DRL Course 2023 Домашнее задание 7

Отчет по выполнению домашнего задания, Nikita Sorokin

Сравнение алгоритмов для среды Pendulum-v1

CEM

Пусть $\pi^\theta : \mathbb{R}^n \to \mathbb{R}^m$ - нейронная сеть.

В цикле по n для $n \in \overline{1, N}$:

(Policy evaluation) В соответствии с политикой

$$\pi_n(s) = [\pi^{\theta_n} + Noise(\varepsilon)]_A,$$

получим K траекторий θ_k и награду $G(\tau_k)$. Оценим матожидание как:

$$\mathbb{E}_{\pi_n}[G] \approx V_n := \frac{1}{K} \sum_{k=1}^{K} G(\tau_k)$$

(Policy improvement) Выбираем элитные тракектории, как

$$T_n = \{\tau_k : k \in \overline{1, K} : G(\tau_k) > \gamma_q\}, \quad \text{где } \gamma_q \text{ - квантиль уровня } q.$$

Определяем лосс:

$$Loss(\theta) = \frac{1}{|T_n|} \sum_{(a|s) \in T_n} ||\pi^{\theta_n}(s) - a||^2$$

Обновляем θ градиентным спуском и уменьшаем ε.

Обучение: (30 минут)

Подобранные гиперпараметры:

```
episode_n = 125
trajectory_n = 100
trajectory_len = 200
q_param = 0.8
```

```
agent.optimizer = torch.optim.Adam(agent.parameters(), lr=1e-1)
```

При обучении разрешаем всего 2 действия 2 и -2 из всего отрезка действий [-2, 2]. Это ограничение позволяет маятнику научиться уверенно раскачиваться. За эту функцию отвечает условие discrete_action = True.

Замечание: При выполнении дз 2 обучение проходило в 3 этапа:

1. Учимся с большим количеством шума, причем разрешаем всего 2 действия 2 и -2 из всего отрезка действий [-2, 2]. Это ограничение позволяет маятнику научиться уверенно раскачиваться. За эту функцию отвечает условие discrete_action = True.

2. Оставляем шум, возвращаем возможность выполнять все действия в отрезке [-2, 2]. Этот этап позволяет научиться маятнику выбирать действия, когда он проходит положение неустойчивого равновесия наверху.

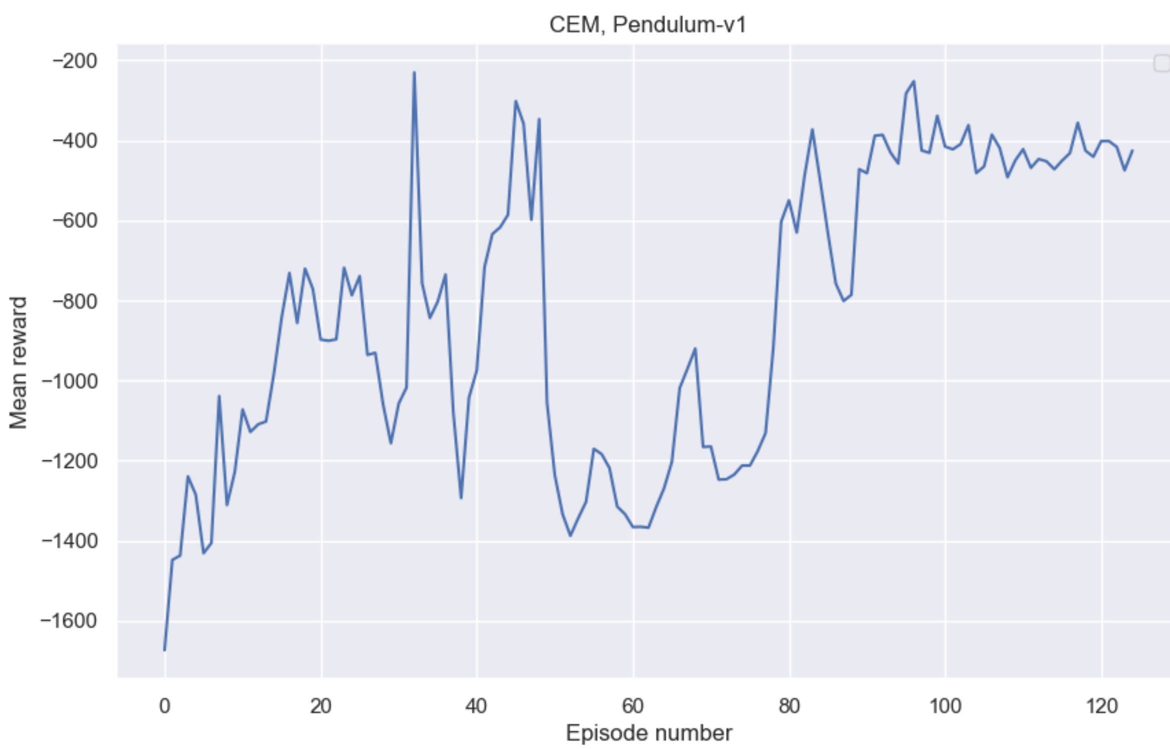
3. Убираем шум, разрешаем действия 2 и -2. К тому же теперь используем условие autosave = True, которое не позволяет модели учиться если mean_total_reward полученный в текущем эпизоде меньше предыдущего. Этот этап позволяет уверенно управлять маятником, когда тот находится наверху и пытается устоять.

