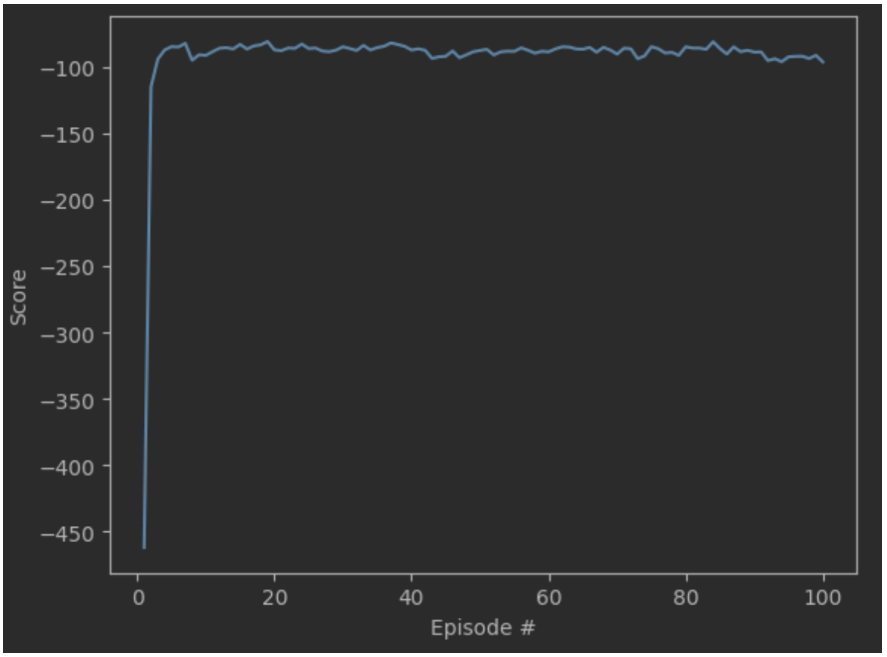
Домашняя работа #2 RL

Acrobot-v1

Обучение

Эксперимент 1

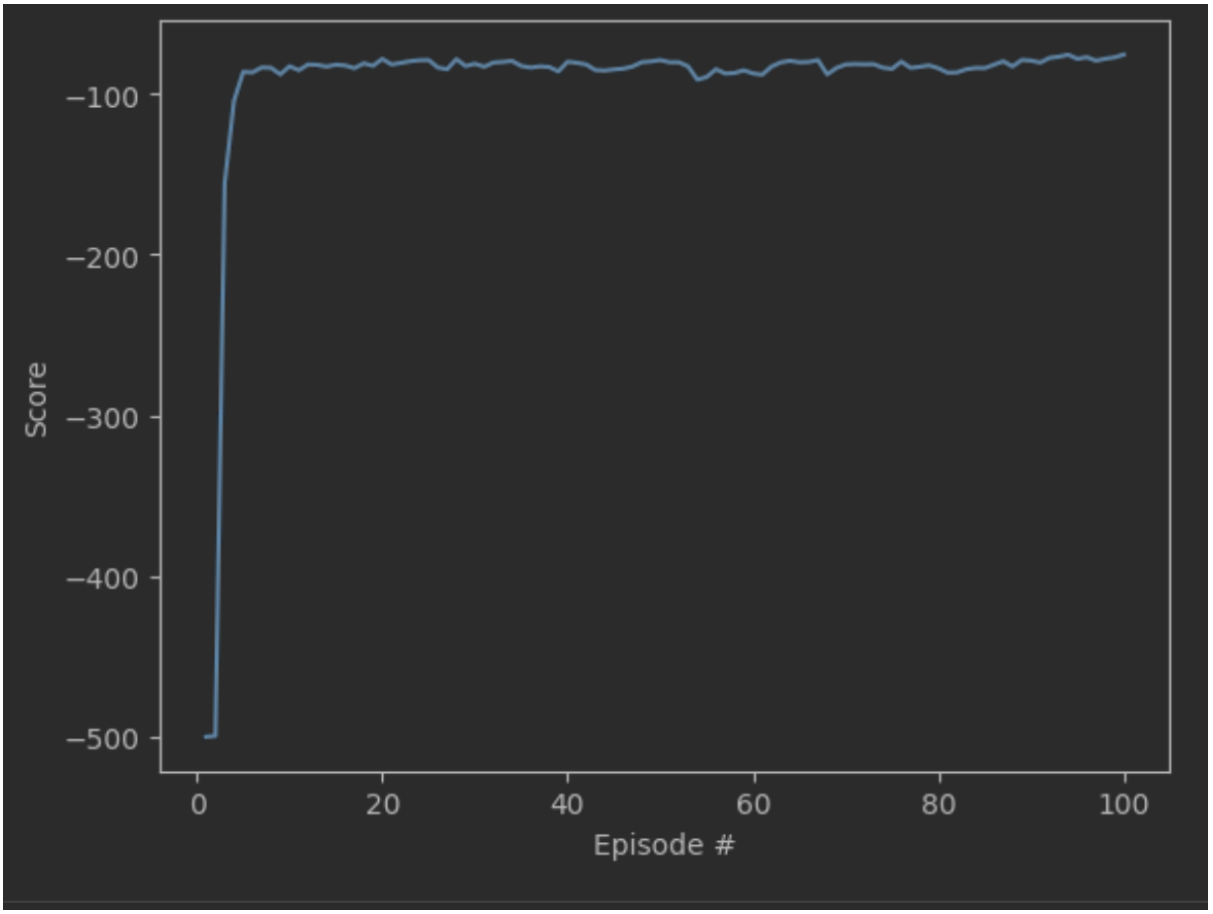
Переписал модель с лекции на Acrobot-v1
Параметры были следующими:
Количество эпизодов - 100
Количество траекторий в эпизоде - 200
Длина каждой траектории - 500
q-параметр -0.8
lr-0.01
нейросеть усложнил. Захотелось поиграться.



Эксперимент 2

Так как первый эксперимент уже дал неплохие для акробота результаты было принято решение сильно не изменять параметры:
Количество эпизодов - 100
Количество траекторий в эпизоде - 200
Длина каждой траектории - 500
q-параметр -0.7
lr-0.01

Это увеличило средний скор(Численные значения были около 80-78):



Вывод

Так как обучался акробот вторым, он был значительно легче для понимания и решился почти со второго эксперимента. Видимо из-за более низкого квантиля какие-то изначально более хорошие действия попали в элитные, и в конце обучили модель лучше. Было бы понятнее к чему стремиться если бы были заданы какие-то бейзлайны которые стоит побить.

