# Лабораторная работа №2.

# Кросс-энтропийный метод (CEM)

**1. Инициализация среды.**

```
import sys, os

import gym
import numpy as np

env = gym.make("Taxi-v3")
env.reset()
env.render()

n_states = env.observation_space.n
n_actions = env.action_space.n

print("n_states=%i, n_actions=%i" % (n_states, n_actions))
```

После запуска появляется окно с визуализацией модели и значениями ее параметров.

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | :|| : |
|Y| : |B: |
+---------+

n_states=500, n_actions=6
```

**2. Создание стохастической политики**

Политика должна быть вероятностным распределением.

policy[s,a] = P(выполнить действие a | в состоянии s)

Поскольку мы по-прежнему используем целочисленные представления состояний и действий, для представления политики можно использовать двумерный массив. Инициализируйте политику равномерно, то есть вероятности всех действий должны быть равными:

```
def initialize_policy(n_states, n_actions):

    < Ваш код: создать массив для хранения вероятности действий >

    return policy


policy = initialize_policy(n_states, n_actions)

assert type(policy) in (np.ndarray, np.matrix)
assert np.allclose(policy, 1./n_actions)
assert np.allclose(np.sum(policy, axis=1), 1)
```

**3. Игра с моделью**

```
def generate_session(env, policy, t_max=10**4):
    """
      Играть до конца или t_max тиков.
     :param policy: массив вида [n_states,n_actions] с вероятностями действий
     :returns: список состояний, список действий и сумма наград
    """
    states, actions = [], []
    total_reward = 0.

    s = env.reset()

    for t in range(t_max):
        # Hint: вы можете использовать np.random.choice для выборки
        #
https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.h
tml
        a = <ВАШ КОД: пример действия из policy>

        new_s, r, done, info = env.step(a)

        # Запись информацию, которая получена из среды.
        states.append(s)
        actions.append(a)
        total_reward += r

        s = new_s
        if done:
            break

    return states, actions, total_reward
```

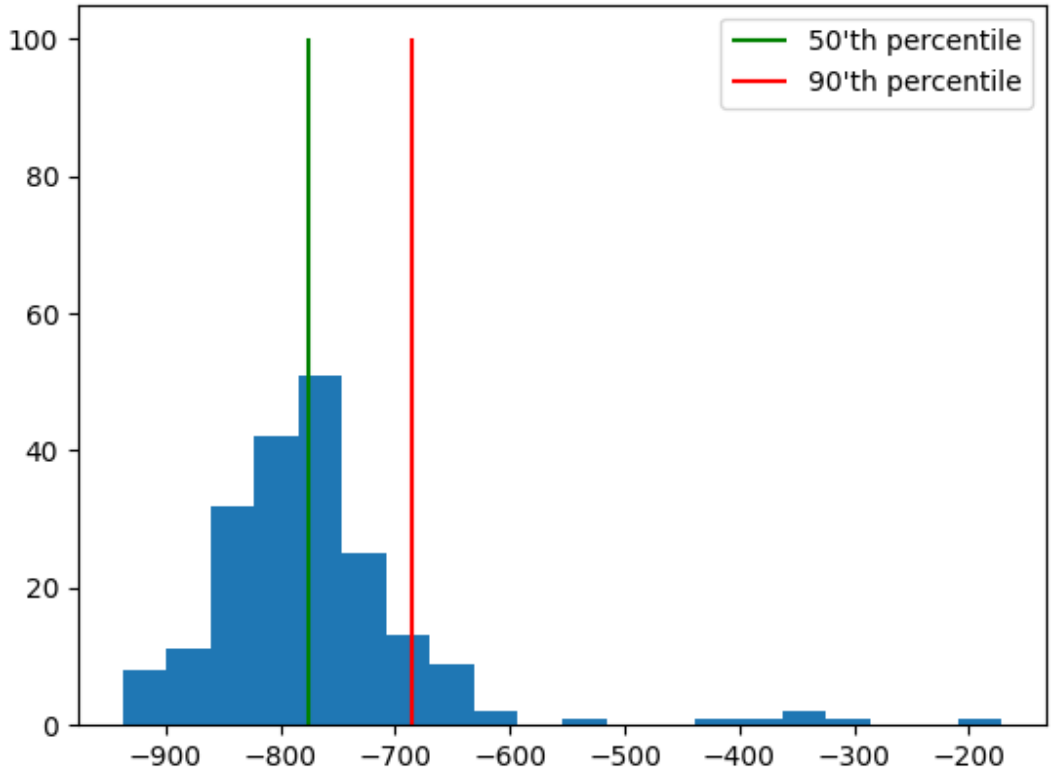
```
s, a, r = generate_session(env, policy)
assert type(s) == type(a) == list
assert len(s) == len(a)
assert type(r) in [float, np.float64]
```