Но для честности сравнения алгоритмов обучение проводится в 1 этап:

1. Действия разрешается всего 2 штуки: 2 и -2. Размеренное использование шума $\varepsilon(n) = 1/\sqrt{n+1}$.

График обучения:

In [1]:
```
from IPython.display import display, Image
display(Image(filename="pics/cem.png"))
```



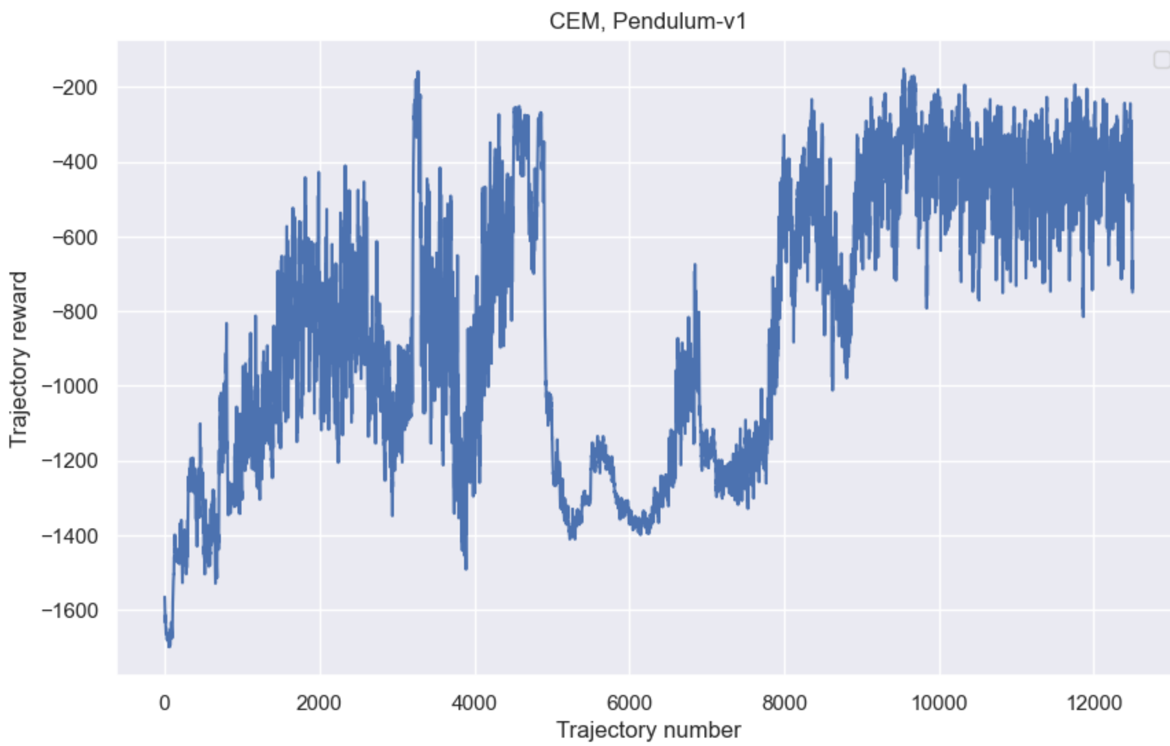
Валидация:

mean total reward on 100 validation trajectories: -441.301006123251

Из-за длительности обучения и нестабильности ограничимся только этим результатом:

Использую экспоненциальное сглаживание, преобразуем график истории обучения

```
In [2]: display(Image(filename="pics/cem_smoothed.png"))
```



При меньшем коэффиценте сглаживания награда в конце становится еще меньше. Поэтому ограничимся таким уровнем сглаживания.

DQN

Задаем структуру аппроксимации Q^θ, начальные вектор параметров θ, вероятность исследования среды $\varepsilon = 1$.