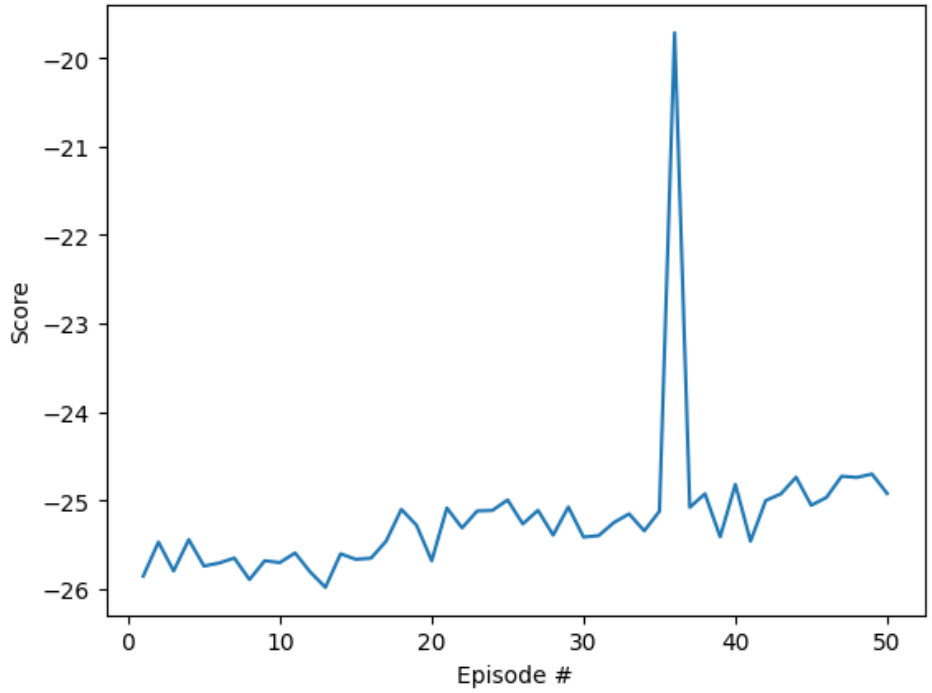
Mountainous-car-v1

Обучение

Эксперимент 1

Optimizer - Adam
LR - 0.001
Нейросеть - как в практическом задании

Количество эпизодов - 50
Количество траекторий в эпизоде - 20
Длина каждой траектории - 500
q-параметр -0.8

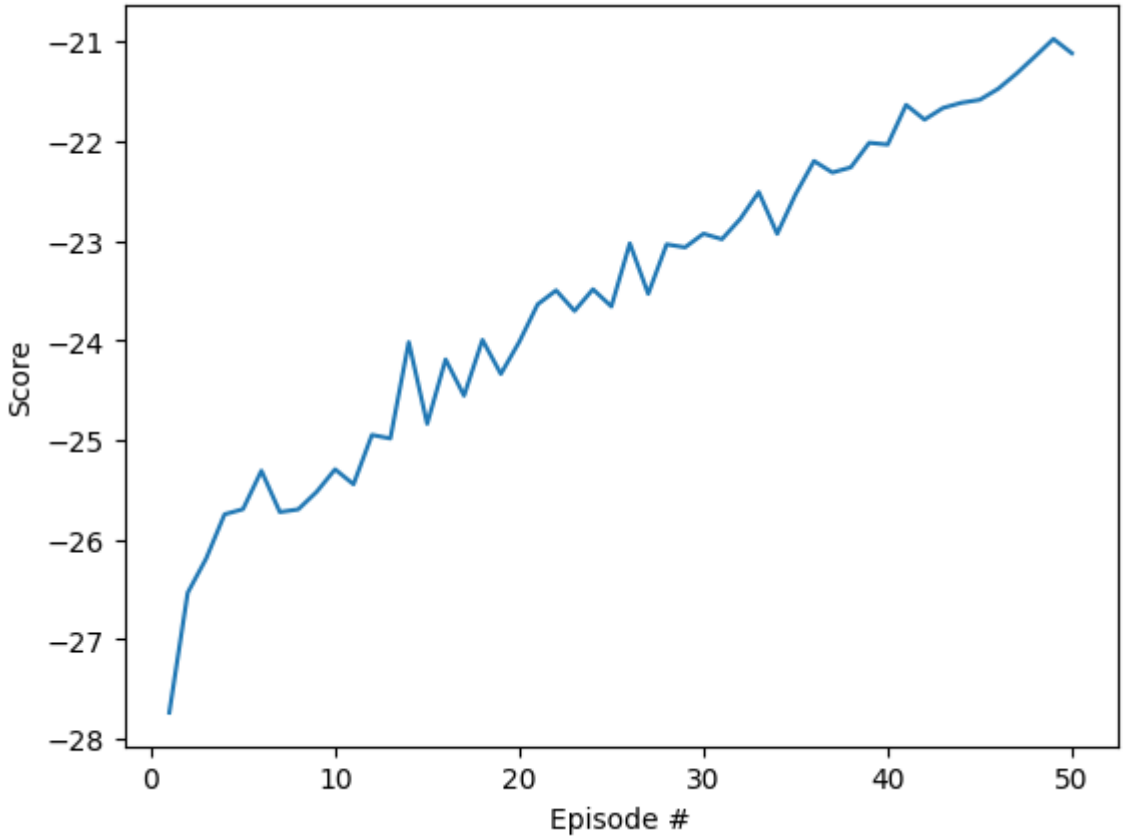


Модель почти не обучалась, и стояла на месте. Чем обусловлен резкое увеличение скора непонятно.

Эксперимент 2

Optimizer - Adam
LR - 0.005
Нейросеть - как в практическом задании
Количество эпизодов - 50
Количество траекторий в эпизоде - 20
Длина каждой траектории - 500

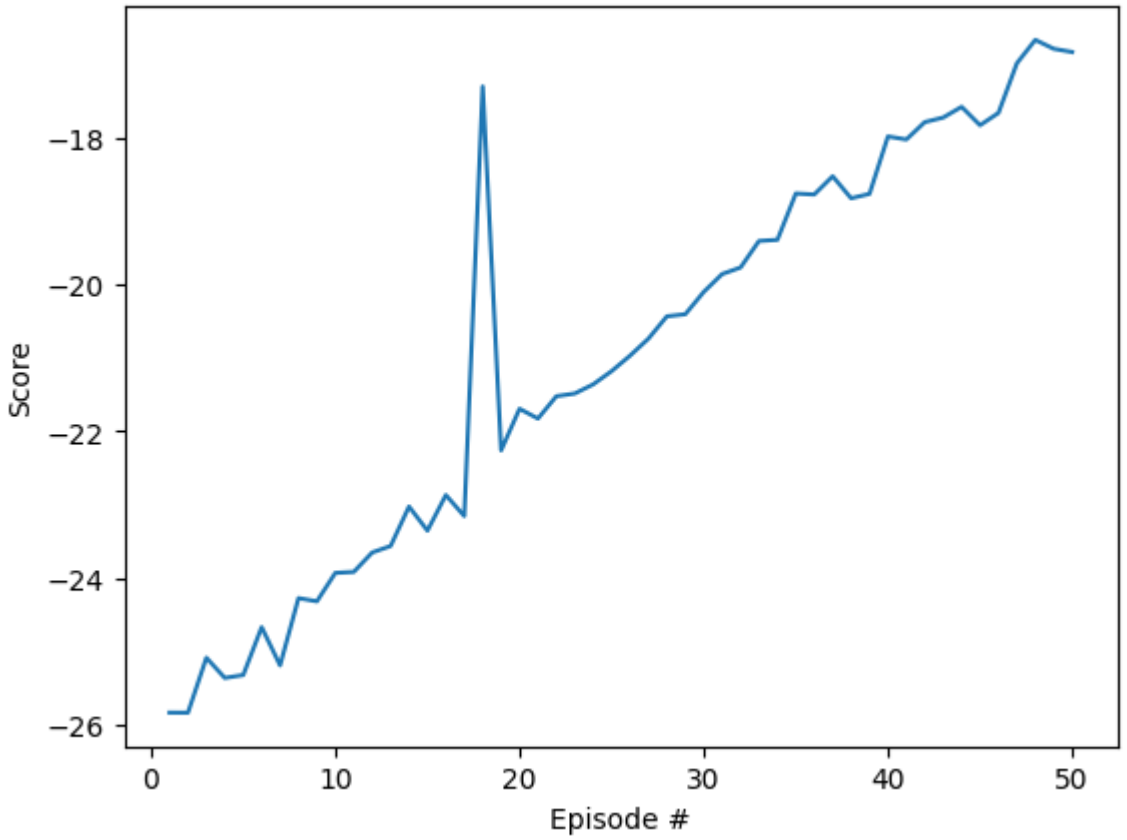
q-параметр -0.8



Уже лучше.

