# 引き分けに対する報酬の追加

　本章では、Q-learningを用いた三目並べAIのより良いモデルの構築を「引き分け」状態に対する報酬を追加することで検証する。

　プログラムは下記の通りであり、引き分け状態の報酬は「reward=={0,0.5,-0.5}」の○○に報酬値を入力することで決定する。報酬無し、負の報酬、正の報酬(reward = 〇〇)それぞれを与えることでどのようにランダムAIとの勝率が変化するかを検証する。具体的には、ランダムAIと10,000回試合を行い、そのモデルを用いて再度ランダムAIと10,000回試合を行う(この際ε=0)。この際の勝率を、並行処理を用いて10000回実行し勝率の分布を得る。実行環境はCPUがAMD Ryzen 9 5950X 16-Core Processor、GPUがNVIDIA GeForce RTX 3090である。

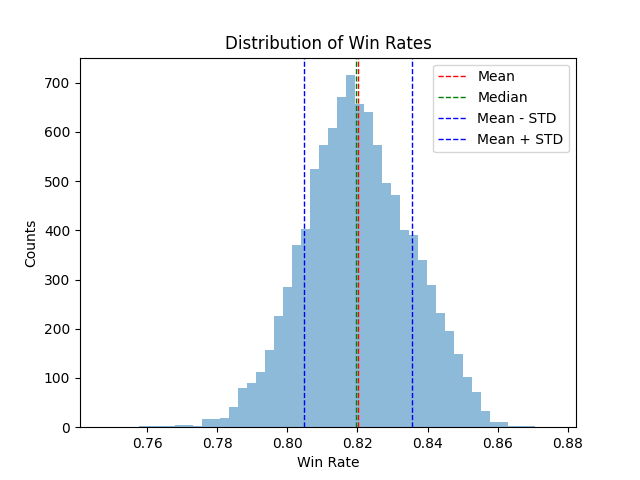
変更プログラム1:randomAI\_vs\_QLAI関数

|  |
| --- |
| # ランダムAIとQ学習AIのゲームを実行する関数  def randomAI\_vs\_QLAI(first\_inputter, q\_table, epsilon=0.5):  inputter1 = 'Random AI'  inputter2 = 'Q-Learning AI'  ql\_input\_list = []  play\_area\_list = []  play\_area = list(range(1, 10))  inputter\_count = first\_inputter  end\_flg = 0  ql\_flg = 0  reward = 0  game\_count = 0 # ゲームの回数をカウントするための変数を追加します  while True:  # Q学習退避用  play\_area\_tmp = play\_area.copy()  play\_area\_list.append(play\_area\_tmp)  # Q学習実行フラグ  ql\_flg = 0  # Q学習AIの手番  if (inputter\_count % 2) == 0:  # Q学習AIの入力  play\_area, ql\_ai\_input = get\_ai\_input(play\_area, first\_inputter, mode=1, q\_table=q\_table, epsilon=epsilon)  winner, end\_flg = judge(play\_area, inputter2)  # Q学習退避用  ql\_input\_list.append(ql\_ai\_input)  # Q学習AIが勝利した場合  if winner == inputter2:  reward = 1  ql\_flg = 1  play\_area\_before = play\_area\_list[-1]  ql\_ai\_input\_before = ql\_input\_list[-1]  # ランダムAIの手番  elif (inputter\_count % 2) == 1:  play\_area, random\_ai\_input = get\_ai\_input(play\_area, first\_inputter+1, mode=0)  winner, end\_flg = judge(play\_area, inputter1)  # ランダムAIが勝利した場合  if winner == inputter1:  reward = -1  #ランダムAIが先手の場合の初手以外は学習  #elif winner == 'Nobody':  # reward = 〇〇 # 引き分けの場合の報酬を追加  if inputter\_count != 1:  ql\_flg = 1  # Q学習実行  if ql\_flg == 1:  ql\_ai\_input\_before = ql\_input\_list[-1]  q\_table = q\_learning(play\_area\_before, ql\_ai\_input\_before, reward, play\_area, q\_table, end\_flg)  if end\_flg:  break  inputter\_count += 1  game\_count += 1 # ゲームの回数をインクリメントします  #print('{} win!!!'.format(winner))  return winner, q\_table |