Для каждого эпизода k делаем:

Пока эпизод не закончен делаем:

- Находясь в состоянии S_t совершаем действие $A_t \sim \pi(\cdot|S_t)$, где $\pi = \varepsilon\text{-greedy}(Q^\theta)$, получаем награду R_t переходим в состояние S_{t+1}. Сохраняем $(S_t, A_t, R_t, S_{t+1}) \to Memory$
- Берем $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^n \leftarrow Memory$, определяем целевые значения

$$y_i = \begin{cases} r_i, & \text{если } s'_i \text{ -терминальное}, \\ r_i + \gamma \max\limits_{a'} Q^\theta(s'_i, a'), & \text{иначе} \end{cases}$$

функцию потерь $Loss(\theta) = \frac{1}{n}\sum\limits_{i=1}^{n}\left(y_i - Q^{\theta}(s_i, a_i)\right)^2$ и обновляем вектор параметров

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} Loss(\theta)$$

- Уменьшаем ε

Для использования этого алгоритма в средах с непрерывным пространством действий воспользуемся дискретизацией. Договоримся, что агент может выполнять только 2 действия: -2 и 2. Функция get_action() агента будет выдавать номер действия (0 или 1). При создании траектории после использования функции get_action() будем преобразовывать ее вывод в действие -2 или 2 (0 переходит в -2, 1 в 2).

Реализация:

```
...
actual_action = np.array([int(-2 + 4 * action)])
...
```

Обучение: (5 мин)

Используется стандартная версия алгоритма (без Soft/Hard Target модификаций). Проблем связанных с автокорелляцией не возникает, поскольку функция наград непрерывна по состоянию для среду Pendulum-v1:

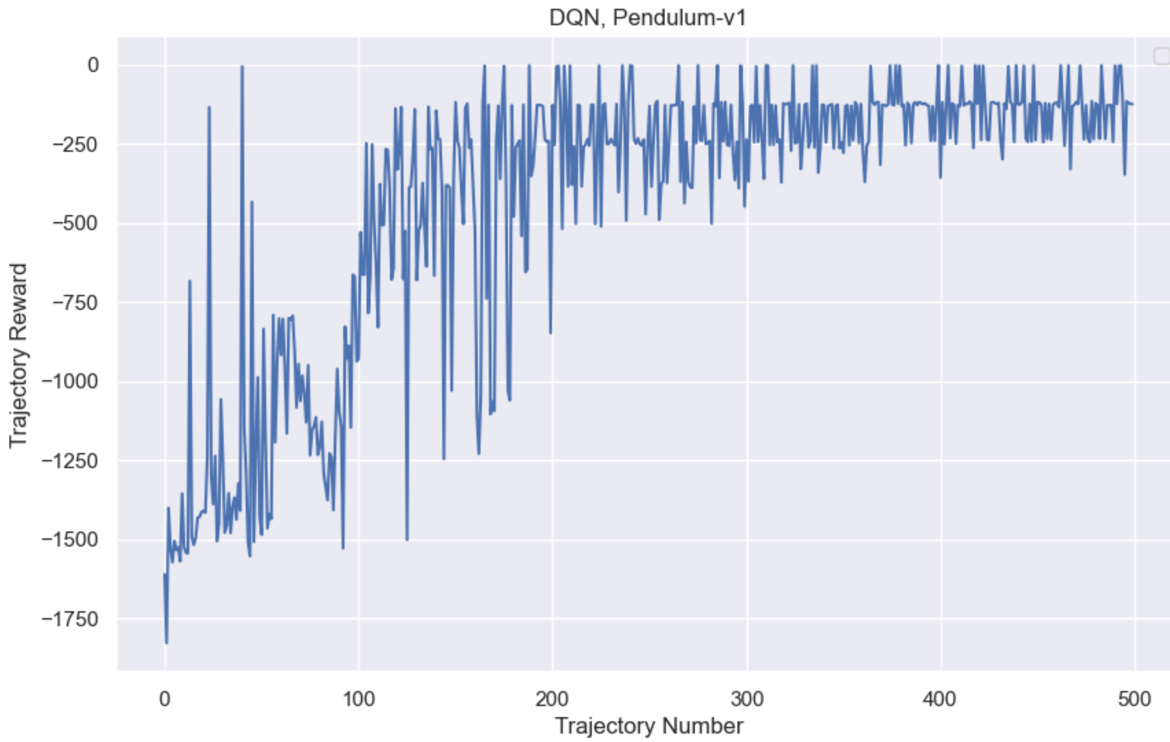
$$r = -\left(\theta^2 + 0.1 \cdot \frac{\partial^2 \theta}{\partial t^2} + 0.001 \cdot u\right)$$

Выбранные гиперпараметры:

```
episode_n = 500
trajectory_len = 200

dqn_agent.lr = 1e-4
dqn_agent.epsilon_decrease = 0.005
```