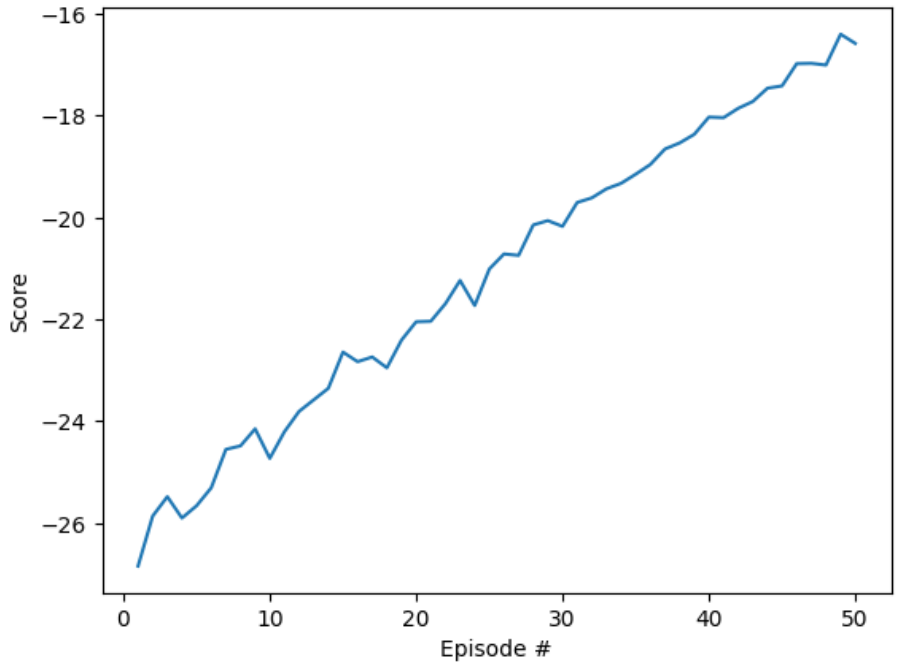
Эксперимент 3

Optimizer - Adam
LR - 0.01
Нейросеть - как в практическом задании
Количество эпизодов - 50
Количество траекторий в эпизоде - 20
Длина каждой траектории - 500
q-параметр -0.8



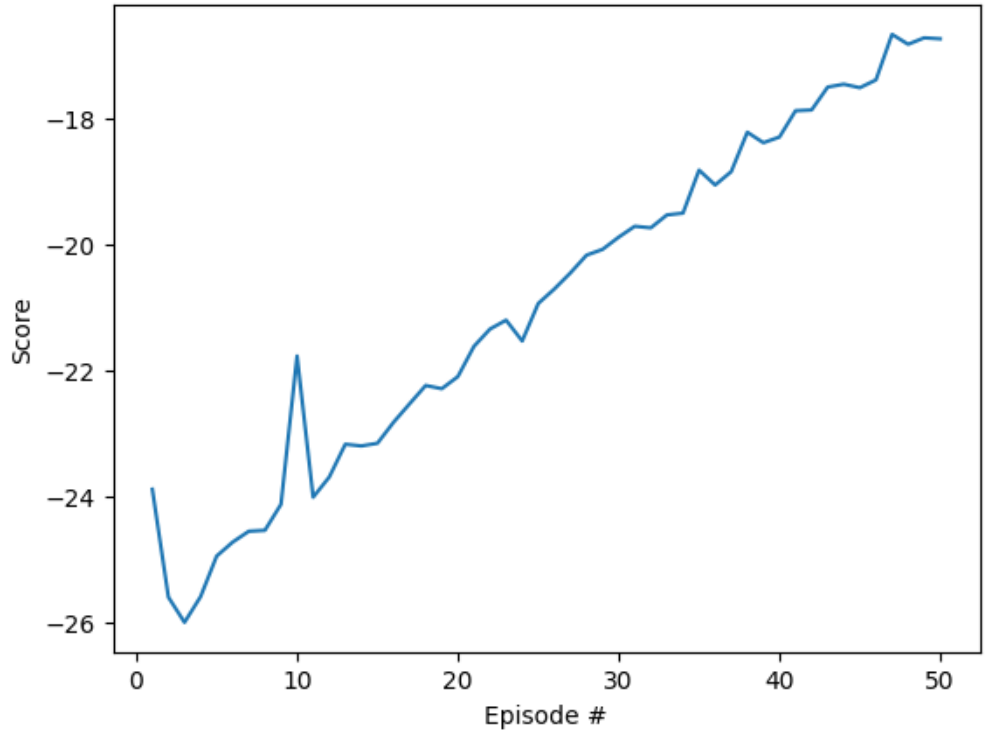
Эксперимент 4

Optimizer - Adam
LR - 0.01
Нейросеть - как в практическом задании
Количество эпизодов - 50
Количество траекторий в эпизоде - 20
Длина каждой траектории - 500
q-параметр -0.7



Эксперимент 5

Optimizer - Adam
LR - 0.01
Нейросеть - как в практическом задании
Количество эпизодов - 50
Количество траекторий в эпизоде - 40
Длина каждой траектории - 500
q-параметр -0.7



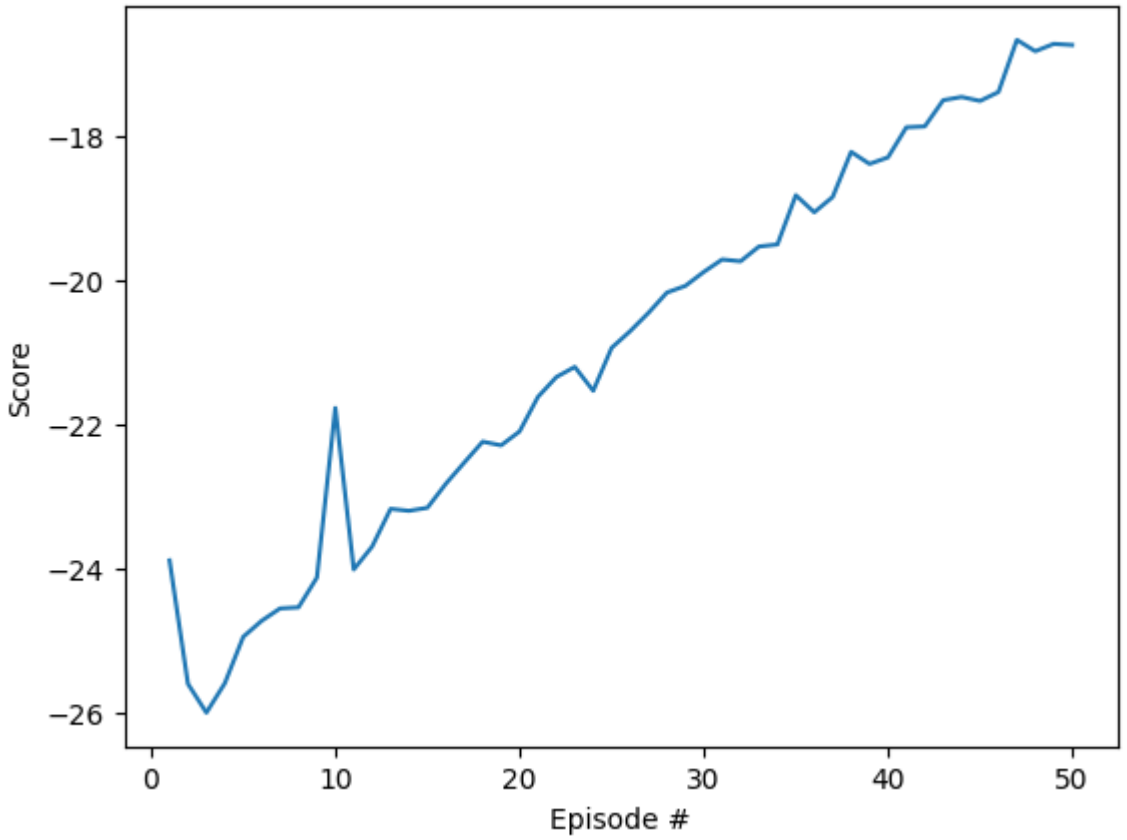
На этом эксперименте я понял что манишка максимизирует награду, и уменьшает амплитуду движений, и это нехорошо.

