1. Цель работы

   Реализовать алгоритм итерации по ценностям действий для среды FrozenLake;

   Исследовать реализованный алгоритм.

2. Задачи

   Изучить теоретическую часть работы;

   Реализовать алгоритм итерации по стратегиям для игры FrozenLake;

   Исследовать влияние функцию вознаграждения на выучиваемую стратегию поведения агента.

3. Краткие теоретические сведения

В данной лабораторной работе требуется реализовать алгоритм итерации по ценностям действий для среды FrozenLake, который относится к алгоритмам машинного обучения с подкреплением, активно развивающимся и широко применяемым в разного рода прикладных задачах.

Алгоритмы машинного обучения (МО) с подкреплением отличаются от обучения без учителя тем, что за решение задач в среде агенту назначается награда, эквивалентная тому, насколько хорошо агент решает задачу.

Агент – некоторая сущность, которая способна «существовать» в среде, а именно – выполнять какие-то действия в среде, наблюдать за изменениями этой среды и получать вознаграждение за выполнение действий в среде.

Среда FrozenLake – среда дискретна, представляет собой доску размерностью 4x4 (16 клеток). Агент в начале игры находится в левой верхней клетке и должен дойти до правой нижней клетки. На поле находятся проруби, в которые агент может провалиться и эпизод заканчивается. За достижение правой нижней клетки агент получает награду, равную 1 и эпизод заканчивается, иначе – 0 и эпизод так же заканчивается. У агента имеется 4 возможных действия – переход влево, вправо, вверх или вниз. Также, из любого положения с равной вероятностью p=0.33 агент может попасть в клетку правее, левее или выше, т.к. покрытие озера «скользкое» и агент может «поскользнуться».

В среде FrozenLake применяется Марковский процесс принятия решения – результат зависит только от текущего положения и последующих действий агента, но никак не от предыдущих состояний.

Коэффициент гамма вводится для того, чтобы уменьшить вклад сильно отсроченных вознаграждений, коэффициент дельта вводится для того, чтобы регулировать степень

обученности агента – на каждой итерации алгоритма, при обновлении таблицы, если все ее значения изменились меньше, чем на дельта, мы считаем, что агент обучен достаточно.

4. Выполнение работы

Для выполнения работы будем использовать пакет gym, реализующий среду FrozenLake, в которой агент будет существовать, выполнять действия и получать награды, также пакет tensorboardX для формирования графиков, позволяющих оценить процесс обучения при разных входных данных.