変更プログラム2:学習統計プログラム

|  |
| --- |
| import multiprocessing as mp  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  eta = 0.1 # 学習率  gamma = 0.9 # 時間割引率  initial\_epsilon = 0.5 # ε  episode = 10000 # エピソード数  # 学習セット数  learning\_sets = 10000  def worker(set\_index, return\_dict):  winner\_list = []  # Q学習テーブルを初期化  q\_table = make\_q\_table()  # 学習プロセス  for i in range(episode):  epsilon = initial\_epsilon \* (episode-i) / episode  \_, q\_table = randomAI\_vs\_QLAI(1, q\_table, epsilon)  # テストプロセス（モデルはこれ以上更新しない）  test\_episodes = 10000 # テストエピソード数を設定  for i in range(test\_episodes):  winner, \_ = randomAI\_vs\_QLAI(1, q\_table, 0) # ε=0で学習せずに実行  winner\_list.append(winner)  ql\_win\_rate = winner\_list.count('Q-Learning AI') / len(winner\_list)  print(f"Learning Set {set\_index+1}, Q-Learning AI Win Rate: {ql\_win\_rate}")  return\_dict[set\_index] = ql\_win\_rate  if \_\_name\_\_ == "\_\_main\_\_":  manager = mp.Manager()  return\_dict = manager.dict()  processes = []  for set\_index in range(learning\_sets):  p = mp.Process(target=worker, args=(set\_index, return\_dict))  p.start()  processes.append(p)  for p in processes:  p.join()  win\_rates = list(return\_dict.values())  # 勝率の平均と中央値、分散と標準偏差を計算  mean\_win\_rate = np.mean(win\_rates)  median\_win\_rate = np.median(win\_rates)  variance = np.var(win\_rates)  std\_deviation = np.std(win\_rates)  print(f"Mean Win Rate: {mean\_win\_rate}")  print(f"Median Win Rate: {median\_win\_rate}")  print(f"Variance of Win Rate: {variance}")  print(f"Standard Deviation of Win Rate: {std\_deviation}")  # 勝率の分布をヒストグラムとして表示  plt.hist(win\_rates, bins=50, alpha=0.5)  plt.title('Distribution of Win Rates')  plt.xlabel('Win Rate')  plt.ylabel('Counts')  # 平均値と中央値、±1標準偏差を縦線でプロット  plt.axvline(mean\_win\_rate, color='r', linestyle='dashed', linewidth=1)  plt.axvline(median\_win\_rate, color='g', linestyle='dashed', linewidth=1)  plt.axvline(mean\_win\_rate - std\_deviation, color='b', linestyle='dashed', linewidth=1)  plt.axvline(mean\_win\_rate + std\_deviation, color='b', linestyle='dashed', linewidth=1)  # レジェンドを作成  plt.legend({'Mean':mean\_win\_rate, 'Median':median\_win\_rate,  'Mean - STD':mean\_win\_rate - std\_deviation, 'Mean + STD':mean\_win\_rate + std\_deviation})  plt.show() |