In [3]:
```
display(Image(filename="pics/dqn.png"))
```

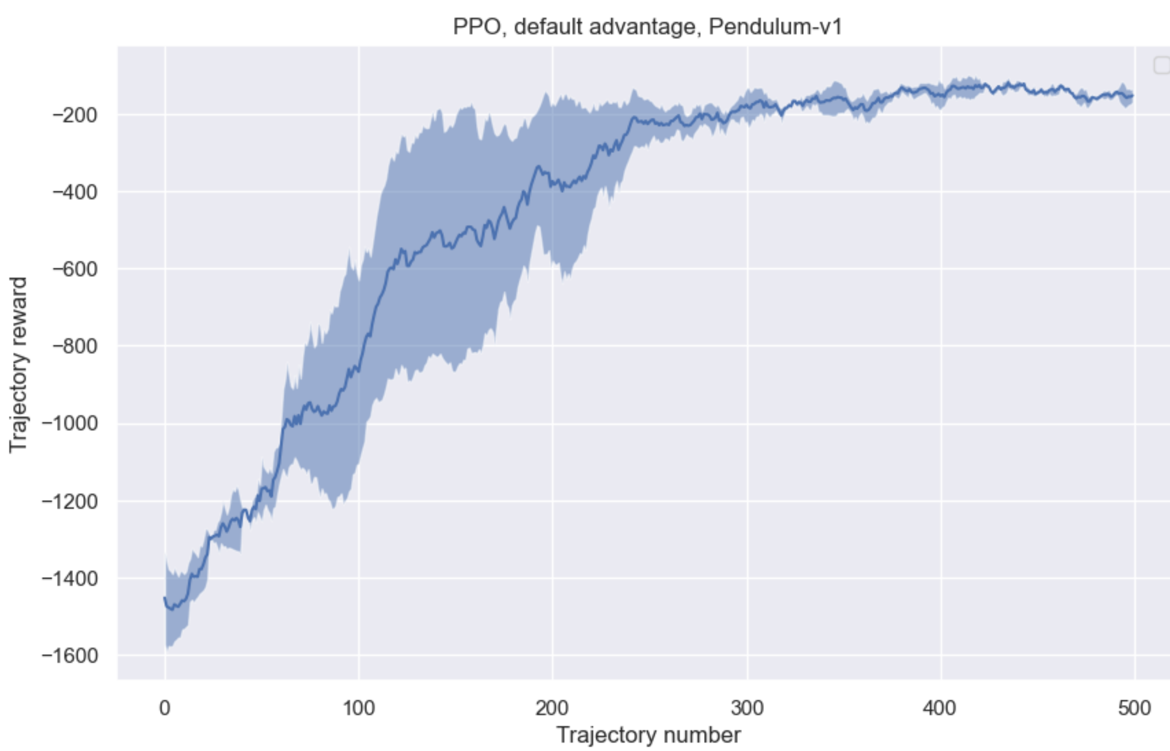


Валидация:

DQN validation_score: -149.3915106810389

Повторим еще 2 раза и построим сглаженный график:

In [4]:
```
display(Image(filename="pics/dqn_smoothed.png"))
```



Обучение проходит очень быстро, а результат получается на удивление хорошим!

PPO

Инициализируем политику $\pi^\eta(a|s)$ и $V^\theta(s)$ нейронными сетями. Устанавливаем η_0 и θ_0

В цикле по k для $k = \overline{1, K}$:

- По политике π^η получаем траекторию (или несколько) $\tau = (S_0,\ A_0,\ \ldots\ S_T)$. Считаем $(G_0,\ \ldots\ G_{T-1})$.
- Определяем лосс как:

$$Loss_1(\eta) = -\frac{1}{T}\sum_{t=0}^{T-1}\min\left\{\frac{\pi^\eta(A_t|S_t)}{\pi^{\eta_k}(A_t|S_t)}A^{\theta_k}(S_t, A_t), g_\varepsilon(A^{\theta_k}(S_t, A_t))\right\}$$

$$Loss_2(\theta) = \frac{1}{T}\sum_{t=0}^{T-1}A^\theta(S_t, A_t),$$

где $A^\theta(S_t, A_t)$ =

- $V^\theta(S_t) - G_t$ - advantage, полученный из Монте Карло оценки Q-функции
- $R_t + \gamma V^\theta(S_{t+1}) - V^\theta(S_t)$ - advantage, полученный из уравнений Беллмана для Q-функции

и обновляем параметры параметры нейронных сетей:

$$\eta_{k+1} \leftarrow \eta_k - \alpha_1\nabla_\eta Loss_1(\eta_k), \quad \theta_{k+1} \leftarrow \theta_k - \alpha_2\nabla_\theta Loss_2(\theta_k)$$

Обучение: (4 мин)

Используется стандартная версия алгоритма для непрерывного одномерного пространства действий.

