep) #毎エピソード目の中１つゴールエピソードを記録する
two_goal_episodes_in_episodes.append(GOAL_EP) #毎エピソード目の中2つゴールエピソードを記録する
trap_episodes_in_episodes.append(trap_ep)#毎エピソード目の中罠にはまったエピソードを記録する


if done_sign == False: # 目標達成していない　最大歩数使い切った
    #print("#####################################全部の目標に到達できず#########################################")
    env.reset() # 環境をリセットする

#print("Episode "+str(episode+1)+" is over!")
result_log.append(str(trial+1))#trial
result_log.append(str(episode+1))#episode
result_log.append(str(ep_r))#total_rewark
result_log.append(str(total_steps))#total_step

result_log.append(str(GOAL_EP))#2goals
result_log.append(str(GOAL_EP + one_goal_ep))#one more
result_log.append(str(trap_ep))

with open(path,'a') as f:
        csv_write = csv.writer(f)
        csv_write.writerow(result_log)

episodes_steps.append(total_steps)
episodes_rewards.append(ep_r)


buffer_data_r1.append(buffer_s1)
buffer_data_r1.append(buffer_a1)
buffer_data_r1.append(s1_)
buffer_data_r1.append(buffer_r)
buffer_data_r1.append(episode_done)

buffer_data_r1.append(ep_r)

#print("buffer_data_r1:{}".format(buffer_data_r1))

episodes_buffer_data_r1.append(buffer_data_r1)

buffer_data_r2.append(buffer_s2)
buffer_data_r2.append(buffer_a2)
buffer_data_r2.append(s2_)
buffer_data_r2.append(buffer_r)
buffer_data_r2.append(episode_done)

buffer_data_r2.append(ep_r)

#print("buffer_data_r2:{}".format(buffer_data_r2))

episodes_buffer_data_r2.append(buffer_data_r2)


#print("episodes_buffer_data_r1:{}".format(episodes_buffer_data_r1))
#print("episodes_buffer_data_r2:{}".format(episodes_buffer_data_r2))


learning_episodes_buffer_data_r1 = copy(episodes_buffer_data_r1)
learning_episodes_buffer_data_r2 = copy(episodes_buffer_data_r2)
#print(len(learning_episodes_buffer_data_r1))
#print(learning_episodes_buffer_data_r1)
data_buffer.append(learning_episodes_buffer_data_r1)
```

CSVに記録

robot1のデータをまとめて記録（一部ステップ・単位で）

s1やa1のデータ

[[ ],[ ],0,[ ],0,0]

さらに圧縮

r1と同様にr2も行った

```
data_buffer.append(learning_episodes_buffer_data_r2)
#print("data buffer のデータの変化{}".format(len(data_buffer[0])))
#print(data_buffer)


if is_share_data:#データを共有する
    #print("the number of data before share:{}".format(len(learning_episodes_buffer_data_r1)))

    learning_data_r1,learning_data_r2 = data_shuffer(share_percent,learning_episodes_buffer_data_r1,learning_episodes_buffer_data_r2)

    #print("the number of data after share:{}".format(len(learning_data_r1)))
else:
    #print("learning data is not be share!")
    learning_data_r1,learning_data_r2 = learning_episodes_buffer_data_r1,learning_episodes_buffer_data_r2
    #print("the number of data :{}".format(len(learning_episodes_buffer_data_r1)))

#学習部分

#ランダム方式, マルチスレッド
if is_share_data:

    workers_agen1 = []
    # 各workerを生成する
    for i in range(N_WORKERS): # N_WORKERS の数
        i_name = 'Worker_agent1_%i' % (i+1)   # workerの名前：eg.Worker_1
        workers_agen1.append(Worker(i_name, AC_1,learning_data_r1))

    workers_agen2 = []
    # 各workerを生成する
    for i in range(N_WORKERS): # N_WORKERS の数
        i_name = 'Worker_agent2_%i' % (i+1)   # workerの名前：eg.Worker_1
        workers_agen2.append(Worker(i_name, AC_2,learning_data_r2))


    total_workers = []
    total_workers = workers_agen1 + workers_agen2
    #print("the number of total workers is {}".format(len(total_workers)))

    worker_threads = []
    # 各workerのthreadを生成する
    #print("=========================="+str(N_WORKERS)+"個のスレッドが更新開始==============================")
    for worker in total_workers:# 学習をmulti threadで実行する
        job = lambda: worker.update_A3C()
        t = threading.Thread(target=job) # 一つのthreadを生成し、任務を配布する
        t.start() # threadが起動する
        worker_threads.append(t)
    COORD.join(worker_threads) #  生成したthreadをCoordinatorに入り、threadが終了するまで待機する
else:

    worker_agent1_name = 'Worker_agent1'
    worker_agent2_name = 'Worker_agent2'
    worker_agent1 = Worker(worker_agent1_name, AC_1,learning_data_r1)
    worker_agent2 = Worker(worker_agent2_name, AC_2,learning_data_r2)
    worker_agent1.update_A3C()
    worker_agent2.update_A3C()