Эксперимент 6

Добавил шум к награде

```
def get_trajectory(env, agent, trajectory_len, reward_noise_std=0.1,
visualize=False):
   trajectory = {'states': [], 'actions': [], 'total_reward': 0}

   state = env.reset()
   trajectory['states'].append(state)

   for _ in range(trajectory_len):

       action = agent.get_action(state)
       trajectory['actions'].append(action)

       state, reward, done, _ = env.step([action])

       # Add noise to reward
       reward += np.random.normal(0, reward_noise_std)

       trajectory['total_reward'] += reward

       if done:
           break

       if visualize:
           env.render()

       trajectory['states'].append(state)

   return trajectory
```

Optimizer - Adam
LR - 0.01
Нейросеть - как в практическом задании
Количество эпизодов - 50
Количество траекторий в эпизоде - 20
Длина каждой траектории - 500
q-параметр -0.9
reward_noise_std -0.1



Машинка не доезжает до конца. В чате пишут, что так не должно быть.

Эксперимент 7

Нашел в интернете код. Переписал получение награды вот в такой вид. Добавился параметр t

```
def get_trajectory(env, agent, trajectory_len, reward_noise_std=1, t = 1000,
visualize=False):
    trajectory = {'states': [], 'actions': [], 'total_reward': 0}

    state = env.reset()
    trajectory['states'].append(state)

    for _ in range(trajectory_len):

        action = agent.get_action(state)
        trajectory['actions'].append(action)

        state, reward, done, _ = env.step([action])

        # Add noise to reward
        reward += reward * math.pow(reward_noise_std, t)
        trajectory['total_reward'] += reward

        if done:
            break

        if visualize:
```

```
            env.render()

        trajectory['states'].append(state)

    return trajectory
```

Эксперимент 8

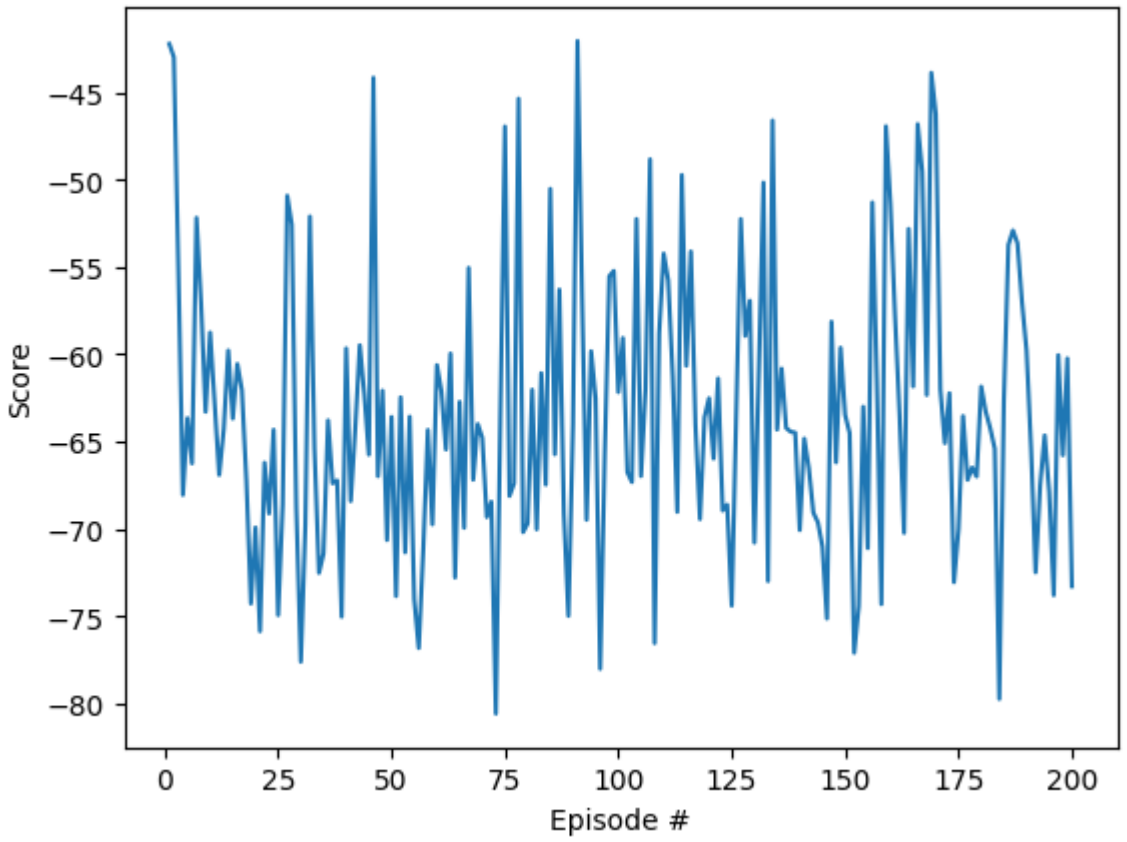
После чтения чата стало понятно, что Эксперимент 7 и все предыдущие тупиковая ветвь развития.
Было добавлен шум к action

```
noise = np.random.normal(0.9, 1.1, size=action.shape)
action += noise
```

А также отбираться в элитные начали траектории больше 50

```
total_rewards = [trajectory['total_reward'] for trajectory in trajectories if trajectory['total_reward'] > 50]
```

