Задание # 1 DQN

Найти оптимальные гиперпараметры. Сравнить с алгоритмом Deep Cross-Entropy на графиках. Был выбрана среда Acrobot-v1.

Подбирались следующие гиперпараметры:

- episode_n_arr = (100, 200, 300)
- t_max_arr = (500, 1000)
- layer_size_arr = (64, 128, 256)
- gamma_arr = (0.9, 0.99)

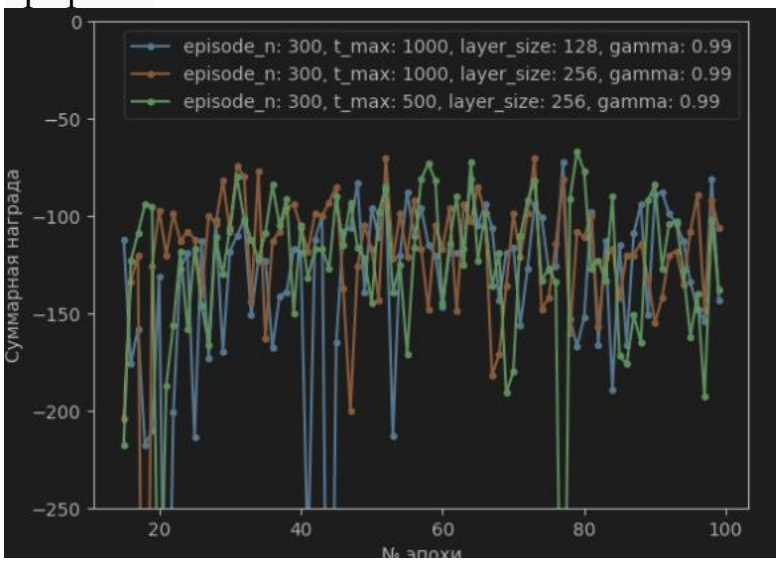
Кол-во эпох, кол-во итераций в эпохе, размер слоя в NN и гамма соотвестенно.

Рассмотрим сводную таблицу результатов экспериментов, обратите внимание, что значения взяты на последней итерации mean_total_reward, и last_total_reward:

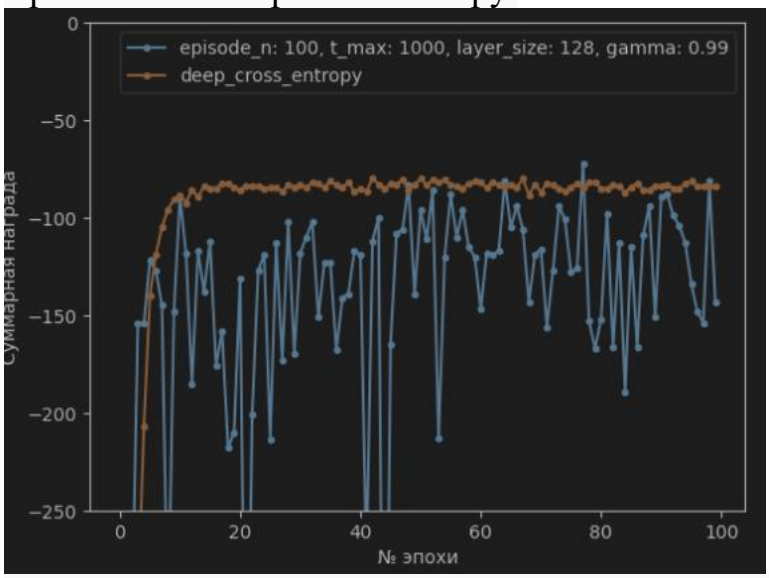
	episode_n	t_max	layer_size	gamma	mean_total_reward	last_total_reward
33	300	1000	128	0.99	-122.763333	-74.0
35	300	1000	256	0.99	-116.063333	-76.0
29	300	500	256	0.99	-122.973333	-86.0
9	100	1000	128	0.99	-148.190000	-91.0
17	200	500	256	0.99	-129.440000	-94.0
27	300	500	128	0.99	-120.030000	-94.0
13	200	500	64	0.99	-135.900000	-97.0

Какие выводы можно сделать, чем больше кол-во эпох и размер итераций в эпохе, тем лучше мы остановились на 300, но можно еще улучшить. Однако улучшаться последняя награда будет все меньше и меньше. Что касается размер слоя в NN, то можно смело утверждать, что нужно брать 128 и более нейронов, но более 256 нет смысла брать, столько параметров избыточно. Для gamma получилось значение 0.99.