```
# Визуализация начального распределения вознаграждения
import matplotlib.pyplot as plt

sample_rewards = [generate_session(env, policy, t_max=1000)[-1] for _ in
range(200)]

plt.hist(sample_rewards, bins=20)
plt.vlines([np.percentile(sample_rewards, 50)], [0], [100], label="50'th
percentile", color='green')
plt.vlines([np.percentile(sample_rewards, 90)], [0], [100], label="90'th
percentile", color='red')
plt.legend()
plt.show()
```

Вывод:




**3. Шаги кросс-энтропийного метода**

```
def select_elites(states_batch, actions_batch, rewards_batch, percentile):
    """
    Выберите состояния и действия из игры, которые имеют награды >=
процентиль
     :paramstates_batch: список списков состояний,states_batch[session_i][t]
     :param action_batch: список списков действий, action_batch[session_i][t]
     :param rewards_batch: список наград, rewards_batch[session_i]

    :returns: elite_states,elite_actions, одномерные списки состояний и
соответствующих действий лучших сессий

    """

    reward_threshold = <ВАШ КОД: вычисление минимального вознаграждения для
лучших сессий. Hint: используйте np.percentile()>

    elite_states = <ВАШ КОД>
    elite_actions = <ВАШ КОД>

    return elite_states, elite_actions
```

```
states_batch = [
    [1, 2, 3],      # игра1
    [4, 2, 0, 2],   # игра2
    [3, 1],         # игра3
]
```

```
actions_batch = [
    [0, 2, 4],     # игра1
    [3, 2, 0, 1],  # игра2
    [3, 3],        # игра3
]
rewards_batch = [
    3,  # игра1
    4,  # игра2
    5,  # игра3
]

test_result_0 = select_elites(states_batch, actions_batch, rewards_batch,
percentile=0)
test_result_30 = select_elites(states_batch, actions_batch, rewards_batch,
percentile=30)
test_result_90 = select_elites(states_batch, actions_batch, rewards_batch,
percentile=90)
test_result_100 = select_elites(states_batch, actions_batch, rewards_batch,
percentile=100)

assert np.all(test_result_0[0] == [1, 2, 3, 4, 2, 0, 2, 3, 1])  \
    and np.all(test_result_0[1] == [0, 2, 4, 3, 2, 0, 1, 3, 3]), \
    "Для процентиля 0 вы должны вернуть все состояния и действия в
хронологическом порядке."
assert np.all(test_result_30[0] == [4, 2, 0, 2, 3, 1]) and \
    np.all(test_result_30[1] == [3, 2, 0, 1, 3, 3]), \
    "Для процентиля 30 вы должны выбрать состояния/действия только из двух
первых"
assert np.all(test_result_90[0] == [3, 1]) and \
    np.all(test_result_90[1] == [3, 3]), \
    "Для процентиля 90 вы должны выбирать состояния/действия только из одной
игры."
assert np.all(test_result_100[0] == [3, 1]) and\
    np.all(test_result_100[1] == [3, 3]), \
    "Убедитесь, что вы используете >=, а не >. Также дважды проверьте, как вы
вычисляете процентиль."
```