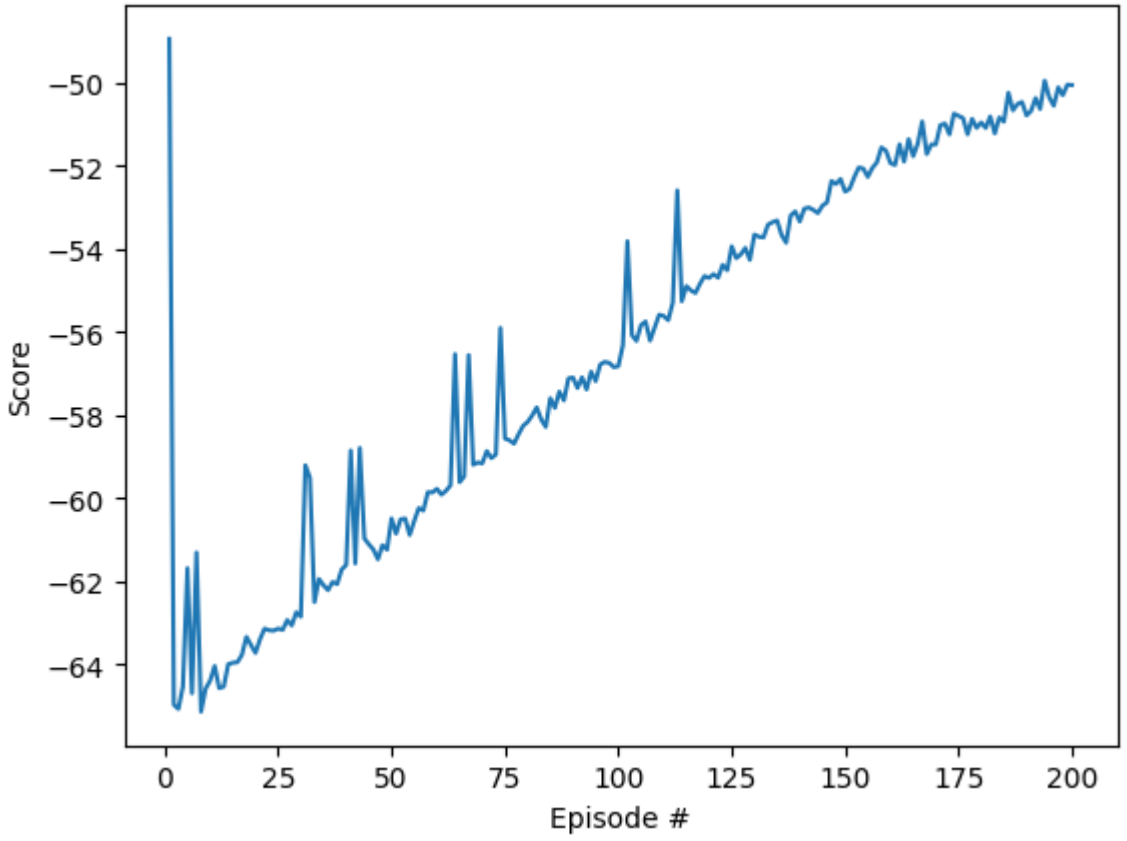
Optimizer - Adam
LR - 0.1
Нейросеть - уменьшилась до 10 нейронов
Количество эпизодов - 200
Количество траекторий в эпизоде - 40
Длина каждой траектории - 999
q-параметр -0.9



Получилось что-то рандомное, но зато были траектории, которые достигали вершины.

Эксперимент 9

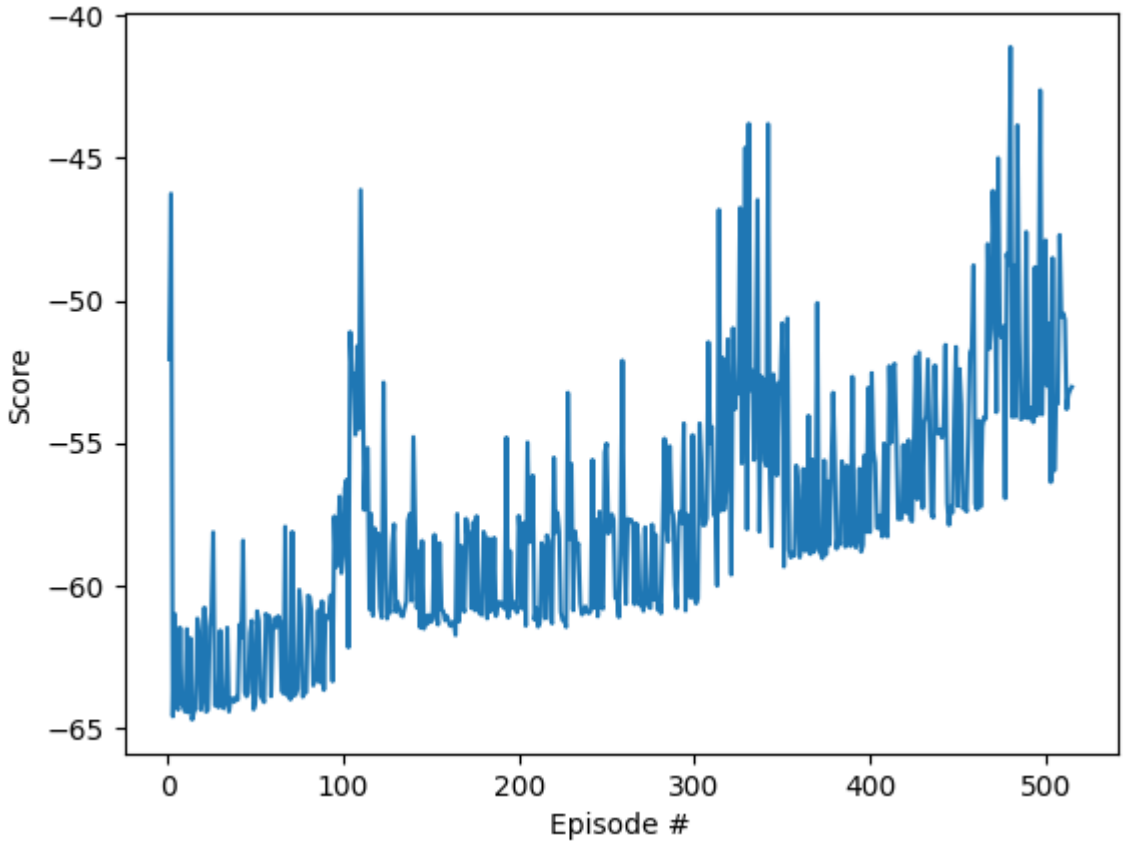
Теперь шум должен затухать со временем. Это должно улучшить качество обучения.
Параметры те же
Optimizer - Adam
LR - 0.1
Нейросеть - уменьшилась до 10 нейронов
Количество эпизодов - 200
Количество траекторий в эпизоде - 40
Длина каждой траектории - 999
q-параметр -0.9



Теперь это больше похоже, на обучение, но результат еще совсем слаб

Эксперимент 10

Поднял количество эпизодов до 2000. q-параметр до 0.1



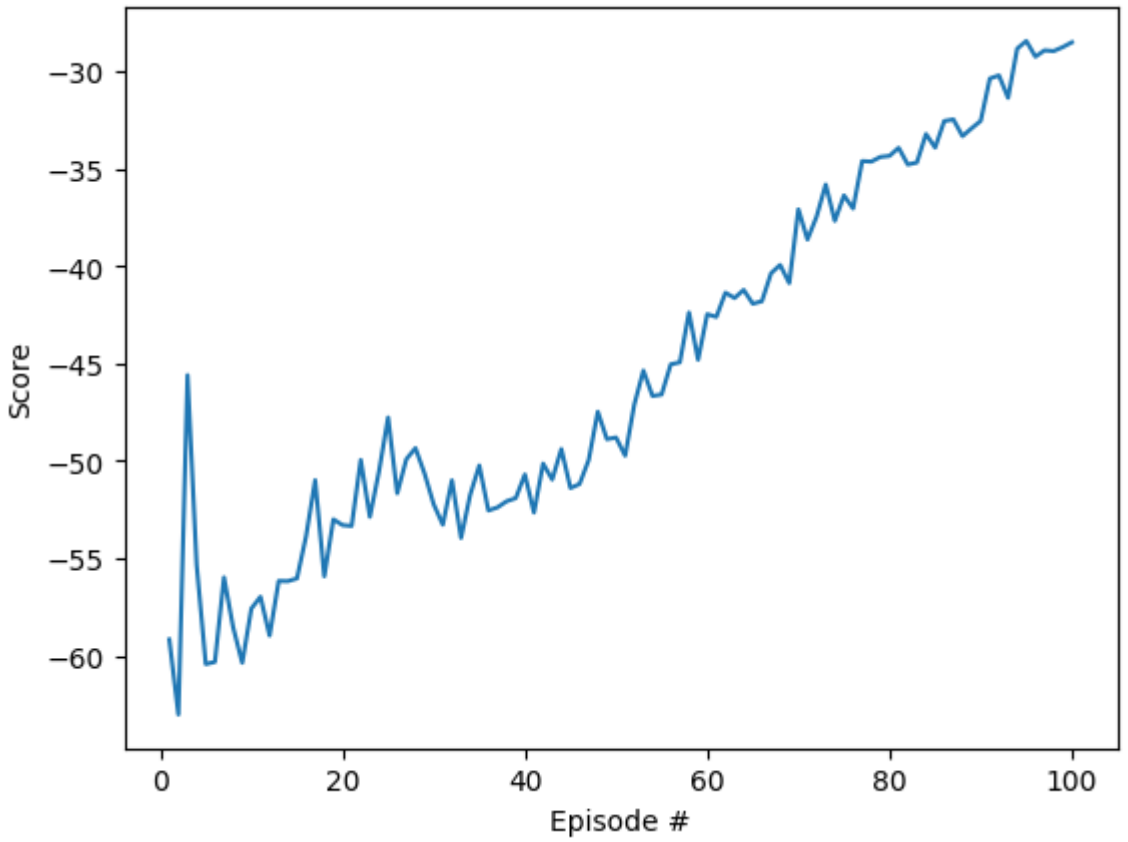
Приблизительно на 500 эпизодах терпение закончилось. В среднем было видно, что модель училась, но очень медленно.