5. Листинг

```
import gym

import collections

from tensorboardX import SummaryWriter

import os

import shutil


ENV_NAME = "FrozenLake-v1"

DELTA = 0.001 #0.0000001

GAMMA = 0.99 #количество шагов, за которое агенту следует проходить эпизод
(отвечает за оптимизацию)

TEST_EPISODES = 20


class Agent:

    def __init__(self):

        # завели среду, чтобы набрать опыт

        self.env = gym.make(ENV_NAME)

        # состояние (новый эпизод) - индекс клетки

        self.state = self.env.reset()

        # таблица награждений, представляющая собой словарь

        self.reward_table = collections.defaultdict(float)

        # таблица переходов

        self.transition_table = collections.defaultdict(collections.Counter)

        self.Q = collections.defaultdict(float)

        self.Q_prev = collections.defaultdict(float)

    def play_n_random_steps(self, count):

        for _ in range(count):

            # просторасноство действий

            action = self.env.action_space.sample()

            new_state, reward, is_done, _ = self.env.step(action)

            self.reward_table[(self.state, action, new_state)] = reward

            self.transition_table[(self.state, action)][new_state] += 1

            self.state = self.env.reset() if is_done else new_state

    def select_action(self, state):
```

```
        best_action, best_value = None, None
        for action in range(self.env.action_space.n):
            action_value = self.Q[(state, action)]
            if best_action is None or best_value < action_value:
                best_value = action_value
                best_action = action
        return best_action
    def play_episode(self, env, do_rendering=False):
        total_reward = 0.0
        state = env.reset()
        while True:
            action = self.select_action(state)
            new_state, reward, is_done, _ = env.step(action)
            if do_rendering:
                env.render()
            self.reward_table[(state, action, new_state)] = reward
            self.transition_table[(state, action)][new_state]+=1
            total_reward += reward
            if is_done:
                break
            state = new_state
        return total_reward
    # из какого состояния наилучшее действие будет адекватным (таблица ценнсости действий)
    def Q_iterarion(self):
        self.Q_prev = self.Q.copy()
        for state in range(self.env.observation_space.n):
            for action in range(self.env.action_space.n):
                action_value = 0.0
                target_counts = self.transition_table[(state, action)]
                total = sum(target_counts.values())
                for tgt_state, count in target_counts.items():
                    reward = self.reward_table[(state, action, tgt_state)]
                    best_action = self.select_action(tgt_state)
```

```
                    action_value +=
(count/total)*(reward+GAMMA*self.Q[(tgt_state, best_action)])
                self.Q[(state, action)] = action_value


if __name__ == '__main__':
    test_env = gym.make(ENV_NAME)
    agent = Agent()
    writer = SummaryWriter(comment='-q-iteration')

    iter_no = 0
    best_reward = 0.0
    is_done = False
    while not is_done:
        iter_no += 1
        # играем 100 раз разными ходами (набираем опыт)
        agent.play_n_random_steps(100)
        # посмотрели, какие действия более ценные
        agent.Q_iterarion()

        reward = 0.0
        # играем количеством эпизодов, считаем награждение
        for _ in range(TEST_EPISODES):
            reward += agent.play_episode(test_env)
        reward /= TEST_EPISODES
        writer.add_scalar("reward", reward, iter_no)
        if reward > best_reward:
            print("Best reward updated {} -> {}".format(best_reward,reward))
            best_reward = reward
        # if reward > 0.90:
           # print("Solved in %d iterations!" % iter_no)
           # break
        if iter_no > 15:
            for state_action in agent.Q.keys():
                # print(state_action)
```

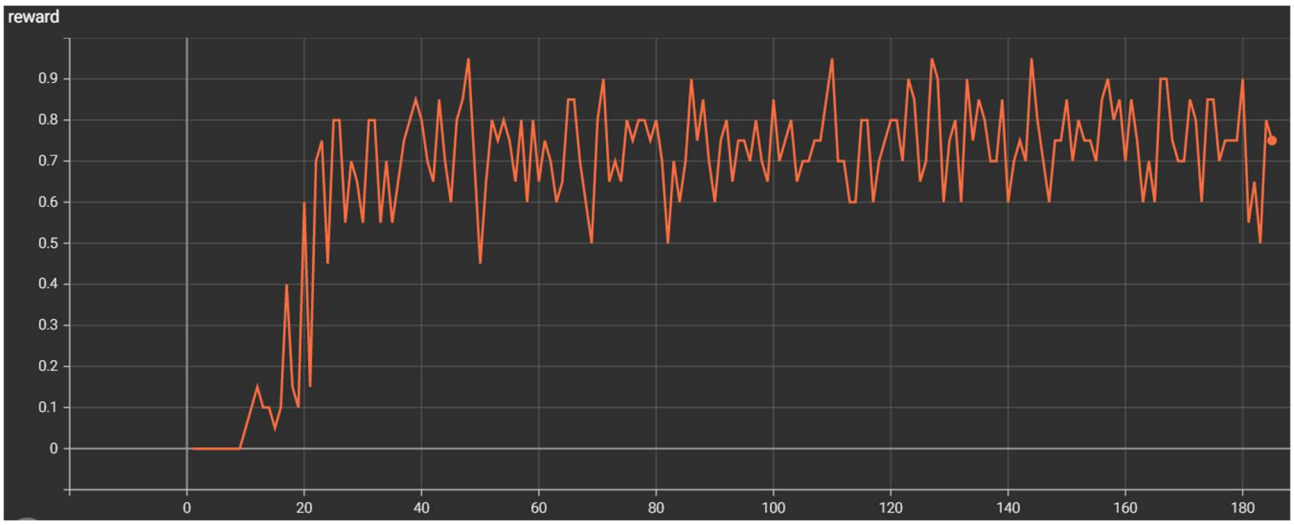
```
                if abs(agent.Q[state_action]-agent.Q_prev[state_action]) <
DELTA:
                   is_done = True
                else:
                    is_done = False
                    break
        if is_done:
            print("Solved in %d iterations!" %iter_no)
            break
    if os.path.exists("recording"):
        shutil.rmtree("recording")
    env = gym.make(ENV_NAME)
    agent.play_episode(env, True)
    writer.close()


    # tensorboard - -logdir "E:\study\8
sem\Artificial_intelligence_systems\runs"


    # мы ориентируемся не на нашу таблицу ценности действий, а на среднее
значение награды
    # при 100 эпизодах меньше вероятность выиграть
```