Графики:



Сравнение с Deep Cross entropy:



Как мы видим DCE алгоритм выходит на плато и более не улучшается, а так как мы оставили min eps у DQN, то он нестабилен, однако видно, что он на 77 итерации превосходит по суммарной награде DCE. И из очевидных плюсов DQN быстрее учится и его нужно лишь стабилизировать.

Задание # 2 Реализовать с сравнить (на выбранной ранее среде) друг с другом и с обычным DQN следующие его модификации:

DQN c Hard Target Update;

DQN c Soft Target Update;

Double DQN.

Задание # 2.1 DQN c Hard Target Update

DQN с HARD Target Networks это модификация базового алгоритма DQN, предназначенная для стабилизации обучения и улучшения его сходимости

Имеет две сети - основную сеть (online network) и целевую сеть (target network).

Обновление целевой сети происходит жестким (hard) копированием параметров из основной сети с некоторой фиксированной периодичностью.

Реализация:

```
self.q_function = Qfunction(self.state_dim, self.action_dim, layer_size)  # основная сеть
self.target_q_function = Qfunction(self.state_dim, self.action_dim, layer_size)  # добавляем целевую сеть
self.target_q_function.load_state_dict(self.q_function.state_dict())  # копируем параметры из основной сети
```

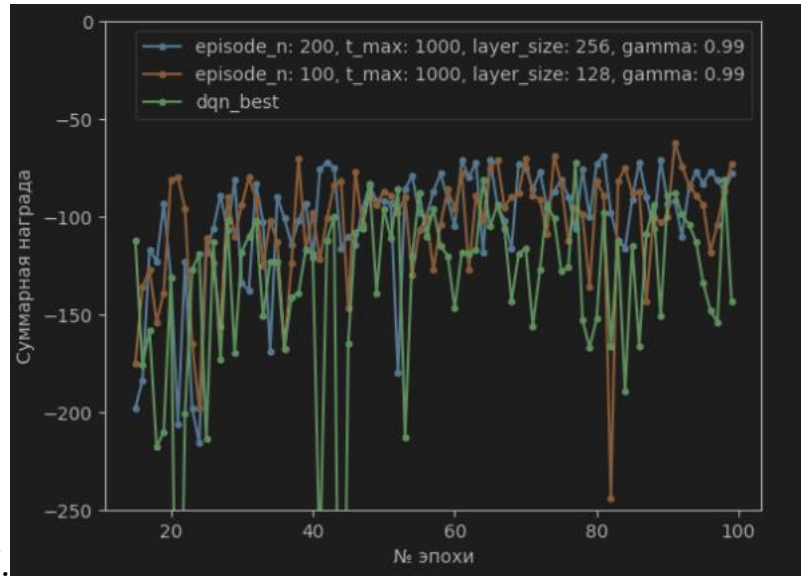
```
# Обновление целевой сети
self.timestep += 1
if self.timestep % self.target_update_freq == 0:
    self.target_q_function.load_state_dict(self.q_function.state_dict())
```

Рассмотрим сводную таблицу результатов экспериментов, обратите внимание, что значения взяты на последней итерации mean_total_reward, и last_total_reward:

	episode_n	t_max	layer_size	gamma	mean_total_reward	last_total_reward
23	200	1000	256	0.99	-102.580000	-64.0
9	100	1000	128	0.99	-126.640000	-73.0
3	100	500	128	0.99	-118.950000	-74.0
33	300	1000	128	0.99	-109.790000	-74.0
29	300	500	256	0.99	-101.136667	-76.0
15	200	500	128	0.99	-114.575000	-77.0
17	200	500	256	0.99	-113.985000	-81.0
27	300	500	128	0.99	-109.786667	-81.0

Можно обратить внимание, что метрики награды стали лучше благодаря более стабильному обучению. Более того, лучшая модель с 256 размерностью слоев (увеличилось) и меньшим кол-вом эпох 200, но по-прежнему с максимальным t_max и gamma.

На графике видно, что HardDQN стабильнее обучается, нет огромных просадок по награде и видно, что график награды преймущественно больше чем у DQN.