#print("=======================================更新完了=========================================")

SESS.close()


#print("目標達成エピソードの報酬:{}".format(GOAL_EPISODE_R))
print("試行　{}　中罠にはまったエピソードの数:{}".format(trial+1,trap_ep))
print("試行　{}　中一つゴールに達成エピソードの数:{}".format(trial+1,one_goal_ep))
print("試行　{}　中二つゴールに達成エピソードの数:{}".format(trial+1,GOAL_EP))
```

```
    trials_episodes_steps.append(episodes_steps)
    trials_episodes_rewards.append(episodes_rewards)
    GOAL_EPISODES.append(GOAL_EP)
    one_goal_episodes.append(one_goal_ep)
    trap_episodes.append(trap_ep)

    trial_one_goal_episodes_in_episodes.append(one_goal_episodes_in_episodes)
    trial_two_goal_episodes_in_episodes.append(two_goal_episodes_in_episodes)
    trial_trap_episodes_in_episodes.append(trap_episodes_in_episodes)

#print("各試行のエピソードのステップ数:{}".format(trials_episodes_steps))
#print("各試行のエピソードの報酬:{}".format(trials_episodes_rewards))

#print("全てのdata buffer:{}".format(data_buffer))
#print("{}".format(len(data_buffer)))

"""
# パラメータを保存する

now_time = datetime.datetime.now().strftime('%Y-%m-%d-%H-%M-%S')
save_path = "data/multiagent"
save_file_name = save_path+"/singleA3C"+str(now_time)+""
saver = tf.compat.v1.train.Saver()
saver.save(SESS,save_path=save_file_name)
print("パラメータを保存した")
"""
#env.destroy()
print("全試行の平均罠にはまったエピソード数:{}".format(np.mean(trap_episodes)))
print("全試行の平均一つゴールに達成エピソード数:{}".format(np.mean(one_goal_episodes)))
print("全試行の平均二つゴール達成エピソード数:{}".format(np.mean(GOAL_EPISODES)))
print("全試行の平均一つ以上のゴール達成エピソード数:{}".format(np.mean(GOAL_EPISODES) + np.mean(one_goal_episodes)))
avg_episodes_steps = np.mean(trials_episodes_steps,axis=0)
avg_episodes_rewards = np.mean(trials_episodes_rewards,axis=0)
avg_trial_one_goal_episodes_in_episodes = np.mean(trial_one_goal_episodes_in_episodes,axis=0)
avg_trial_two_goal_episodes_in_episodes = np.mean(trial_two_goal_episodes_in_episodes,axis=0)
avg_trial_trap_episodes_in_episodes = np.mean(trial_trap_episodes_in_episodes,axis=0)

#print(len(avg_trial_two_goal_episodes_in_episodes))




# episodeとその歩数と累積報酬を図に出力する

fig = plt.figure(figsize=(16,4))
axL = fig.add_subplot(1, 3, 1)
axC = fig.add_subplot(1, 3, 2)
axR = fig.add_subplot(1, 3, 3)

axL.plot(np.arange(1,len(avg_episodes_steps)+1), avg_episodes_steps)
axL.set_title('Steps')
axL.set_xlabel('episode')
axL.set_ylabel('Total moving steps')

axC.plot(np.arange(1,len(avg_trial_two_goal_episodes_in_episodes)+1), avg_trial_two_goal_episodes_in_episodes)
axC.set_title('Goal Episode')
axC.set_xlabel('episode')
axC.set_ylabel('Goal Episode')


axR.plot(np.arange(1,len(avg_episodes_rewards)+1), avg_episodes_rewards)
axR.set_title('Reward')
axR.set_xlabel('episode')
axR.set_ylabel('Total moving reward')
fig.show()
```

```
env.destroy()
elapsed_time = time.time() - start_time

print("trials: {},Maximum steps: {}, episodes: {}, workers: {}, elapsed_time: {}".format(trials,max_steps_num,episode_num,N_WORKERS,elapsed_time) + "[sec]")
```