6. Результаты работы

6.1. Эксперимент для gamma = 0.99, delta = 0.001

```
Best reward updated 0.0 -> 0.05
Best reward updated 0.05 -> 0.1
Best reward updated 0.1 -> 0.15
Best reward updated 0.15 -> 0.4
Best reward updated 0.4 -> 0.6
Best reward updated 0.6 -> 0.7
Best reward updated 0.7 -> 0.75
Best reward updated 0.75 -> 0.8
Best reward updated 0.8 -> 0.85
Best reward updated 0.85 -> 0.95
Solved in 185 iterations!
```

Рисунок 1 - Динамика изменения наград в зависимости от количества совершенных итераций

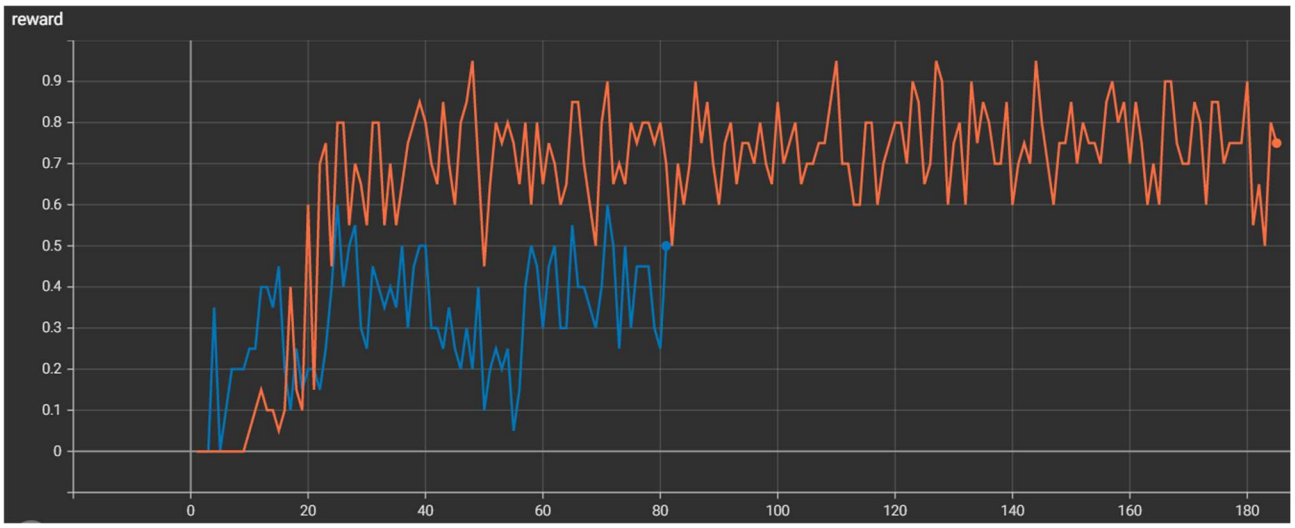



Рисунки 2 – График результата

6.2. Эксперимент для gamma = 0.7, delta = 0.001

```
Best reward updated 0.0 -> 0.35
Best reward updated 0.35 -> 0.4
Best reward updated 0.4 -> 0.45
Best reward updated 0.45 -> 0.6
Solved in 81 iterations!
```

Рисунок 3 - Динамика изменения наград в зависимости от количества совершенных итераций




Рисунки 4 – График результата (синий)

7. Выводы

В ходе выполнения лабораторной работы был проведен эксперимент с изменением значения Гамма. Исходя из графиков на рисунке 4 можно сделать вывод, что увеличение гаммы дает большее количество итераций (агент будет дольше обучаться), а уменьшение может привести к ошибке обучения, а именно попадание в локальный минимум.