Задание # 2.2 DQN с SOFT Target Update

DQN с SOFT Target Network - веса обновляются плавно с использованием экспоненциального сглаживания.

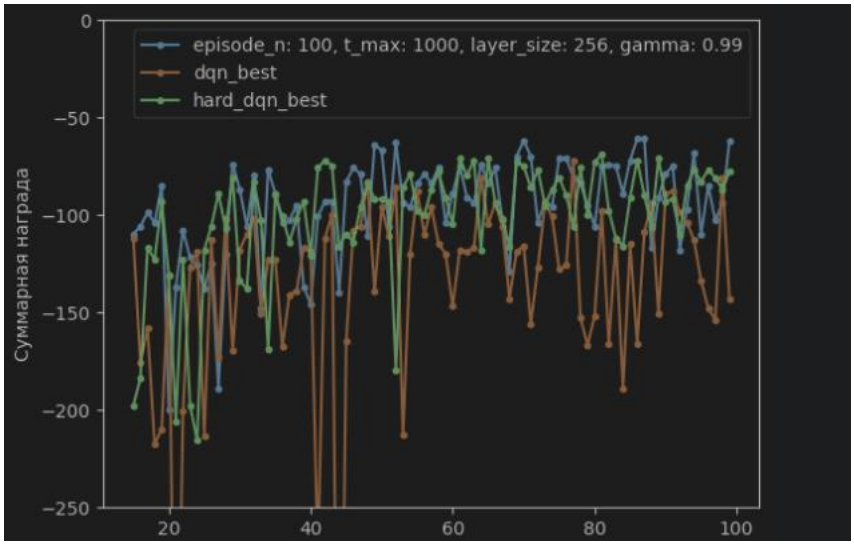
```
# Обновление целевой сети (SOFT Target Networks)
for target_param, param in zip(self.target_q_function.parameters(), self.q_function.parameters()):
    target_param.data = self.tau * param.data + (1.0 - self.tau) * target_param.data.detach()
```

Рассмотрим сводную таблицу результатов экспериментов, обратите внимание, что значения взяты на последней итерации mean_total_reward, и last_total_reward:

	episode_n	t_max	layer_size	gamma	mean_total_reward	last_total_reward
11	100	1000	256	0.99	-116.030000	-62.0
19	200	1000	64	0.99	-119.305000	-70.0
17	200	500	256	0.99	-103.170000	-71.0
23	200	1000	256	0.99	-107.945000	-75.0
27	300	500	128	0.99	-105.220000	-75.0
15	200	500	128	0.99	-106.335000	-77.0
29	300	500	256	0.99	-101.846667	-78.0
35	300	1000	256	0.99	-109.386667	-79.0
25	300	500	64	0.99	-94.606667	-83.0

Отметим, что модель стабилизируется за еще меньшее кол-во эпох = 100, но по прежнему t_max, layer_size, gamma максимальны, хочется отметить, что last_total_reward стала еще больше -62

По графику видно, что SoftDQN незначительно, но лучше HardDQN.



Задание # 2.3 DDQN

Два сети: использует две нейронные сети - для выбора действий и оценки их эффективности.

Минимизация завышения: разделяет выбор и оценку действий, чтобы избежать завышения Q-значений.

Отложенное обновление: обновляет оценочную сеть реже для стабильности обучения.

Реализация:

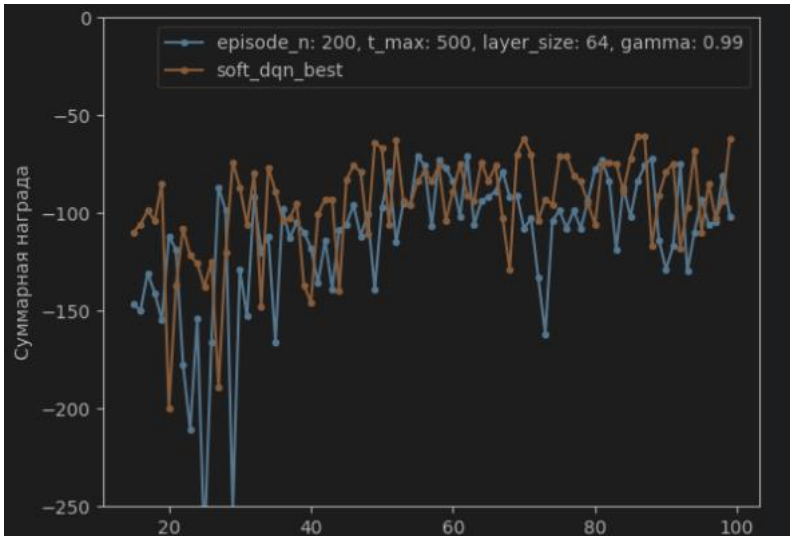
```
        if len(self.memory) > self.batch_size:
            batch = random.sample(self.memory, self.batch_size)
            states, actions, rewards, dones, next_states = map(torch.tensor, list(zip(*batch)))

            with torch.no_grad():
                best_actions = torch.argmax(self.q_function(next_states), dim=1)
                targets = rewards + self.gamma * (1 - dones) * self.target_q_function(next_states)[
                    torch.arange(self.batch_size), best_actions]

            q_values = self.q_function(states)[torch.arange(self.batch_size), actions]

            loss = torch.mean((q_values - targets.detach()) ** 2)
            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()

            if self.epsilon > self.epsilon_min:
                self.epsilon -= self.epsilon_decrease

    def update_target_network(self):
        self.target_q_function.load_state_dict(self.q_function.state_dict())

    def train(self, env, episode_n=100, t_max=500, target_update=10):
        rewards = []
        for episode in range(episode_n):
            total_reward = 0
            state = env.reset()
            for t in range(t_max):
                action = self.get_action(state)
                next_state, reward, done, _ = env.step(action)
                total_reward += reward
                self.fit(state, action, reward, done, next_state)
                state = next_state
                if done:
                    break
                if t % target_update == 0:
                    self.update_target_network()   You, 2 minutes ago • Uncommitted changes
```

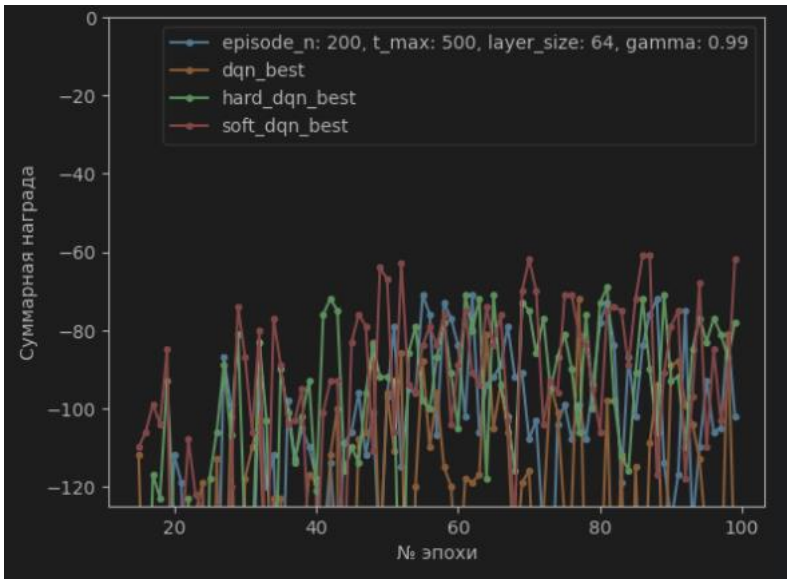
Рассмотрим сводную таблицу результатов экспериментов, обратите внимание, что значения взяты на последней итерации mean_total_reward, и last_total_reward:

	episode_n	t_max	layer_size	gamma	mean_total_reward	last_total_reward
13	200	500	64	0.99	-115.085000	-62.0
25	300	500	64	0.99	-108.260000	-80.0
21	200	1000	128	0.99	-101.275000	-80.0
29	300	500	256	0.99	-101.030000	-84.0
11	100	1000	256	0.99	-121.280000	-84.0
5	100	500	256	0.99	-116.810000	-86.0
3	100	500	128	0.99	-130.720000	-87.0
9	100	1000	128	0.99	-121.370000	-89.0
19	200	1000	64	0.99	-121.365000	-90.0

Неоднозначные результаты t_max впервые поменьше = 500 и layer_size=64, но эпох больше =200, хочется отметить, что last_total_reward осталась такой же - 62

По графику видно, что SoftDQN незначительно, но лучше DDQN.





Вывод: лучшая модель SoftDQN – стабильнее других и лучше по метрикам.