```
def get_new_policy(elite_states, elite_actions):
    """
    Учитывая список лучших состояний/действий от select_elites,
     возвращает новую политику, где вероятность каждого действия
пропорциональна

        policy[s_i,a_i] ~ #[появления s_i и a_i в элитарных
состояниях/действиях]

    Не забудьте нормализовать политику, чтобы получить действительные
вероятности и обработать случай 0/0.
     Для состояний, в которых вы никогда не находились, используйте
равномерное распределение (1/n_actions для всех состояний).

     :param Elite_states: одномерный список состояний лучших сессий.
     :param Elite_actions: одномерный список действий лучших сессий.

    """

     from collections import defaultdict

     new_policy = np.zeros([n_states, n_actions])

    <ВАШ КОД: Установите вероятности для действий, которые привели к лучшим
состояниям и действиям >
```

```
    # Не забыть выставить 1/n_действий для всех действий в неизвестных
состояниях.

    return new_policy
```

**4. Тренировочный цикл**

Сгенерируйте сеансы, выберите N лучших и подстройтесь под них.

```
def show_progress(rewards_batch, log, percentile, reward_range=[-990, +10]):
    """
    Удобная функция, отображающая прогресс обучения
    """
    mean_reward = np.mean(rewards_batch)
    threshold = np.percentile(rewards_batch, percentile)
    log.append([mean_reward, threshold])

    plt.figure(figsize=[8, 4])
    plt.subplot(1, 2, 1)
    plt.plot(list(zip(*log))[0], label='Mean rewards')
    plt.plot(list(zip(*log))[1], label='Reward thresholds')
    plt.legend()
    plt.grid()

    plt.subplot(1, 2, 2)
    plt.hist(rewards_batch, range=reward_range)
    plt.vlines([np.percentile(rewards_batch, percentile)],
               [0], [100], label="percentile", color='red')
    plt.legend()
    plt.grid()
    clear_output(True)
    print("mean reward = %.3f, threshold=%.3f" % (mean_reward, threshold))
    plt.show()
```

```
# сбросить политику на всякий случай
policy = initialize_policy(n_states, n_actions)
```

```
#Эксперимент
n_sessions = 250     # число сессий
percentile = 50      # процент сессий с наивысшей наградой
learning_rate = 0.5  # насколько быстро обновляется политика, по шкале от 0
до 1

log = []

for i in range(100):
    sessions = [ <ВАШ КОД: генерирование списка n_sessions новых сессий> ]

    states_batch, actions_batch, rewards_batch = zip(*sessions)

    elite_states, elite_actions = <ВАШ КОД: выбор лучших состояний и действий
>

    new_policy = <ВАШ КОД: вычисление нового policy>

    policy = learning_rate * new_policy + (1 - learning_rate) * policy

    # display results on chart
    show_progress(rewards_batch, log, percentile)
```

**Задания к лабораторной работе**

***1. Написать свой код согласно заданиям для всех фрагментов, помеченных как: <ВАШ КОД:>***

```
<ВАШ КОД: >
```

***2. Запустить полученную программу, посмотреть на графиках процесс обучения.***

***3. Проанализировать как сходится задача такси и объяснить почему она быстро сходится от менее чем -1000 до почти оптимального значения, а затем снова снижается до -50/-100. Ответ приложить к отчету .***

## Требование к отчету

1. В качестве отчета принимается Python файл с кодом и текстовый файл с ответом на Задание № 3.
2. Код должен быть в достаточной мере прокомментирован.
3. Локальные датасеты также должны входить в отчет.
4. Отчет предоставляется в виде архива zip, формат имени архива: <номер группы>_<Фамилия_Имя>_LR<номер работы>.zip