まず、引き分けの際の報酬を追加しない(reward = 0)従来のプログラムで検証する。

Mean Win Rate: 0.8200714899999999

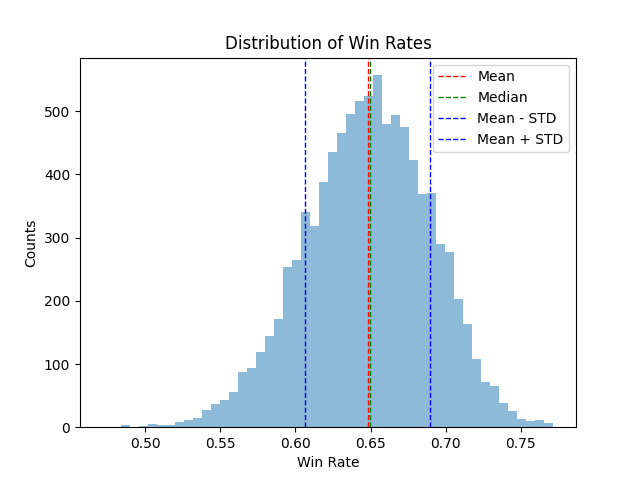
Median Win Rate: 0.81955

Variance of Win Rate: 0.00023723890617990002

Standard Deviation of Win Rate: 0.01540256167589989

となった。

次に引き分けの際正の報酬を取る(reward = 0.5)場合を検証する。



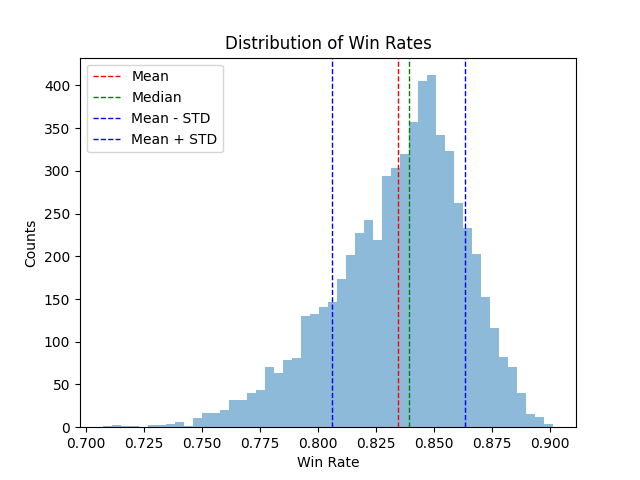
Mean Win Rate: 0.6479948093904804

Median Win Rate: 0.6493

Variance of Win Rate: 0.001732383102499744

Standard Deviation of Win Rate: 0.041621906521683315

最後に引き分けの際負の報酬(reward = -0.5)を取る場合を検証する。

Mean Win Rate: 0.8343601144789906

Median Win Rate: 0.8385

Variance of Win Rate: 0.0008255917225314503

Standard Deviation of Win Rate: 0.02873311195348409

Elapsed time: 2937.4432940483093 seconds

Mean Win Rate: 0.834803006407097

Median Win Rate: 0.8394

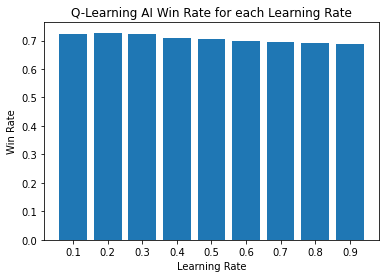
Variance of Win Rate: 0.0008168061368461886

Standard Deviation of Win Rate: 0.028579820448109687

Elapsed time: 2402.6597578525543 seconds

# 最適な学習率の探求

|  |
| --- |
| # 学習率のリスト  etas = np.arange(0.1, 1, 0.1)  win\_rate\_for\_each\_eta = []  # それぞれの学習率で学習を行います  for eta in etas:  q\_table = make\_q\_table()  gamma = 0.9  initial\_epsilon = 0.5  episode = 100000  winner\_list = []  for i in range(episode):  epsilon = initial\_epsilon \* (1 - i / episode)  winner, q\_table = randomAI\_vs\_QLAI(1, q\_table, epsilon)  winner\_list.append(winner)  # 勝率を計算し、リストに追加  win\_rate = winner\_list.count('Q-Learning AI') / len(winner\_list)  win\_rate\_for\_each\_eta.append(win\_rate)  print(f"eta={eta}, Q-Learning AI Win Rate={win\_rate}")  # 学習率ごとの勝率を棒グラフで表示します  plt.bar(etas, win\_rate\_for\_each\_eta, width=0.08)  plt.title('Q-Learning AI Win Rate for each Learning Rate')  plt.xlabel('Learning Rate')  plt.ylabel('Win Rate')  plt.xticks(etas)  plt.show() |



# 学習回数による勝率の分布の検証