Эксперимент 11

Был принято внедрить подход с repetition на выигравших траекториях
Optimizer - Adam
LR - 0.1
Нейросеть - уменьшилась до 10 нейронов
Количество эпизодов - 100
Количество траекторий в эпизоде - 100
Длина каждой траектории - 999
q-параметр -0.7
elite-weight - 3 - 20 В зависимости от траектории

```
def update_policy(self, elite_trajectories, elite_weight=3):
    elite_states = []
    elite_actions = []

    for trajectory in elite_trajectories:
        for _ in range(elite_weight):  # Repeat elite trajectories
            elite_states.extend(trajectory['states'])
            elite_actions.extend(trajectory['actions'])

    elite_states = torch.FloatTensor(elite_states)
    elite_actions = torch.FloatTensor(elite_actions)
```

```
means = self.forward(elite_states)
log_stds = self.log_std.expand_as(means)
stds = log_stds.exp()

normal = torch.distributions.Normal(means, stds)
log_probs = normal.log_prob(elite_actions).sum(-1)
loss = -log_probs.mean()

self.optimizer.zero_grad()
loss.backward()
self.optimizer.step()
```



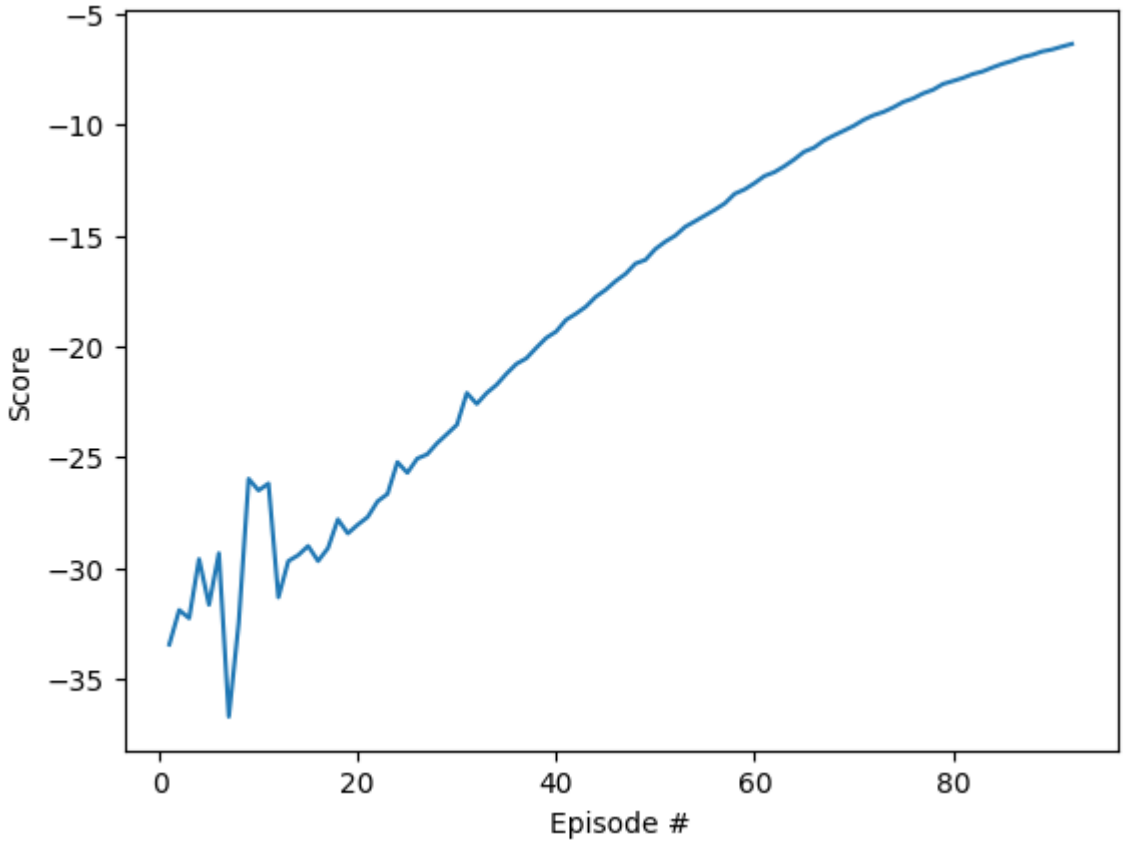
Модель демонстрирует обучение.

Эксперимент 12

```
q_param = start_q - episode * (start_q - end_q) / total_episodes
```

Сделал q_pararm динамичным
Уменьшил начальный параметр для затухания шума до 1.0
Optimizer - Adam
LR - 0.1
Нейросеть - уменьшилась до 10 нейронов
Количество эпизодов - 100
Количество траекторий в эпизоде - 100
Длина каждой траектории - 999
q-параметр -0.7