Для обучения были подобраны следующие гиперпараметры:

```
episode_n = 25
trajectory_n = 20

gamma = 0.9
batch_size = 128
epsilon = 0.2
epoch_n = 30
pi_lr = 1e-4
v_lr = 5e-4
```

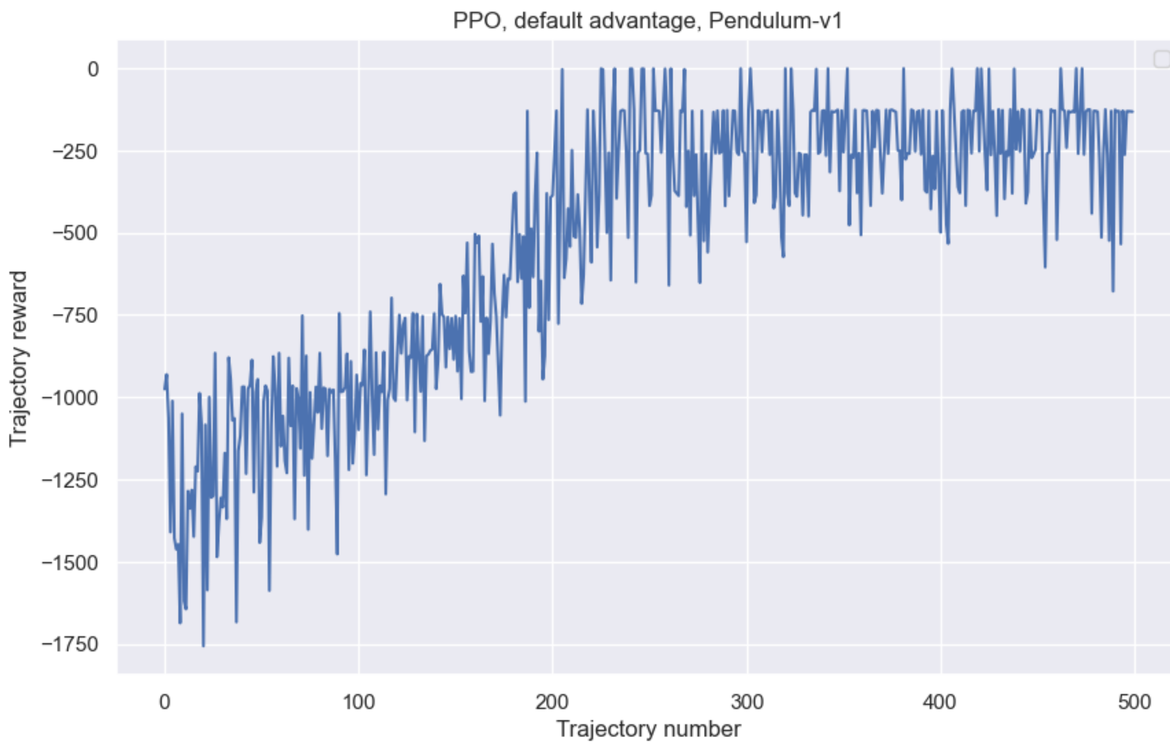
Архитектуры нейронных сетей:

```
self.pi_model = nn.Sequential(nn.Linear(self.state_dim, 128), nn.ReLU(),
                              nn.Linear(128, 128), nn.ReLU(),
                              nn.Linear(128, 2 * self.action_dim),
nn.Tanh())
```

```
self.v_model = nn.Sequential(nn.Linear(self.state_dim, 128), nn.ReLU(),
                             nn.Linear(128, 128), nn.ReLU(),
                             nn.Linear(128, 1))
```

График обучения:

In [5]:
```
display(Image(filename="pics/ppo.png"))
```

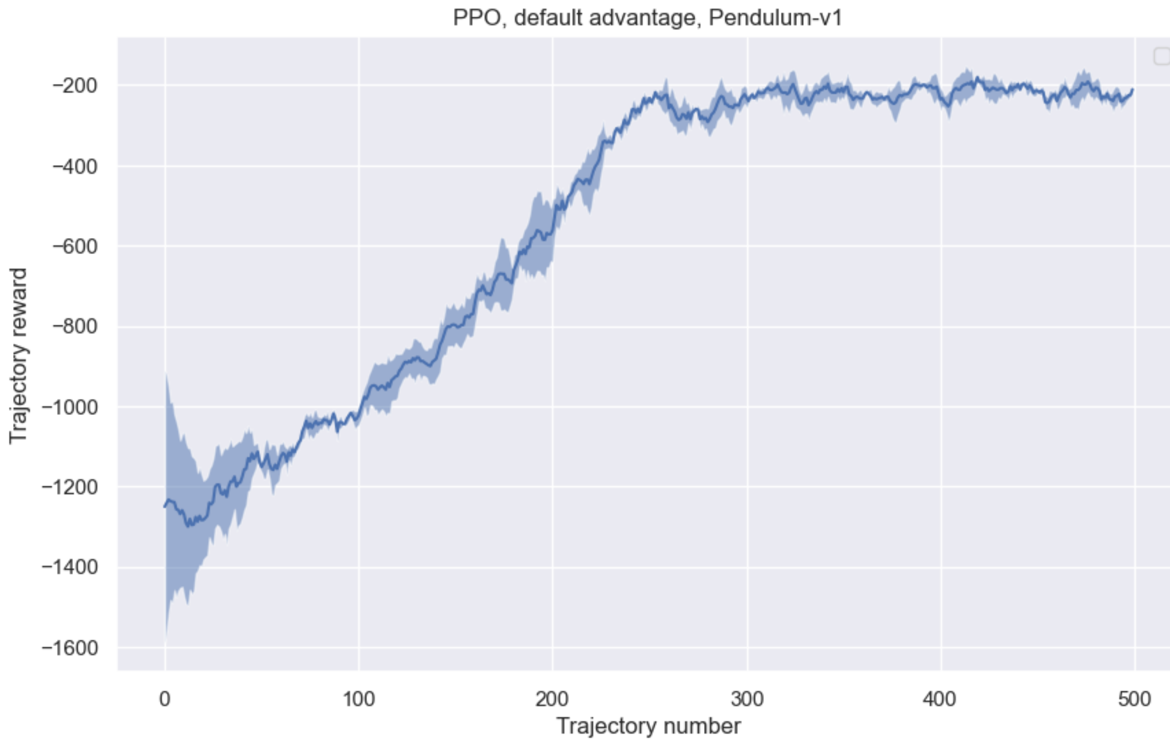


Валидация:

PPO, default advantage, validation_score: -252.77200815790158

Повторим еще 2 раза и построим сглаженный график:

In [6]:
```
display(Image(filename="pics/ppo_smoothed.png"))
```



SAC

Инициализируем политику π^{η} и Q^{θ_i}, $Q^{\theta_i'}$ нейронными сетями. Далее на каждом эпизоде выполняем:

- Добавляем в буффер $(S_t, A_t, R_t, D_t, S_{t+1}) \to M$
- Достаем батч из буффера $\{(s_j, a_j, r_j, d_j, s_j')\}^n \leftarrow M$,

$$y_j = r_j + \gamma(1 - d_j)\left(\min_{i=1,2} Q^{\theta_i'}(s_j', a_j') - \alpha log\pi(a_j'|s_j')\right)$$

Определяем лосс функции:

$$L_i(\theta_i) = \frac{1}{n}\sum_{j=1}^{n}(y_j - Q^{\theta_i}(s_j, a_j))^2$$

$$L_3(\eta) = \frac{1}{n}\sum_{j=1}^{n}(\min_{i=1,2} Q^{\theta_i}(s_j, a_j^{\eta}) - \alpha log\pi^{\eta}(a_j^{\eta}|s_j))$$

И обновляем параметры градиентным спуском.

Обучение: (6 мин)

Выбранные гиперпараметры:

```
episode_n = 200
```

```
gamma=0.99
```

```
alpha=1e-3
tau=1e-2
batch_size=64
pi_lr=1e-3
q_lr=1e-3
```

Архитектуры нейронный сетей:

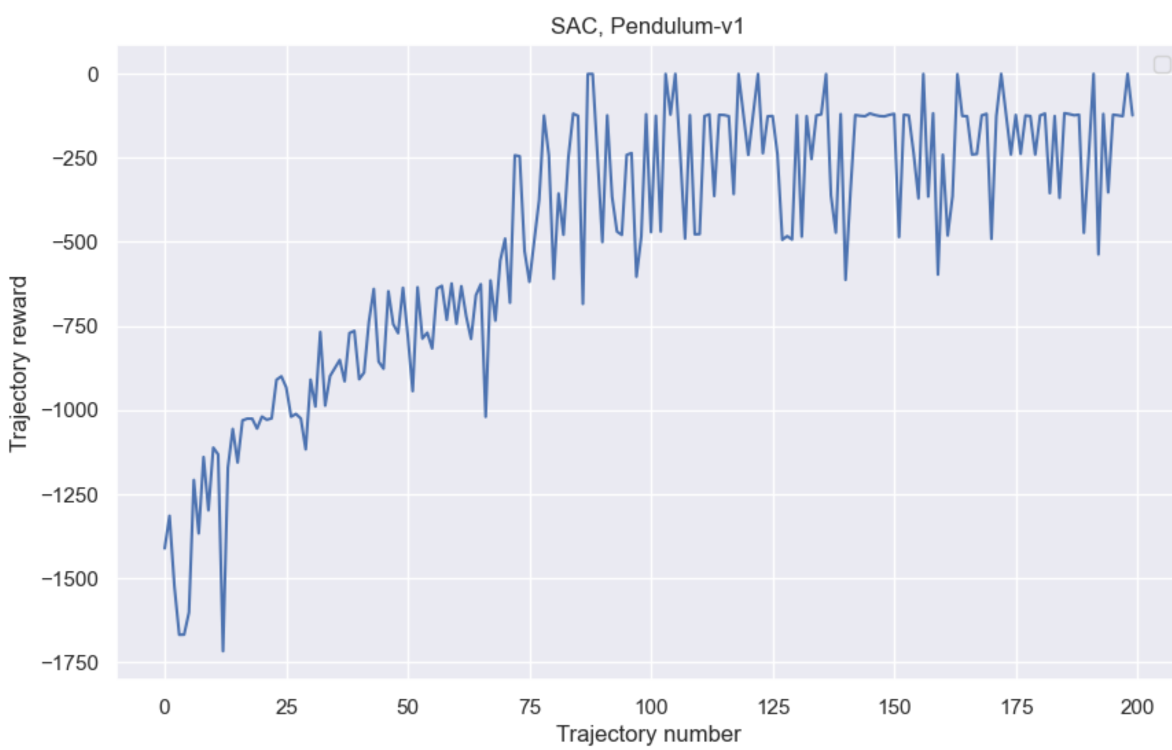
```
self.pi_model = nn.Sequential(nn.Linear(state_dim, 128), nn.ReLU(),
                              nn.Linear(128, 128), nn.ReLU(),
                              nn.Linear(128, 2 * action_dim), nn.Tanh())

self.q1_model = nn.Sequential(nn.Linear(state_dim + action_dim, 128),
nn.ReLU(),
                              nn.Linear(128, 128), nn.ReLU(),
                              nn.Linear(128, 1))

self.q2_model = nn.Sequential(nn.Linear(state_dim + action_dim, 128),
nn.ReLU(),
                              nn.Linear(128, 128), nn.ReLU(),
                              nn.Linear(128, 1))
```

График обучения:

In [7]:
```
display(Image(filename="pics/sac.png"))
```

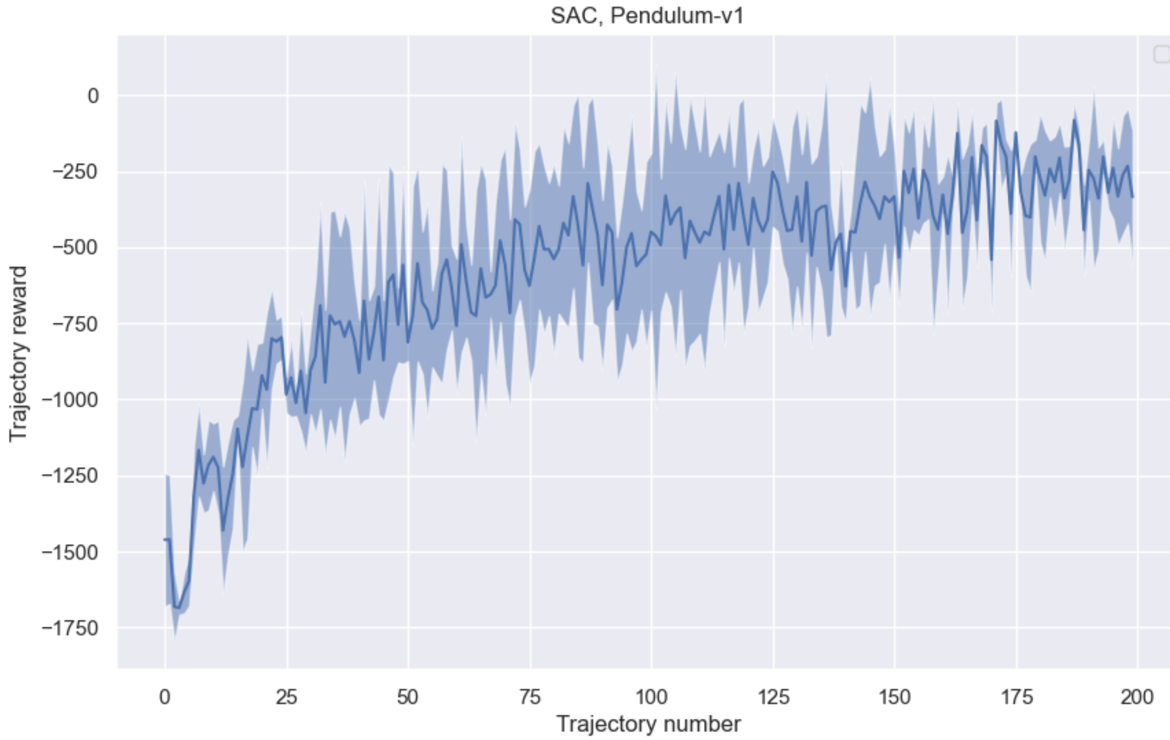


Валидация:

SAC, validation_score: -235.55826693948117

Повторим еще 2 раза и построим сглаженный график:

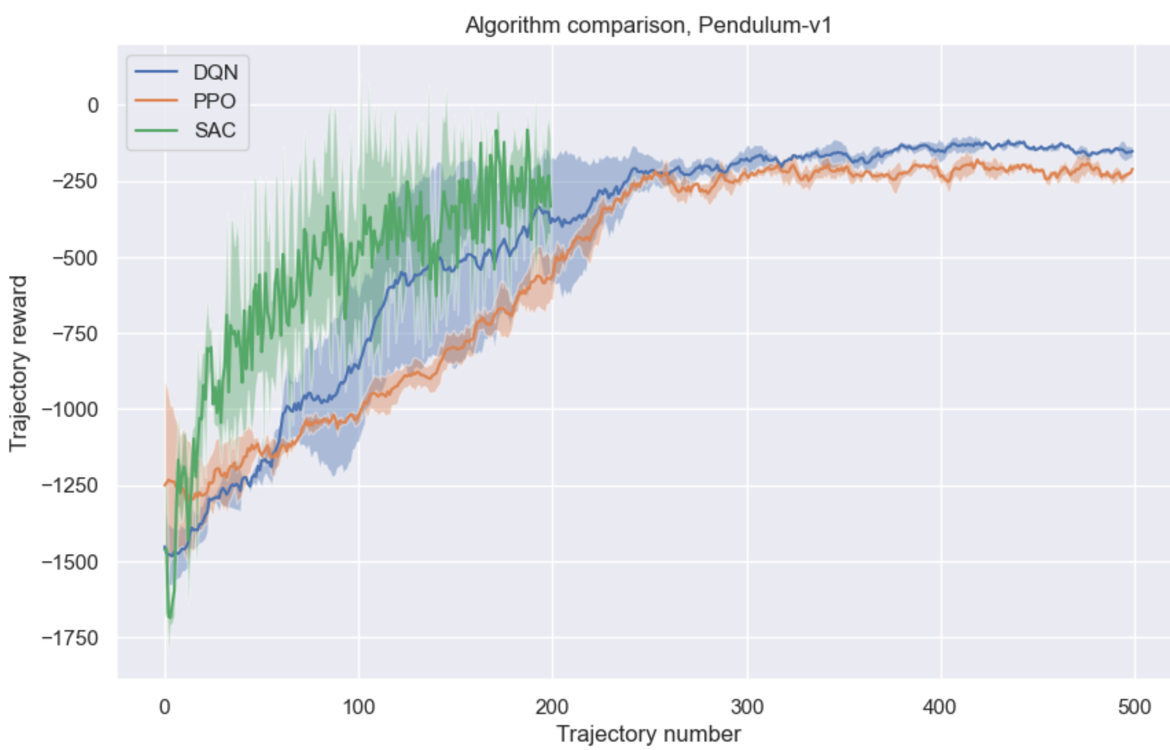
In [8]:
```
display(Image(filename="pics/sac_smoothed.png"))
```



Полученные результаты:

Уже понятно, что CEM использует сильно больше траекторий. Изобразим кривые обучения DQN, PPO и SAC на одном графике:

In [9]:
```
display(Image(filename="pics/dqn_ppo_sac_comparison.png"))
```



Вывод:

Наилучшим алгоритмом по результату на валидации оказался стандартный DQN, использующий 2 действия: -2 и 2 из отрезка [-2, 2]. Результат на валидации алгоритма DQN оказался -150.

Алгоритмы PPO и SAC показывают значение на валидации не больше -200 независимо от количества эпизодов обучения. Возможно для улучшения этого результата следует аналогично сделать пространство действий дискретным, оставив 2 значения (-2 и 2). В свою очередь моя реализация CEM использует эту идею дискретизации действий, однако обучается хуже и получает значение на валидации не выше -400.

По времени обучения лидируют алгоритмы DQN и PPO, обучаясь по 5 минут. Однако SAC не сильно отстает и обучается за 6-7 минут. Обучение CEM для получения указанного выше результата занимает 30 минут.

Алгоритму SAC требуется наименьшее количество сгенерированных траекторий для успешного обучения - всего 200 штук. DQN и PPO используют по 500 траекторий. CEM потребовал 12500 траекторий, однако это связано со спецификой алгоритма. CEM является эволюционным алгоритмом и не задействует всего того теоретического аппарата Q-функций и Policy Gradient теорем, которые задействуют другие алгоритмы.