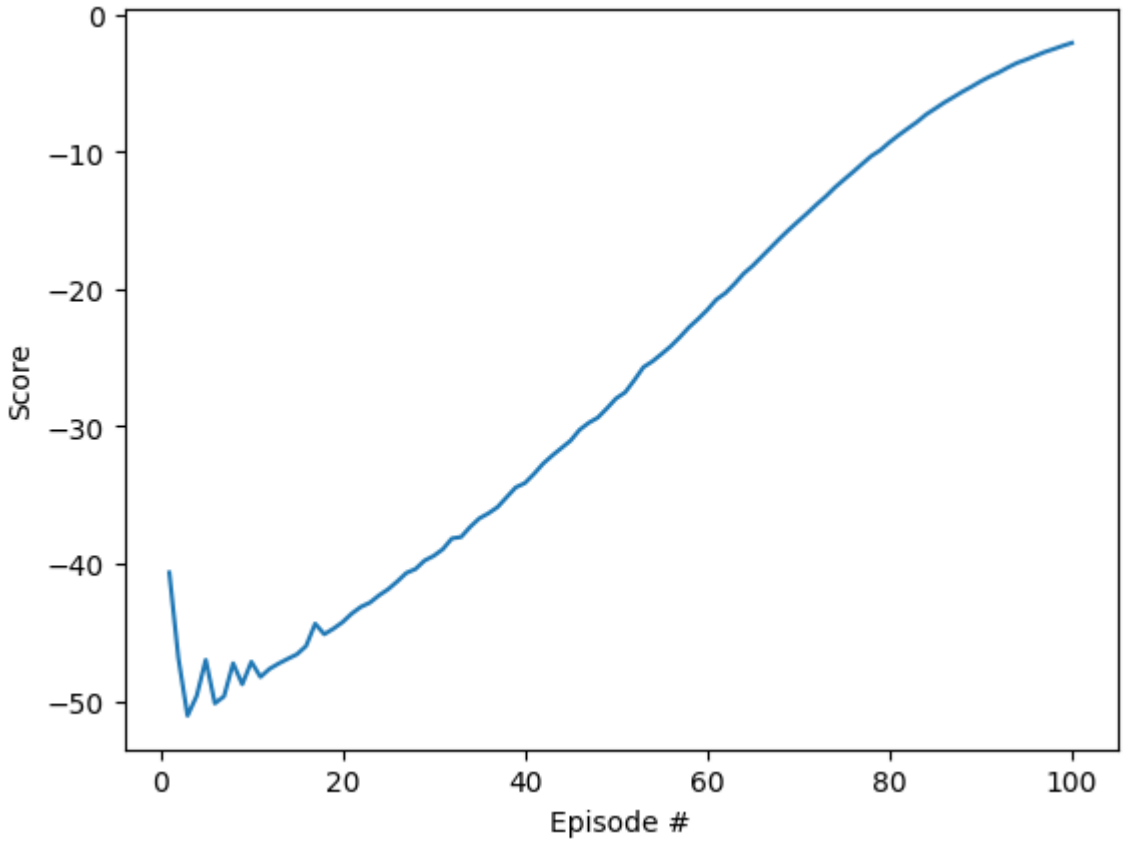
elite-weight - 3 - 10 В зависимости от траектории
Понял, что очень некорретно генерирую шум. Переписал генерацию шума на uniform.



Понял, что происходит. Модель обучалась на шумных траекториях с ревардом больше 50, но в какой-то момент без шума перестала генерировать выигрышные стратегии, и впоследствии просто уменьшался шум, и модель двигалась к точности.

Эксперимент 13

Увеличил начальный параметр для затухания шума до 1.0
Optimizer - Adam
LR - 0.1
Нейросеть - уменьшилась до 10 нейронов
Количество эпизодов - 100
Количество траекторий в эпизоде - 100
Длина каждой траектории - 999
q-параметр -0.7
elite-weight - 3
Шум динамически убывает. Теоретически его можно вообще не уменьшать до конца

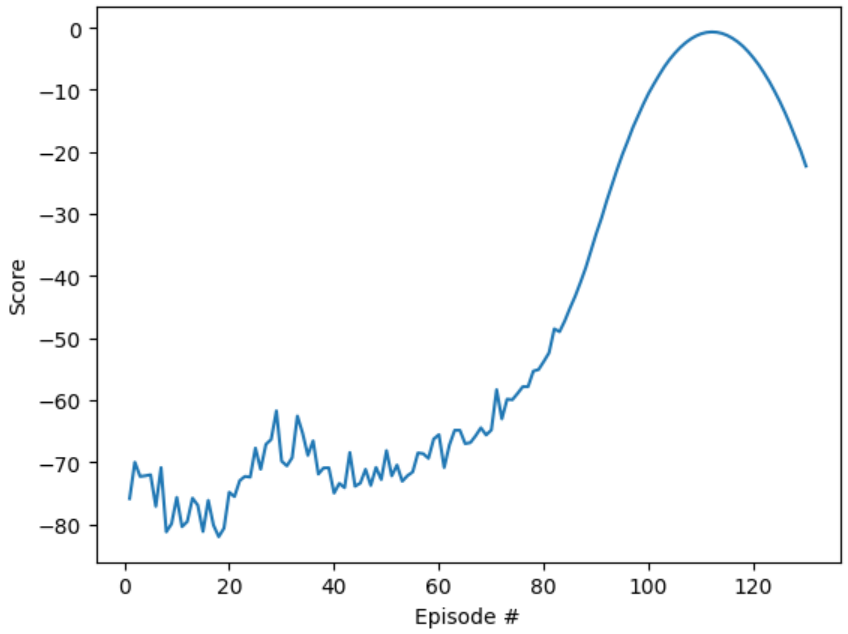


Эксперимент 14

Стало ясно, что генерируется очень мало элитных траекторий, и необходимо что-то с этим сделать. Поменял генерацию шума на следующую :

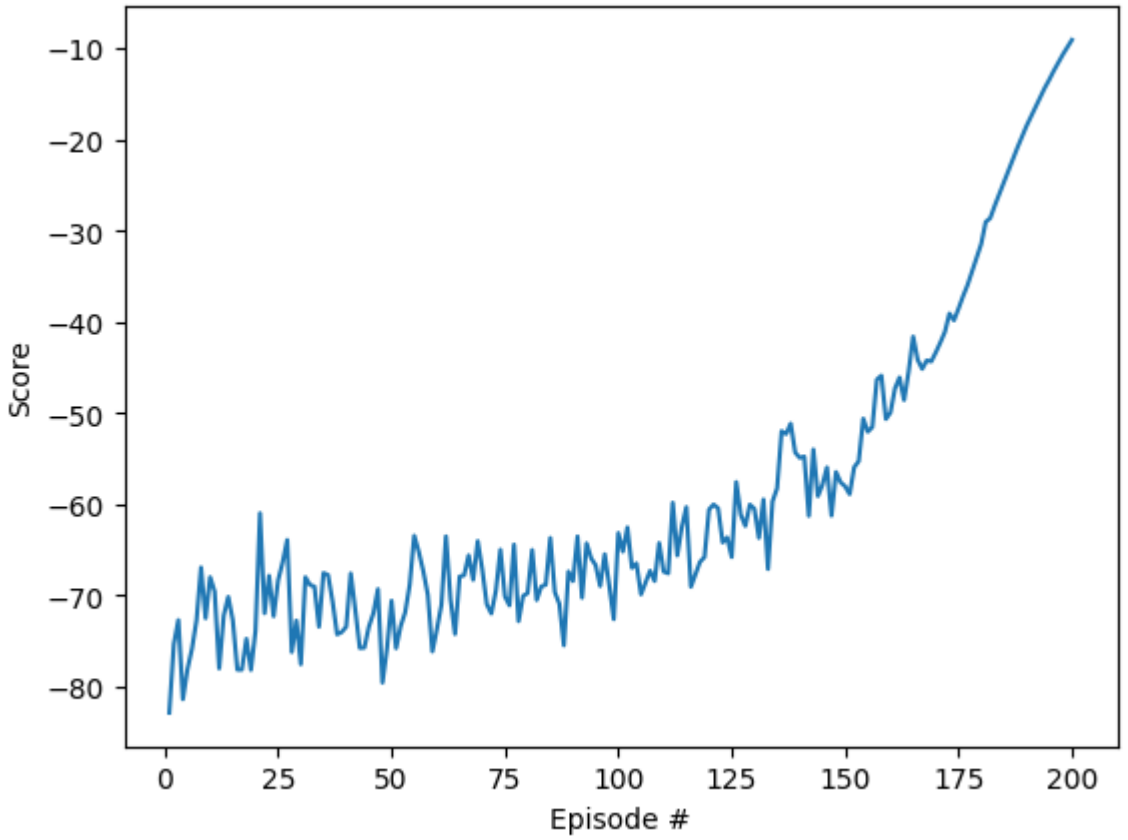
```
noise =  (2*np.random.sample() - 1)*5
```

Увеличил количество эпизодов до 130. Несмотря на то что удается приблизиться к 0 reward не получается перешагуть планку в 0.



Эксперимент 15

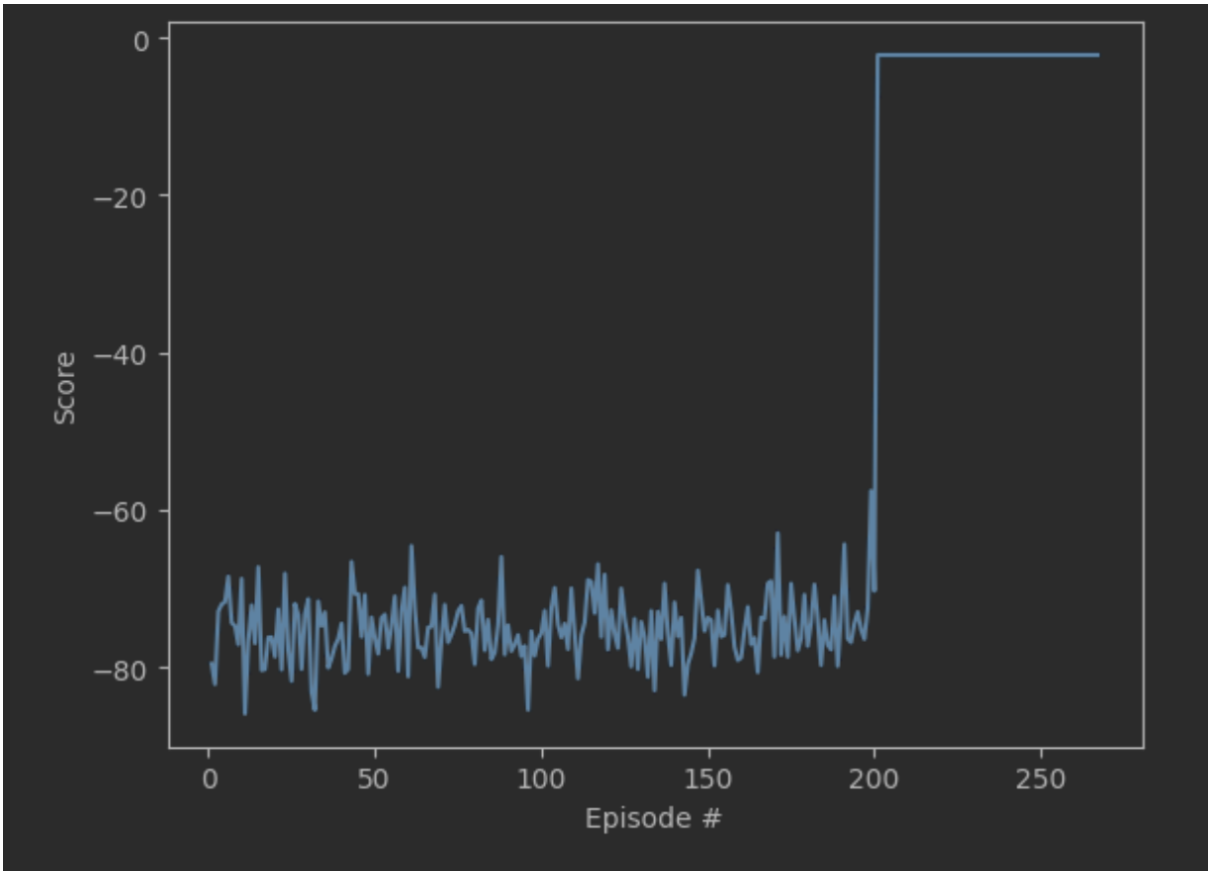
Решил поменять reward ограничение для элитных с 50 на 0. Убрал вес элитности, и решил увеличить количество итераций до 200. Квантиль будет расти с 0.1 до 0.9



Попал в тупик. Не знаю, что делать. Машинка просто раскачивается из право в лево.

Эксперимент 16

Решил тренировать модель в два этапа 1) с большим количеством шума и lr1 2) с меньшим шумом и более чувствительным lr.



Модель как будто не тренируется после определенных значений.

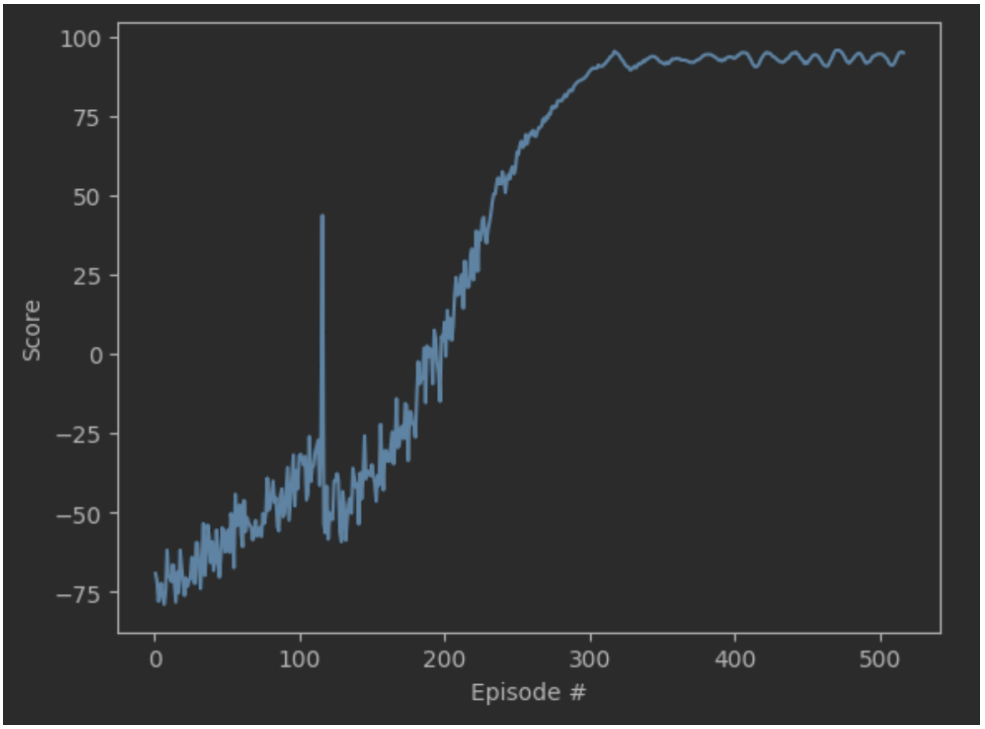
Эксперимент 17

Добавил буффер для элитных траекторий. Результат такой же.

Эксперимент 18

Переписал саму сеть. Поменял loss function на L1loss(). Оказалось там была ошибка мешавшая обучению. Стоило обратить на это внимание заранее. Обучал 300 эпизодов с уменьшающимся шумом, и еще 200 с малым константным шумом. Помимо этого динамически возрастал квантиль с 0.1 до 0.8. Количество траекторий - 100. Длина - 999.

В результате mean total award стремилась к 95, что хорошо.



Вывод

Научился большому количеству интересных вещей:
Буферу элитных траекторий, влиянию шума на изначальные параметры модели, локальные оптимумы, и прочему. Стоило изначально более внимательно подойти к параметрам самой сети, и проверить ее обучаемость. Большое внимание к траекториям генерируемым шумами меня сбило. Интересно, то что сама сеть не так важна, и значительно важнее гиперпараметры.